AN ENHANCEMENT AND IMPROVEMENT OF A PROTOTYPE DISTRIBUTED SYSTEM BASED ON ELEMENTS OF AN INTEGRATION ARCHITECTURE

by
Kunal R. Shah

A Thesis
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Master of Science

Department of Computer and Information Sciences

January, 1993

APPROVAL PAGE

An Enhancement and Improvement
of a Prototype Distributed System Based on
Elements of an Integration Architecture

by

Kunal R. Shah

Dr. Wilhelm Rossak
Assistant Professor Software Engineering, NJIT

Dr. Peter Ng
Chairperson
Department of Computer and Information Sciences, NJIT

BIOGRAPHICAL SKETCH

Author: Kunal R. Shah

Degree: Master of Science in Computer Science

Undergraduate and Graduate Education:

- Master of Science in Computer Science),
  New Jersey Institute of Technology, Newark, NJ, 1993

- Bachelor of Engineering in Computer Engineering,
  The University of Bombay, Bombay, India, 1989

Major: Computer Science

ABSTRACT

An Enhancement and Improvement
of a Prototype Distributed System Based on
Elements of an Integration Architecture

by
Kunal R. Shah

The concepts and results presented in this thesis are related to Integrated System Development. It provides introduction to Generic System Integration Framework (GenSIF). And hence its three principal components, Domain Analysis, Integration Architectures and Enabling Technology. It addresses certain issues of distributed processing relating to systems integration.

The primary objective of this thesis is to develop/improve a prototype by applying concepts and ideas presented in GenSIF, with an example channel based building block integration architecture as an example. This prototype was developed with the objective of studying the effect of system intregration framework in mind while working on an application. It would help in comparing the traditional software life cycle with the life cycle of software development under the effect of the framework.

Sun RPC implementation of Remote Procedure Calling technique has beenused to realize the integration of physically distributed prototypes of a system/subsystem(s). Advantages and disadvantages of using RPCs have also been presented. This work also looks into another communication fabric, viz. ANSA, to analyze its feasibility for use in systems integration.

This thesis is dedicated to
my parents

ACKNOWLEDGEMENT

The author wishes to express his sincere gratitude to his advisor, Dr. Wilhelm Rossak, for his guidance, friendship, and moral support throughout this work.

Special thanks to all the members of Sytems Integration research group for their valuable suggestions and criticism.

The author is grateful to Saraswathy Devi for her expert advice on ANSAware.

And finally, a thank you to all the members of my family, who provided inspiration and support in difficult phases of this thesis.

TABLE OF CONTENTS

Chapter Page

1 INTRODUCTION TO SYSTEMS INTEGRATION 1

2 GENERIC SYSTEMS INTEGRATION FRAMEWORK 5

3 FUNCTIONAL OVERVIEW OF THE CS DEPARTMENT INFORMATION SYSTEM 25

4 THE ROLE OF RPC IN THE IMPLEMENTATION 33

5 THE ROLE OF ANSAware IN THE IMPLEMENTATION 44

6 REALIZATION AND/OR EVALUATION OF SUGGESTED IMPROVEMENTS 50

7 COMPARISON OF THE TWO IMPLEMENTATIONS 63

APPENDIX 69

REFERENCES 147

LIST OF FIGURES

Figure Page

2.1 The Components of the GenSIF Framework. 7

2.2 Meta-Model for System Develpment 7

4.1 Remote Procedure Call (RPC) Model 34

4.2 Files Involved in Generating Sun RPC Program 37

4.3 Inter Building Block Coomunication 39

6.1 Entity Relationship Diagram for EIS. 51

6.2 Level 1: Data Flow Diagram 52

6.3 Level 2: Data Flow Diagram Explosion of FDLBB 53

6.4 Level 2: Data Flow Diagram Explosion of PLBB 54

CHAPTER 1

INTRODUCTION TO SYSTEMS INTEGRATION

1.1 Information Management in an organization.

Information is recognized as an organization's most valuable asset. In many forms, information is spread throughout an organization and kept within divisions, branches, departments, work groups and individual user's files. The primary goal of every organization, department or a group is to build, operate on and maintain the huge amount of information. Each organization or a group may maintain different kinds of information. It can be an organization's employees payroll information, or an organization's client-account and their business transactions information. It can also be some graphical form of information for statistical representation or in a form of a report/chart.

In order to make this task of information management easier, they are automated into their respective softwares. This realization of unautomated systems into the automated systems, is what we refer to as a SYSTEM or a PROJECT.

In today's automated world, information is usually stored and processed on a wide variety of computer systems. The software systems/projects to manage/maintain this information could have been built by more than one contractors/groups at different times as a relatively independent projects. But they still need to work together in a globally integrated fashion. This issue is discussed in detail in the next section.

1.2 Need for Systems Integration Framework.

Integration is a term commonly used by system planners. Although it always conveys some aspect of unity, integration can refer to many things. Integration in its technical context can mean physical integration (of functions into one device) or a functional integration. It also can mean the sharing of information between applications that provide related services. The ability to move, and therefore, share information in intelligent forms between users of the same system is another form of integration. Integration, more importantly, also can be used in a system sense, that is, for communication between systems and system applications that share common communication protocols. This is in turn allows users to broaden their community to include users and information sources on other systems. Integration also can include the establishment of a consistent user interface across several applications in a class having some common nature.

Thus over the past years system development has become increasingly demanding. Due to generic requirements of various software products and to the popularity of distributed systems, software engineers and designers have started to think about integrating various, independent software systems, such that they would work in a globally integrated fashion. This may involve heterogeneous environments of computer systems. And due to this varying degrees of reliability, accessibility and availability between software products and their computing environments, need was felt for the design of a heterogeneous software/hardware environment to cooperate as a whole in a LOOSELY COUPLED DISTRIBUTED configuration. We have to recognize the fact that current methodologies and development strategies are unable to deal with very large distributed systems.

Following are some of the critical factors, as described by Dr. Wilhelm Rossak in [1], which can focus more on the urge for a systems integration framework.

Software engineers have realized that,

A. Development support goes beyond software support.

The development process of a complex system should support not only software but also hardware and organization. We need to shift our focus from the single aspect of a system, since we need well-disciplined control and communication between groups and projects, and interaction between problem (application) domain and solution.

B. More than a single client is involved.

The client for our product is neither a single person nor a homogeneous group of people. We will have to live with the fact that fairly clear requirements can be specified only for limited projects, and that it is infeasible to freeze and to document in advance the needs for a set of interrelated projects which cover a full application domain.

C. More than single producer is involved.

All the above mentioned factors become critical if the system is developed by multiple groups or by multiple companies and contractors. Different groups working on projects which are parts of an integrated system require extra need for communication and control and must be guided by a framework which underlies long-term goals and strategies for developing the system.

D. More than a single project is involved.

Different projects can be run by applying different methodologies and in different time frames. In a dynamic environment we will even have to expect that some projects are specified after other projects are nearly completed, or in the execution and maintenance phase.

Thus, there is no chance to organize a huge and centralized effort in the very beginning of system development to coordinate all possible projects.

The above mentioned problems can be solved by applying an approach of meta-level control. The next chapter describes in more detail the generic framework for integrated systems development (3).

1.3 What is meant by Mega-systems?

Today software systems are used in nearly every application domain and their size and complexity keeps growing. While the variety of application domain has made it necessary to adapt our methodologies to more and more environments, if we were able to do so, growth has scaled up the difficulties in system development and has changed the characteristics of the development process.

The traditional strategies focus mainly on development of one system for a specific problem, instead of providing means to develop an integrated group of systems which provides a coordinated domain wide solution. Mega-systems cover the needs of a full application domain in a structured way. An application domain is considered to be a comprehensive, internally coherent, relatively self-contained area or business enterprise supported by software systems (4).

Mega-systems cover at least one of the characteristics of being constructed from more than one system, by more than one developer group and for more than one consumer. They include huge systems, package systems, and systems of systems.

All the views expressed here on Mega-systems and system of systems, are from the research papers by Tamar Zemel and Dr. Wilhelm Rossak,(3) and (4).

CHAPTER 2

GENERIC SYSTEMS INTEGRATION FRAMEWORK.

2.1 Meta-Level of Control.

In a dynamic systems integration environment we will have to expect that some projects are specified after others are nearly completed, or are in execution, or in maintenance phase. If we use traditional systems development approach in such a situation, there is no chance to organize a huge and centralized, productive effort in the very beginning of system development and to coordinate all possible projects.

Therefore to guarantee the integration, optimization, and consistent user semantics of the ultimate product, management control over the autonomously running projects is suggested. The basic concept is to introduce a meta-level of control and management as a complement for the project-level.

This concept of meta-level control brings us to the aspect of analysis and specification of an application domain, which encompasses the application environment of an integrated system and to derive/decide on suitable integration architecture. It also motivates us to consider the various enabling technologies and decide on the best possible alternative for achieving interoperability over the application domain. Thus this chapter will mainly talk about the Generic Systems Integration Framework and the introduction to a channel based integration architecture, which is the basis of implementation for the Distributed System that was developed.

2.2 GenSIF and its components.

Mega-systems and especially systems of systems, need integration support as a built in feature of their development support environment to enable a planned and structural approach to managing independent developer and customer groups to finally come up with a solution of not only interdependent but interoperable systems. GenSIF, a generic systems integration framework, is such an approach oriented towards a solution for a specific application domain (1), (2), (3), (4).

As we all know, systems development is project oriented. GenSIF, strives to leave the project teams as much freedom as possible but provides domain wide integration measures for co-ordination of projects. Each project can organize its own strategies and development efforts, but it should also comply to the integration framework in order to guarantee the interoperability of developed systems.

This framework does not ask for prior knowledge of all the (future) parts of the mega-system nor all requirements of the domain. The goal here is to be concept driven rather than welcoming various problems due to an uncoordinated technology driven approach.

A system planner, following GenSIF, certainly makes use of the technology and its tools, but it is based on a firm set of domain wide conceptual models and hence explicitly spelled out prerequisites and decisions.

Figure 2.1 and 2.2 illustrates various components of GenSIF and their relationships to the systems development phases at the project level. These three components as seen in figure 2.1 reflect different levels of abstraction and address different goals and needs during an integrated development process.

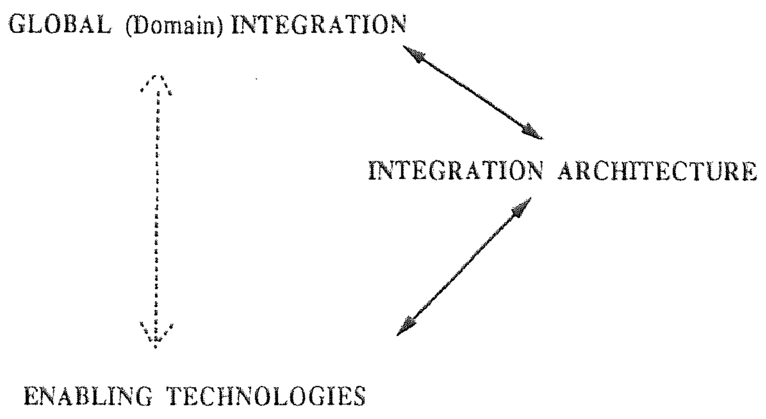

Figure 2.1 The Components of GenSIF Framework

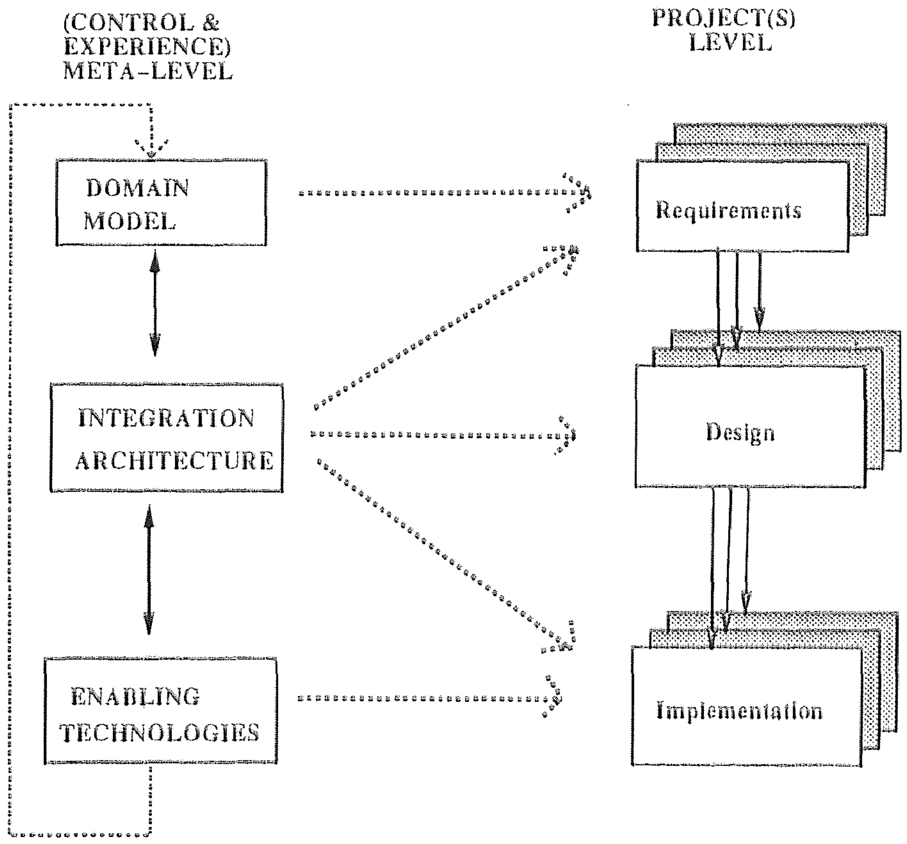

Figure 2.2 Meta–Model for Systems Development

Figure 2.2 illustrates the relationship of integrated system development to a traditioanl project driven approach. It is basically a meta-model for integrated system development process.

[1] Global (Domain) Integration.

- specifies the conceptual basis for the integration architecture. It deals with the concepts and semantics of an application domain and with the mapping of these concepts into the installed applications. This involves integration of information in a meta-data system and the specification of a generalized process model. These global activities relate to semantic integration and involve all analysis of the application domain in order to define a common model of the environment the system is going to serve.

Domain analysis not only provides a basis for systems integration, but it also is the main decisive factor in choosing/designing an integration architecture.

[2] Integration Architecture.

- is the core of GenSIF. It is a conceptual/structural model that bridges the gap between the outcome of domain analysis and the technological tool-level. It also is an infrastructure which provides the necessary utilities and components for the implementation of an application system on the basis of the conceptual model. It must fit the needs of an application domain.

[3] Enabling Technologies.

- are given by the tools and products that are required by the infrastructure of a generalized integration architecture, in order to develop and implement applications that will fill the abstract architecture with functionality and data.

This level apart from being concerned with state-of-the-art, should also provide views and suggestions for standards and developments in this area.

Domain analysis is the most important factor in determining the integration architecture and in addressing the other issues of global integration. Furthermore, it influences (via the integration architecture) the technological basis that is used to implement computer-based services in the application domain. Decisions made on the level of domain analysis and integration architecture affect not only existing applications and the currently planned global system structure, but pre-structure the environment for future software and hardware development.

The integration and control efforts on a meta-level above the development efforts for single projects constitute the components of GenSIF. They provide models and processes to handle strategic decisions and technical integration issues. From this meta-level of systems engineering, the components of GenSIF affect and guide the development of all system parts. A system part is an application system which provides a specific functionality in the application domain.

Therefore, the next section focuses on the various types of integration architectures and then explains in detail the channel based integration architecture. The concepts of this architecture form a basis for the design of the prototype Computer Science Department information system and its operations. The tool used here is the Sun RPC tool, for Remote Procedure Call, to communicate between the building blocks (to be explained in section 2.4) to achieve interoperability.

2.3 Types and Features of Integration Architectures.

An integration architecture is neither the description of a traditional development cycle nor an analysis and design method for traditional project oriented development. It is a conceptual model and a tool-box, describing the meta-level of control, design constraints, and the support for all development projects in a given application domain.

The domain analysis, feedback from enabling technologies, and some actions of global integration that are applied before the existence of an architectural framework - all influence the design/choice of an integration architecture. But once it is specified, it becomes a bulletin board which lists the rules and concepts for all the decision processes related to system development in a given domain. It provides a common understanding of the systems main structural attributes.

Since an integration architecture plays a major role in coordinating domain analysis and enabling technologies, it is a central element of decision making on a meta-level above the single project. Here are some of the types of integration architectures for large systems:

- Channel Based Systems,
- Systems with central data repository,
- Generic Systems and
- Object Oriented Systems.

The discussions on these types is beyond the scope of this document, but the next section describes an integration architecture, which we believe is a channel-based architecture.

However, all types of integration architectures can be decomposed into two main aspects:

o Conceptual/structural architectural model and o the technical infra-structure.

2.4 Channel Based Building Block Architecture

A channel based building block architecture presented in this chapter is based on the OSCA^{TM} architecture as published by Bellcore. However, the building block architecture is our own interpretation of OSCA architecture elements and represents in no way an authorized Bellcore document. OSCA is a trademark of Bellcore - Bell Communications Research. For an official publication see (6).

A channel based architecture is the one which provides guidelines for the communication of heterogeneous, distributed systems with goal of promoting interoperability and operability of software products. This is an implementation independent system design framework. The idea here is to provide a common software bus concept for promoting interoperability of software components which typically consists of large numbers of a programs, transactions, and data repositories. Interoperability here means the ability to intercommunicate heterogeneous software systems of a selected domain, irrespective of their suppliers, to provide access to data repositories and functionality of each of the systems to an authorized user. Operability is the ability to efficiently and cost effectively control and manage installation, administration, execution and user access of this loosely coupled collection of softwares to meet the performance, reliability, consistency, availability and security issues of the domain of these systems.

Our example architecture mainly relies on separation of concerns principles to achieve the goals described above. The following sections will describe in detail these principles.

2.4.1 Separation of Concerns Principle.

At the highest level of abstraction, a software system that conforms to this channel based architecture is seen as separated into three logical layers: Data Layer, Processing Layer (business operations), and a Human User Layer. As as is very obvious, this division is done based on functionality.

[1] Data Layer Functionality.

This layer has to provide the functionality to manage the semantic integrity, consistency and access of the data repositories.

[2] User Layer functionality.

The goals and tasks of a human user are translated by the user layer to appropriate processes and functionality in data, processing and user layers.

[3] Processing Layer Functionality.

This layer provides functionality for prescribed business operations and management processes of the domain.

2.4.2 Building Blocks.

Each of the logical layers is made up of one or more well-defined deployable, interoperable units containing the necessary functionality, called building blocks. The building blocks have a interface specification document that every requesting building block has to follow to have access to the deployable building blocks. These interface specification are well documented and are made available to the deployers and implementors. Any building block can communicate with any other building block that provides a function it requires.

A building block consists of a cohesive set of functions solely contained in one layer. Each of the building blocks should have following characteristics:

(a) installable and deployable together as a whole,

(b) should be able to interact with functions in other building blocks in a lossely coupled manner,

(c) should be collectively releasable, independently of functions in other building blocks.

The example architecture used here for the implementation of a Enhanced Information system for systems integration utilizes a building block approach to software product delivery in which each logical layer of the architecture is composed of many building blocks working together. Each building block is a set of computer programs, data schemas, and other related software which possesses well defined, coherent, functionality and interfaces. A description of a building block's interfaces unambiguously defines the functions it provides to other building blocks. A building blocks needs no knowledge of the internal structure of other building blocks. Thus, if the interface specifications are kept the same while updating the internal implementation of a building block, the deployer will not know about this change at all.

The architecture allows a building block to be both compatible with other building blocks and functionally substitutable. The building block should be able to interoperate with any other building block irrespective of its supplier. Also, its functionality, and the expression and use of that functionality (syntax,semantics,pragmatics) should be well-defined. For a building block to act as a substitute of another building block, it need not offer the exact same set of functionality, as the original building block.

To do its work, a building block utilizes hardware and software functions from the infrastructure provided by the operating systems, the communications network, database management systems, and other resources. This functionality may be shared among building blocks so long as such sharing does not result in any violation of building block principles. Many instances of a building block may exist, possibly on different computer systems. They exist for purposes such as load sharing, partitioning, reliability and availability.

2.4.3 Role of Infrastructure.

We can define an infrastructure as a platform, or a set of tools to enable execution, interoperability, operability of building blocks. Thus it is a backplane that allows building blocks to execute, operate individually and interoperate collectively. A uniform infrastructure is very critical for a uniform Integration Architecture. A uniform infrastructure is one that can operate as a cohesive whole, be provided by multiple vendors, and on which building blocks can be deployed so that they may fulfill their respective architectural requirements in an efficient, unified, and uniform manner.

2.5 Design of Building Blocks of an Application.

Here we will look into certain aspects of the architecture which are necessary to understand before actually separating the functionality of the system(s) into various layers and its building blocks.

2.5.1 Guideline-1.

It is desirable that a building block have low functional coupling with other building blocks and high functional cohesion within itself. These are the two main characteristics behind building block requirements.

Let me define what is meant by coupling and cohesion, so as to make things clear before moving on. Coupling is a degree of interdependence between two building blocks. Cohesion is the degree to which the functions within a building block are related to each other. Thus, coupling defines the inter-building block relationships unlike cohesion, which defines the intra-building block relationships. Thus from a building block architecture point of view what is desirable a balanced cohesion within a building block and a minimum of coupling between building blocks.

Minimizing the cohesion among the functions in different building blocks minimizes the coupling between the building blocks. Balancing the cohesion within a building block with the decoupling among building blocks leads to optimized building block definitions.

A set of functions are candidates to be in the same building block, if and only if :

a) they are in the same logical layer of the threearchitectural layers,

b) they are cohesive as a set, but decoupled from functions not in the set,

c) they will always be deployed together in the same recoverable domain,

d) collectively they can be released independently from functions not in the set.

Thus, as can be inferred from the points a) and b), they constrain the maximum size of a building block by limiting what categories of functions that can go together. And c) and d) above constrain the minimum size by including in the grouping all those functions that are deployable, releasable, and addressable together.

2.5.2 Guideline-2.

Considerations such as human characteristics, tasks, requirements, and participation (Authority/Priviledge) are central to improve the quality of building blocks. Even though the Processing and data layer building blocks are not directly concerned with the user interface, they are nonetheless concerned with supporting the human user's work processes. They must deliver the appropriate information, in the appropriate granularity, with appropriate response time, to ensure that those tasks can support the user's business activities.

2.5.3 Guideline-3.

It is desirable that any function that is not the infrastructure be examined whether it can be made generically available to multiple building blocks. If so, then it can be implemented as part of the infrastructure. This promotes more re-use of functions and decreases the complexity of building blocks.

2.6 Building Block Principles.

Before we go into detail explanation of these principles, lets make it clear that these principles do not apply among programs or transactions within a single building block. These principles apply among building blocks. For example, although two building blocks must be release independent, the programs within a single building block can be release dependent.

2.6.1 Release Independence.

To maximize deployment flexibility of building blocks, each instance of a building block must be able to be installed, upgraded, changed, activated without concurrently installing, upgrading, changing, or activating other instances of itself or other building blocks.

Release independence implies that:

1. the old functionality must be available throughout the upgrading period of time.
2. If a software application consists of multiple building blocks, it is up to the supplier to furnish documentation to the customer that details what configuration will yield the required functionality.

3. Agreed upon sets of consistent rules to provide nonconcurrent release installation across all building blocks is needed. These rules provide agreement by contracts on how release independence will be supported.
4. A upgraded building block may be required to continue to support previous functionality.

2.6.2 Infrastructure and Resource Independence.

Infrastructure and the underlying resources can be shared among building blocks so long as no building block is forced to violate any principles of the architecture. Typical resources such as files, DBMSs, icons, and display devices, are the physical or semi-physical objects that are made available and manipulated via the infrastructure functions.

Some implications of these are:

a. The communication network must not violate any principles of the architecture.

b. When a DBMS or a file system is used by more than one building block, it must allow support of the architecture.

c. If a resource, such as the operating system or a DBMS is shared, and it fails or otherwise becomes unavailable, it will impact all sharing building blocks.

2.6.3 No accessibility assumptions between building blocks.

To sustain an operable environment, an instance of a building block communicating with another building block instance must be able to respond to the unavailability of the target building block instance in a manner which preserves its own availability. A building block instance may be unavailable due to a variety of situations, including failure of the communications network, operational failure of the instance, incorrect building block version for an instance, or intentional compromise of the integrity of the system. A building block instance may respond to the absence of the target instance in wide range of ways from announcing to its invoker it cannot complete a task to seeking alternate building block instances providing required functionality to deferring the work to be done until that needed building block becomes available.

Some implications of this principle are:

a) Because any instance of a building block or the communication network to it may fail, mandatory building blocks cannot be assumed to be available.

b) Also, the building blocks communicate in such a way that effects of a delayed or non-existent response to one request is localized to that request and thus will not impact the processing of other autonomous requests.

c) In case of an interactive building block interface, a mechanism is required to detect the failure of either of the participants and take appropriate action.

d) The inaccessibility of a building block may be the result or symptom of an intentional compromise of the building block or infrastructure. Appropriate security measures must be provided by a building block and associated infrastructure.

2.6.4 Execution in only one recoverable domain

To sustain, preserve and restore the operability of function and data in case of failure of interoperating building blocks, a building block is deployed in one and only one recoverable domain. A building block cannot span recoverable domains.

A recoverable domain is the span of control for a single logical unit of work within a system. In the case where the logical unit of work is a transaction, a recoverable domain is the span of control of a single transaction manager within a system.

The distributed processing environment is the set of systems where each system in the set is reachable by any other system in the set either directly or indirectly via another system and the set of those services and functions for the interconnection of these systems.

Recoverability requirements:

A failure that results in inconsistent data or inaccessible functions or data is a Catastrophic failure and every building block must have a strategy to recover, such that it does not adversely affect the capability of the building block. The recovery must be limited to involved building blocks and should not require recovery of the building blocks that are not involved in the interactions that failed. Rollbacks resulting from transaction aborts or failures are not catastrophic. If a building block is installed more than once at a corporation, each instance must be treated as an independent building block.

2.6.5 Location Independence

A building block is installed in a recoverable domain independent of what other building blocks are installed in that domain. Thus an instance of a building block cannot assume that another building block resides at a specific network location. Thus no instance of any building block may be required to be installed in the same recoverable domain as any other building block or any other instance of itself. Moreover, it cannot be restricted form being installed in the same recoverable domain as another building block or instance of itself. All the addressable and identified units of the example architecture, such as building blocks, human users, contract providers, and units of infrastructure, are identified by logical addresses that are network location independent.

2.6.6 Contracts among Building Blocks.

Definition: A contract is the definition both of the set of functionality and of the interface to that functionality, and a commitment by the building block to offer both set of functionality and that interface to all other building blocks, in a way which adheres to the contract principles.

Contracts are the means of attaining interoperability among building blocks. Therefore,

- building blocks should offer general contracts as against contracts tailored specifically for use by a particular interfacing building block.
- the syntax encoding must be widely used and have general industry acceptance, utilizing appropriate international and national standards where available.
- relevant infrastructure functions must adhere to a set of standards agreed upon among the suppliers and deployers of a building block,
- the agreed upon standards should be targeted towards industry and the emerging national and international standards,
- implementation of a building block must be transparent to the invoker of the building block's contract providers, and
- the definition of contracts must be precise and explicit, and made available to all deployers and implementors of a building blocks.

These criteria are satisfied when the interactions among building blocks adhere to the principles for contracts stated as follows:

A. Use of Standards,

B. Restricted Set of Syntax Encodings,

C. Isolation from Building Block Internals,

D. Release Independence,

E. Equality of Invocation,

F. Well Defined Interfaces,

G. Location Independence,

H. No Contract Accessibility Assumptions,

I. Recognition of Authorized Humans and Building Blocks,

J. Minimum Trust of Invoker,

K. Maintain Identity of Invoking Human and Building Block,

L. Security Audits.

We will not go into a detailed discussion of these principles.

2.6.7 Secure Environment.

A building block must provide a secure environment, provide for the recognition of authorized humans and building blocks, and audit security relevant events according to following guidelines:

- There must be no entry points to building block functions other than those defined in the offered contracts, or in the case of user layer building blocks, those provided for authorized secure human access.
- Sensitive information must be protected appropriately. These protective mechanism should be provided by the infrastructure.
- The identity of the invoking human and building block must be maintained and passed through to any other building block.
- Depending on their security requirements, building blocks may need to re-authenticate the identity of invoking humans and/or building blocks.
- Any access to functions and data within a building block must be limited to authorized invokers. This may be provided by the infrastructure.

- All building blocks must have a capability to provide audit information at a level coinciding with the security requirements.

2.7 Data Layer.

Fundamental to our integration architecture is the separation of data, which is in the data layer, from the operations and functions in the processing layer and the user interaction functions in the user layer. The data layer contains functionality to support the description of a particular domain's information objects.

Each Data Layer Building Block must adhere to all of the general building block principles. The separation of data management functions from specific processing functions provides a framework whereby future technology advances in data base management and heterogeneous distributed data base management systems can be utilized without major changes into building blocks other than the affected data layer building block. The data layer is not merely a data access layer, but it contains functionality to provide invokers with update operations that will preserve the semantic integrity of the totality of the data, including security measures to insure only authorized building blocks on behalf of authorized users to invoke relevant contracts. Thus in summary we can say that the data layer includes as a whole the create, update, delete; ad-hoc and pre-defined retrieval; semantic consistency of data; support of redundancy management.

2.8 User Layer.

This is the layer which provides the means for human users to accomplish the automated portions of their job. The user layer serves as an agent for carrying out the user's tasks by providing functionality and accessing functionality of other building blocks, which may provide data, processing, or other intelligence required to meet the user's business needs. The user layer may access other building blocks through contract invocations with those building blocks. One of the purposes of this is to hide the complexity of computer and communications so that users may focus directly on to their task. The other fundamental purpose of the user layer is to provide means by which humans can make of use of the functionality provided within the user, processing, and data layers to meet their business needs.

The user layer building block is a deployable unit that contains functionality that supports direct interaction with a human. Any function that is related to the concerned domain and interacts with a human belongs and resides only in a user layer building block. The complexity of these functions range from displaying data to interpreting a voice activated signal. The key idea is direct interaction with a human.

2.9 Processing Layer.

The processing layer provides a functional abstraction that does not presume interaction with a human. Functionality in the processing layer is the functionality that can be shared among various work tasks, or operate in the background or batch, or is long running. Processing layer building blocks among other functions crunch numbers, construct report contents (but not the format of the report) and complex retrievals, control process flows, coordinate multiple tasks and manage relationships and behavior between individual user tasks encapsulated in different user layer building blocks, and provide coordination of human interactions, activity control, and access to shared objects encapsulated among different user layer building blocks.

Processing layer building blocks do not own data, but they use and produce data. They can obtain data from and send data to the data layer. To do this they issue a request to a appropriate data layer building block. That DLBB is now responsible for accessing the data and returning the desired values.

Based on the discussion of the last few section of this chapter, we will now discuss the example prototype that was developed to conform to the concepts of the architecture described above. First we will go into the details of the requirements of the unautomated system. Then we will focus on the previous attempt to develop the prototype Computer Science Department Information System based on the channel based building block architecture. These are the main points of discussion of Chapter-3.

The chapters following the third one go into the enhancement/improvements and detailed design issues of this Enhanced Information System for CS Department.

CHAPTER 3

FUNCTIONAL OVERVIEW OF THE CS DEPARTMENT INFORMATION SYSTEM

3.1 Operations and Requirements of CS Information System

First of all we look into the requirements of the Computer Science Department and its related operations, based on which we have developed a distributed information system conforming to the principles of systems integration. As with any department in a University, the CS department is organized into various entities, such as Faculty/Staff members, students studying in that department, courses being taught/studied by faculty/students respectively. And there are various activities associated with these entities, such as, student registration which requires information on which faculty is teaching which subject, when, where and whether any prerequisite courses are needed for that particular course or not. Thus it involves other departments such as registrar's department, admissions office, finance department, international student's office and Graduate/Undergraduate studies office; which are interested in the departments information relating to the faculty members and the students. These departments may be located in different buildings. If all of these operations were not automated it would be very tedious job to keep the records of students and faculty members up to date across all of the above mentioned departments, due to variety of reasons, including time delays, human inefficiency, inaccuracy etc. The major operations relating to the CS department are the student admission, registration, and certain other basic data management functions for each of the entities and some queries and specialized reports for the use by students and/or administrative officials and faculty members.

Thus the system will have basic add, delete, list all, view a specific information, certain specialized query options such as, Courses taught by a particular faculty, courses registered by a student and number of students in a particular course etc. The system is very simple in its range of operations, but it serves the underlying design issues.

Thus, we strive to have a application for CS department that can be accessed from various locations, from various computer systems, with minimum of information redundancy, no duplication of code and high accuracy and efficiency of the desired tasks. A prototype DIS application has been developed previously that conforms to the Generic Integration Framework principles briefly explained in the first chapter. One of the goals of this thesis is to improve this DIS system in its design so that it conforms more closely to the framework and the underlying integration architecture. The other fundamental objectives of this work are to compare the traditional system development life cycle with one in the integrated system development efforts. The improved version of the DIS system has been developed separately using two different communication tools, with the purpose of comparing their infrastructural facilities as against the needs of the systems integration development. Since the basic nature of the prototype is distributed the next section tries to focus on the differences between centralized systems and the distributed integrated systems.

3.2 Distribution Transparency in an Integrated System

The data (information) and the software that constitute an application system reside on a single hardware machine, with associated secondary storage devices such as disks for on-line data storage and tapes for backup. Such a system is called Centralized Application System, since all system components reside at a single computer or site.

In recent years, there has been a rapid trend towards the distribution of computer systems over multiple sites and the interoperability among various systems, which are spread over a heterogeneous computing environment(s). In a good distributed system the distribution of functionality and information is hidden from the user, as if the user were operating on a centralized system. Distribution transparency is thus a combination of localization and access transparency. As described in earlier chapters this is achieved by having a meta-level controller framework and its underlying architecture, which influence the design as well as the development of the system with the help of certain infrastructural communication tools. The generic functionality provided by this tools acts as a software bus/channel.

This concept of sharing of functionality of different independently running systems in order to achieve interoperability, reusability and consistency and integrity of data, addresses the need to study the security, consistency and validity issues in an Integrated System environment. Those are precisely the issues that have been looked into while developing the new prototype. But before we go into the details of the newer implementation of the prototype let us first understand the implementation and the problems related to the DIS system. Then we will look into some desired improvements and how much has been achieved in the improved version, the EIS.

3.3 Overview of the DIS system Implementation

The DIS (7) was built based on the concepts of the integration architecture described in chapter 2. It was the first attempt in NJIT to develop a distributed system based on systems integration principles, and hence provides a basis to the objective of this thesis. Thus it is very important to understand it and put some light on the concepts of fully transparent communication between the building blocks of a system through a software bus mechanism. We also discuss the possible improvements that can be made to the DIS, so as to fine tune the rough edges of it as well as adding new properties and characteristics to it. The ultimate goal is to have a system that conforms more and more closely to the systems integration framework and also addresses information consistency and security issues. Another p[oint is to evaluate a more sophisticated approach to the software bus concept.

Let us begin by looking into what kind of functionality is provided by DIS (7). The user of the DIS can typically be divided into following categories, a student, a member of faculty, CS department administrative staff, staff of registrar's office, and the Chairperson of the department, faculty advisors, and staff of the other offices related to student and faculty information. Users from each of these categories are interested in different functionality of the system and hence can have different views of the system. The DIS has the detailed information on each of the students registered under CS department and also the information on the faculty members of the department, and the courses offered by the department. It provides functionality to Add, Delete, Update, View, and List these information so as to assist the user in completing his/her tasks at hand easily and efficiently. It also provides certain specialized queries for the purpose of report generation and statistical information retrieval. The user is asked to input certain information, such as Student Identification Number, or Faculty Social security Number, etc. in order to process user's request. The system has functionality, that interacts with the human user directly, and some relate only to the raw data in the data repositories, while some depend on this data to produce some other form of data, or change the existing data after doing some processing and/or computations, formatting, reorganizing, etc (7).

If we compare this to our example architecture, this application can be easily divided into three layers of functionality based on the separation of concerns principle, viz. User Layer, Data Layer and Processing layer. The tool used by the DIS system developer is the Sun's Remote Procedure Calls to achieve distributed nature of the application.

The systems has four building blocks (executables) communicating with it each other depending on user request. These are frdbs (Faculty Building Block), rrdbs (Register BB), srdbs (Student BB), crdbs (Course BB); located on two different machines. Any of these four building blocks can be put on any of the machines. The user layer building block is in this case a client program through which the user has access to the above mentioned building blocks. This will result in multiple copies of the user layer building block on various machines from which the user wants to access it. Thus it involves duplication of code, and certain incompatibility issues as regards to the compilers and display devices used by those machines.

There are certain other drawbacks into the DIS system, such as improper implementation of data and the processing layer building blocks. It does not look into the issues of building block failure and security of data repositories. Thus the following section lists some improvements that can be made to DIS system.

3.4 List of Possible Improvements

1. Having a server building block for User-Interface.

The DIS system has a drawback, among others, of duplicating the user interface programs on each of the machines from which the building blocks of the application are to be accessed. Thus this approach duplicates the functionality related to the Menu-Handling and Data Entry routines, and the other routines that require direct involvement of a human being. I disagree with this approach as it is not efficient and is a waste of efforts and storage space. It also puts certain limitations on the portability / remote accessibility issues of the system due to incompatibility of the utility softwares / tools on various machines. Therefore the better alternative is to have one server machine which has the user-interface building block(s), which provide functionality that require direct involvement of a human being. Now the application which wants to access these functionality can build contracts with the necessary and sufficient building blocks.

This will enable the programmers to provide view for the different categories of the users of a particular system. This approach will also eliminate the hassles of installing the user interface programs on each of the required machines/systems from which the application is supposed to be accessed.

2. Better Building Block Implementation

From my observations while using the DIS system, and studying its design, I have found that it does not conform to our example integration architecture. In my view it was designed with a different kind of interpretation of the principle of building blocks and hence the building blocks have misgrouped functionality and also the application as a whole has not been divided into three logical layers based on the type of functionality. This thus defeats the separation of concerns principle which we are looking for. Though the DIS system claims to conform to the architecture and its building block principles, it contradicts itself in the explanation of the design.

In my view a building block from a programmers point of view is a runnable entity, an executable, or a software system that conforms to the contract rules of the architecture and provides clear and detailed specification for accessing its functionality. The functionality it provides should conform to the principle of three kinds of functionality layers and should provide functionality relating only to one of the three layers.

3. Consistency and Security issues of building blocks

The DIS system does not look into the consistency and security issues of the information maintained and managed by the application. Thus as one of the improvements to it the enhanced version of the application will also focus on the failure of building blocks and its consequences, a primitive commit protocol implementation.

This can be best explained when we discuss the implementation of the enhanced version (EIS, Enhanced Information System) in detail.

4. Improvement to the Trader

As it stands now the DIS system, since it has been developed using Sun RPC tool, has some restrictions in message passing and also requires prior knowledge of the target building blocks. Thus we need to look into these two issues relating to the trader, which serves as a software bus for all the message passing mechanisms among the building blocks.

5. Physically distributed Data

We have looked into the aspect of physically distributed data repositories, but realized soon enough that by distributing data on different machines we would first of all complicate the data management algorithms and also give rise to a few more consistency and security issues. And on top of everything the idea of distributing a data repository of a particular entity does not conform to our architectural requirements. This is because of the data layer principles discussed in chapter 2.

6. Evaluation of Features of another Tool

The last, but very important change from the DIS system is use of another communication tool called ANSAWARE, that is based on ANSA architecture's principles. Here we will keep the design same as in the RPC-version of the enhanced system, and look into the issues discussed above and then compare and evaluate it usage with the corresponding features in RPC implementation.

CHAPTER 4

THE ROLE OF RPC IN THE IMPLEMENTATION

Remote procedure call (RPC)s are a method of interprocess communication over a network. The benefits of RPCs is that they allow a programmer to program at a higher level of abstraction than most network interprocess communication mechanisms. This abstraction means that the programmer need not learn a new syntax to create a distributed application or network service. RPC uses XDR routines to marshall data on a network. The process of making a remote procedure call is very much like that of making any procedure call. The reason we even call this a "procedure call" is because the intent is to make it appear to the programmer that a normal procedure call is taking place. We call the caller, a client, and the callee the server. We use the term request to refer to the client calling the remote procedure, and the term response to describe the remote procedure returning its results to the client.

4.1 How Does RPC Work?

Figure 4.1 shows the steps that normally take place in a remote procedure call. The numbered steps in Figure 4.1 are executed in order.

Step 1.

The client calls a local procedure, called the client stub. It appears to the client that the client stub is the actual server procedure that it wants to call. The purpose of the stub is to package up the arguments to the remote procedure, possibly put them into some standard format and then build one or more network messages. The packaging of the client's arguments into a network message is termed marshalling.

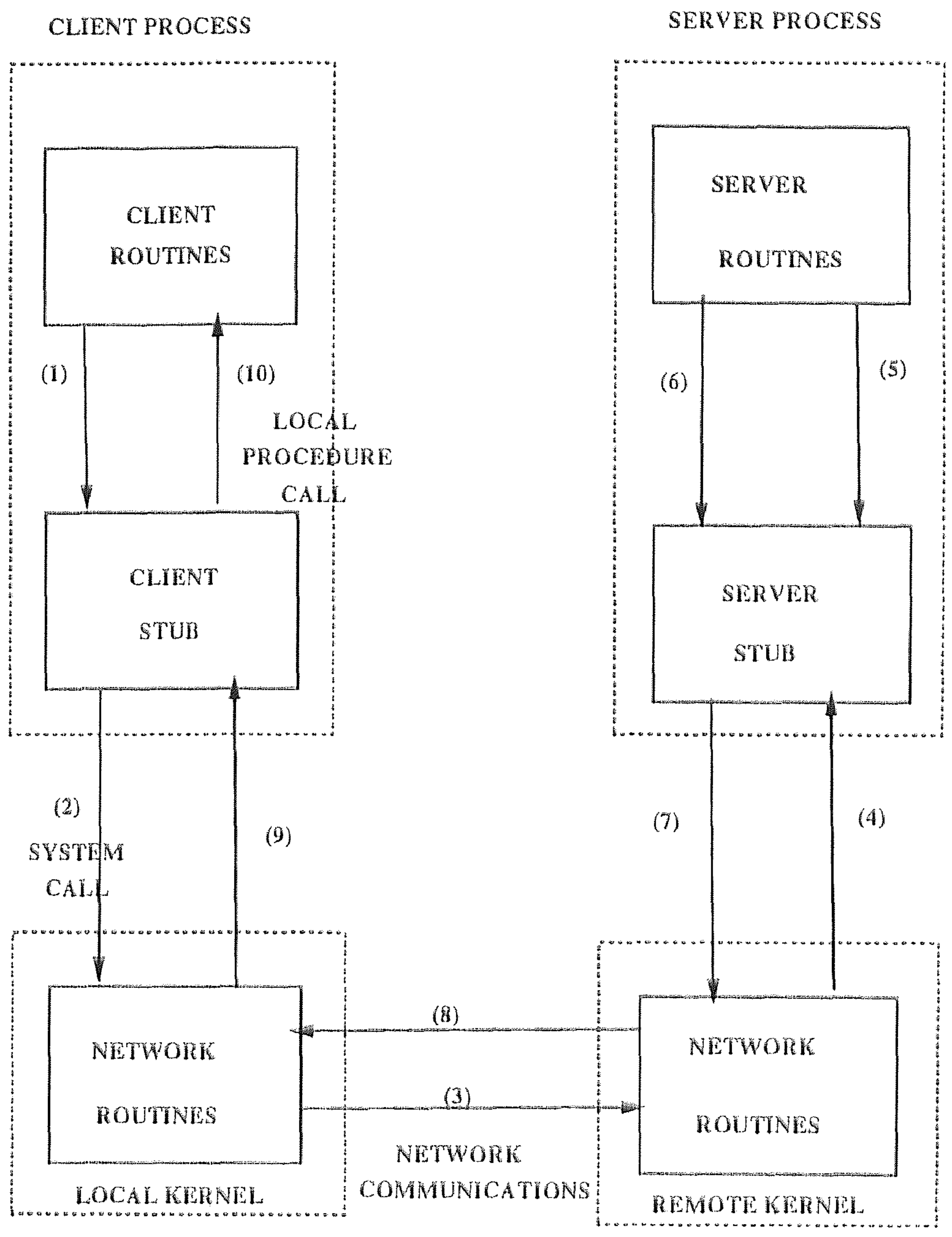

Figure 4.1 Remote Procedure Call (RPC) Model

Step 2.

These network messages are sent to the remote system by the client stub, by a system call into the local kernel.

Step 3.

The network messages are transferred to the remote system. Either a connection-oriented or connectionless protocol is used.

Step 4.

A server stub procedure is waiting on the remote system for the client's request. It unmarshals the arguments from the network messages and possibly converts them.

Step 5.

The server stub executes a local procedure call to invoke the actual server function, passing it the arguments that it received in the network messages from the client stub.

Step 6.

When the server procedure is finished, it returns to the server stub, returning whatever its return values are.

Step 7.

The server stub converts the return values, if necessary, and marshals them into one or more network messages to send back to the client stub.

Step 8.

The messages get transferred back across the network to the client stub.

Step 9.

The client stub reads the messages from the local kernel.

Step 10.

After possibly converting the return values, the client stub finally returns to the client function. This appears to be a normal procedure return to the client.

What the concept of RPC provides is the hiding of all the network programming into the stub routines. This prevents the application programs from having to worry about details such as sockets, network byte order, and the like. There are many different implementations of RPC, such as, Sun Microsystems' **Open Network Computing(ONC)/RPC, Xerox Courier/RPC, Apollo's Network Computing Architecture (NCA)/RPC**. We have used the ONC/RPC implementation, so we will go into its detail where we explain its usage with reference to building block definition and inter building block communications.

4.2 How a Building Block is Defined

Since we have used Sun RPC implementation, let us go into some detail about its parts:

- rpcgen, a compiler that takes the definition of a remote procedure interface, and generate the client stubs and the server stubs.

- XDR (eXternal Data Representation), a standard way of encoding data in a portable fashion between different systems.

- A run-time library to handle all the details.

Please refer to Figure 4.2 to understand the files that are involved in generating a Sun RPC program. The Figure is self-explanatory. In order to look into the process of defining a building block let us consider a simple example of a remote date and time service. Let's look at the specification file that is the input for the rpcgen compiler.

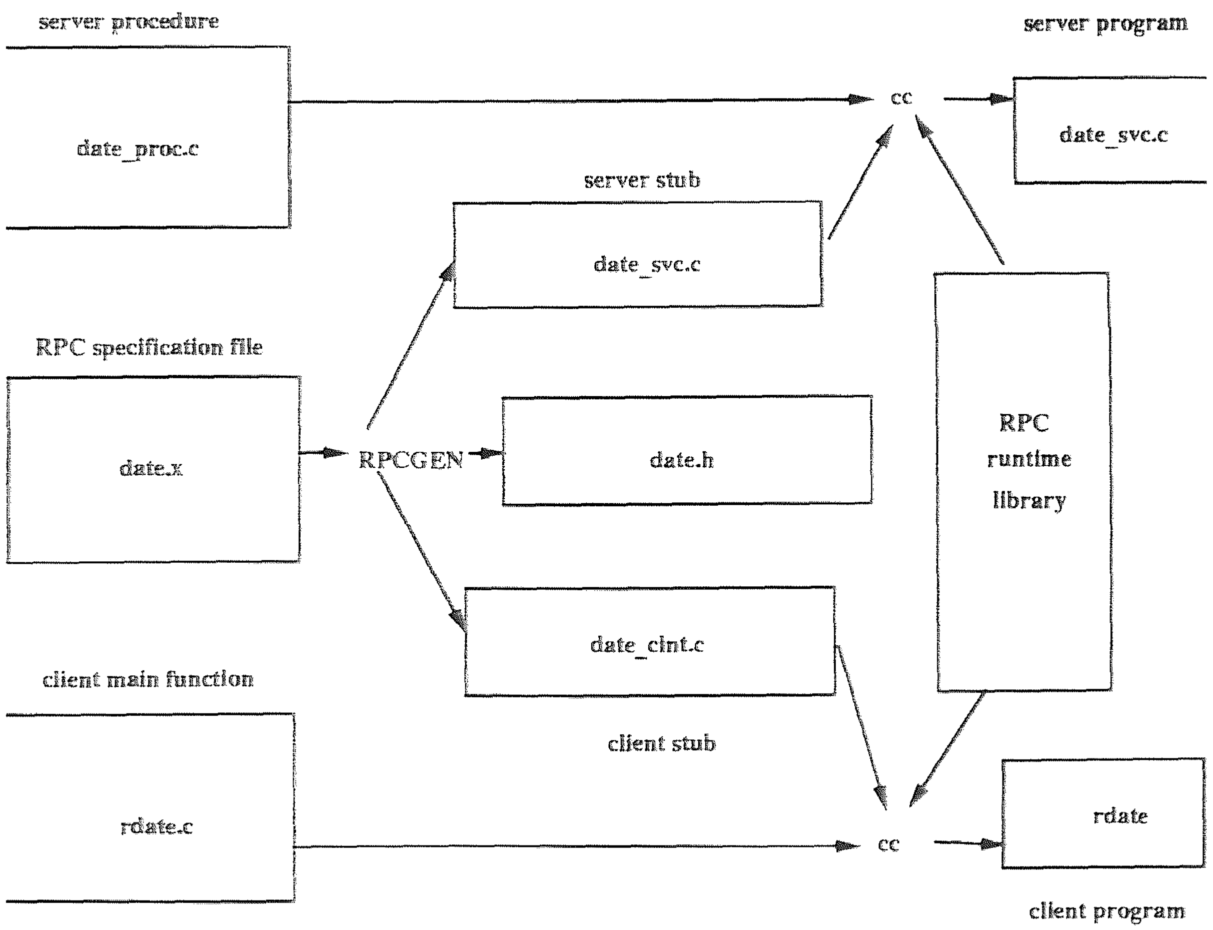

Figure 4.2 Files Involved in Generating Sun RPC Program

We declare both the procedures and specify the argument and return value for each. We also assign a procedure number to each function (1 and 2), along with program number (0x31234567) and a version number (1).

```
/*      date.x - Specification of Remote Date and Time service.
        Operations: bin_date_1() :binary date/time
        str_date_1() takes a binary time and returns a string.
program DATE_PROG {             /* Program Name */
        version DATE_VERS {
                long   BIN_DATE(void)   = 1; /*procedure # = 1 */
                string  STR_DATE(long)  = 2; /*procedure # = 2 */
        } = 1;                          /*version # = 1 */
} = 0x31234567;                         /*program # = 0x31234567
```

This specification file (date.x), from our building block point of view is a specification of name of building block, its version number, definition of its functionality (input arguments and output results). The key point to note here is that the client building block that wants to request this date/time remote functionality has to use the client stub and the XDR routines generated by the rpcgen program with itself, while linking. Similarly the actual building block program that provides this functionality has to use the server stub and the XDR routines with itself.

A specification file in Sun RPC, which is a protocol definition file, defines a building block for us. Thus, it is very important to decide which functions to put in a protocol definition file. We still have to use our concepts about the integration architecture while designing/deciding what functions can be grouped into one specification file.

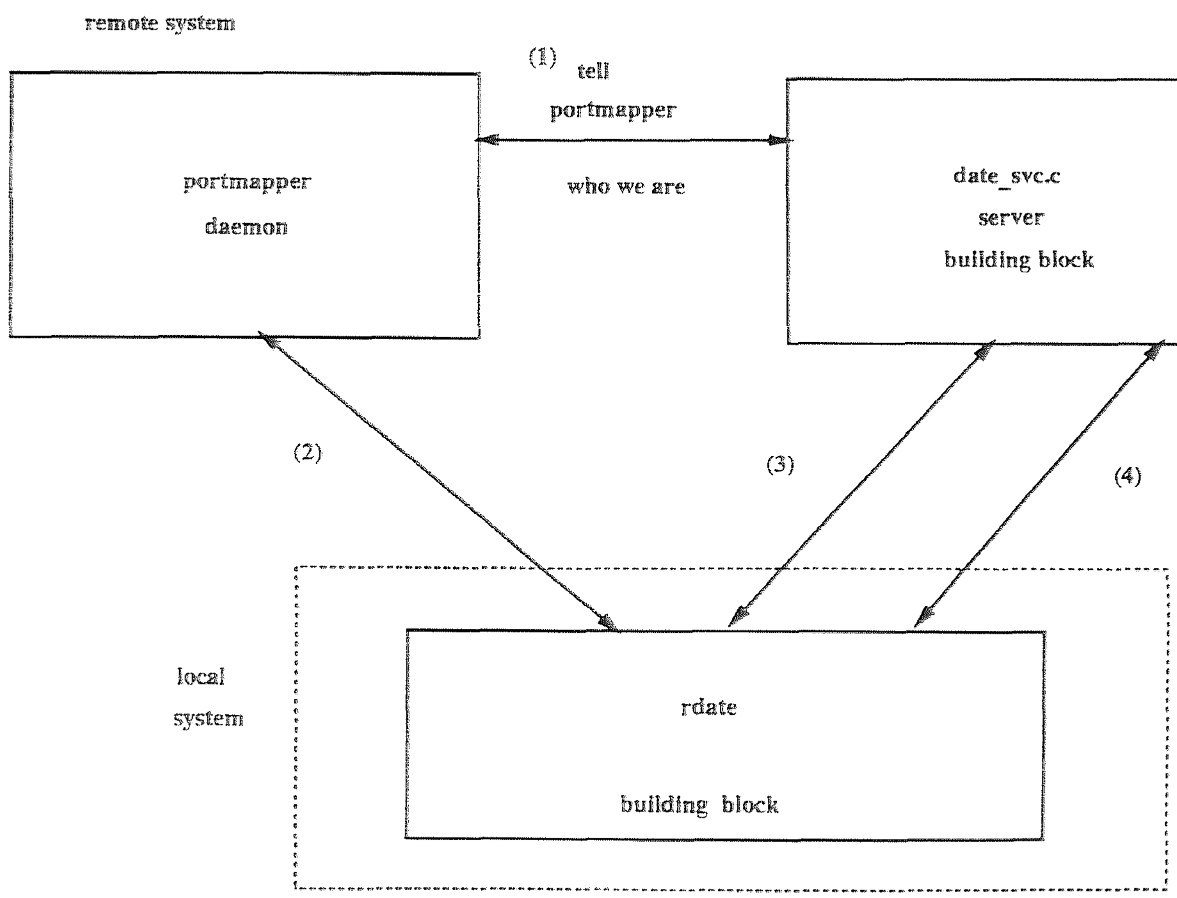

Figure 4.3 Inter Building Block communication

Thus from our integration architectural point of view and from the programmer's point of view this protocol definition/specification file is a very important, since it can serve as a building block specification document.

Now let us look at how these building blocks can communicate with each other.

4.3 Inter Building Block Communication

Pictorially we have the processes shown in Figure 4.3. The steps shown in Figure 4.3 are executed in the following order:

Step 1.

When we start the server program on the remote system it creates a UDP socket and binds any local port to the socket. It then calls a function in RPC library, *svc_register*, to register its program number and version. This function contacts the port mapper process to register itself. The port mapper keeps track of the program number, version number, and the port number.

Our building block then waits for a request from any other building block or a client application program. All these actions are taken by the server stub generated by the *rpcgen* compiler.

Step 2.

We start our application program (or through it another building block) that then calls the *clnt_create* function. This call specifies the name of the remote system, the program number, version number, and the protocol. This function contacts the portmapper on the remote system to find out the UDP port of the server.

Step 3.

The client building block calls one of the operations from the date-building block. This operation is defined in the client stub. This function in the client stub sends a datagram to the server, using the UDP port number from the previous step. It then waits for a response, retransmitting the request a fixed number of times if a response isn't received. The datagram is received on the remote system by the server stub associated with our building block program. This stub determines and calls the required procedure. When that particular function of the building block returns to the server stub, the stub takes the return value, converts it into the XDR standard format, and packages it into a datagram for transmission back to the requester. When the response is received, the client stub takes the value from the datagram, converts it as required, and returns it to our client building block program.

4.4 RPC - Pros and Cons and Trade-Offs.

As is evident, taking something as simple as a procedure call and transforming it into system calls, data conversions, and network communications, leads to a greater chance of something going wrong. The goal is to make the use of RPC transparent to the application, compared to calling a local procedure, but issues of parameter/message passing, binding. transport protocol, exception handling, call semantics, are important ot be considered.

Parameter Passing

A single argument and a single result are allowed. Multiple arguments or multiple results must be packaged into a structure. In other words it has to send/receive in one-message format. This is a kind of restriction that complicates things and takes away some flexibility of programming. This requires packaging and unpacking of information, which will not be necessary if had it allowed multiple arguments/results to be passed.

Binding

The port mapper daemon on the desired remote system is contacted to locate a specific program and version. The requester building block has to explicitly specify the name of the remote system and the transport protocol. Thus the program has to know the location of the target building block, which is not a perfect way of inter building block communication. One of our goals is to hide the knowledge of building block location from the programmer. Thus this requirement of explicit knowledge of the location of a building block does not conform to full distribution transparency property as we desire.

Transport Protocol

Sun RPC currently supports either UDP or TCP. There are some differences in the call semantics for each one that we discuss below, which are of special interest to us.

When a stream-oriented protocol such as TCP is used, there has to be some way to delimit the records in the byte stream. Sun RPC uses a 32-bit integer at the beginning of every record to specify the number of bytes in the record.

When using UDP, the total size of the arguments must not generate a UDP packet that exceeds 8192 bytes in length. Similarly the total size of return values must also be less than 8192 bytes. There is no such limit when using TCP. This is one of the reasons why EIS uses the TCP protocol.

Exception Handling

When UDP protocol is used, it automatically times out and retransmits a request for a functionality from a remote building block, if necessary. It terminates and returns an error to the caller after a fixed number of unsuccessful tries. If TCP is used, an error is also returned to the caller if the connection is terminated by the server building block. There is no way for a building block to send an interrupt to any other building block.

Call Semantics

Sun RPC protocol provides for every requester building block a unique transaction ID, 32-bit integer termed *xid*. This ID is initialized to some random number when a client (requester) handle is created. This is then changed every time a new RPC request is made. Before returning the results this ID is checked internally for a match. This is to ensure that the response is from the request that the requester (client) made. This is one reason why EIS fails to provide a primitive commit/rollback protocol.

If this is the case with the transaction IDs then how is it possible to for UDP protocol to retransmit the request? The reason is that the UDP server functions have an option that causes the UDP server to remember the requests that it receives. This is useful only in returning the saved previous response. This approach, however, assumes that the previous response must have been lost or damaged. Thus it is not useful in implementing a primitive commit/rollback protocol.

We want to provide this commit/rollback feature for inter building block transaction with the aim of providing security and consistency to the data/information managed by the application, by the collection and communication of various building blocks.

CHAPTER 5
THE ROLE OF ANSAWARE IN THE IMPLEMENTATION

ANSA is an architecture for Open Distributed Processing. ANSAWARE is an example implementation of that architecture. ANSA supports the design and construction of flexible distributed applications. It is not constrained by network structure and size , or mixes of differing hardware and operating systems - and goes beyond distributed operating systems, databases, networking.

5.1 ANSAware.

ANSAware is a suite of software for building Open Distributed Processing systems, providing a basic platform and software development support in the form of program generators and system management applications. ANSAware provides a uniform view of a multi-vendor world, allowing system builders to link together distributed components into network wide applications.

5.1.1 Objects and Interfaces

ANSAware allows applications to be written in an object-based style, using the client/server model of interaction. An ANSAware *object* is an encapsulation of an application and its data. An object provides services via *Interfaces*. Several named operations may be provided in each interface which may be used either locally or remotely by client objects. The object based approach allows the physical separation of distributed program components to be managed effectively, and allows the containment of system failure. To reduce complexity, transparency mechanisms are available to hide the mechanics of distributing objects. ANSAware cuts out the networking overheads.

5.1.2 System Management

A suite of system management application extends the functions of the basic platform. These management applications are themselves built using ANSAware.

- **Trading**

  Traders give access to information about available servers. Trading matches offers and requests for particular services, using service names, interface types and service properties in combination as selection criteria. In this way, the separate parts of a distributed application can find each other on demand.

Also there are **factories** and **node management** facilities available.

5.1.3 TOOLS

Interface Definition Language (IDL) - specifies the operations available in an interface. All interactions between ANSAware objects are via interface specifications written in IDL, preventing errors and misuse.

STUBC - is the utility which complies an interface specification written in IDL into stub routines and header files in C for inclusion in programs which will provide and use that interface.

PREPC - is the preprocessor which extracts control commands from C programs and translates them into calls to the stub procedures prepared by STUBC. Control commands exist for declaring interfaces, performing trading functions and calling operations in local or remote interfaces.

5.2 How Building Blocks are defined

Building blocks are defined as interfaces, using IDL specifications. We will consider the same example as in the RPC explanation.

```
/*
**      INTERFACE Definitions for Remote Date and Time Service
**      Define Operation bin_date that requires no arguments
**      and returns an INTEGER as a result
**
**      Define another operation str_date, that requires an
**      INTEGER
**      argument and returns a STRING as a result.
*/
/*      INTERFACE NAME */
date_time_service : INTERFACE =
BEGIN
bin_date : OPERATION [] RETURNS [result : INTEGER ];
str_date : OPERATION [var : INTEGER ] RETURNS [result2  STRING ];
END.
```

This way we can define our building blocks using IDL. Please refer to APPENDIX B for definitions of data layer building blocks of our EIS application. EIS, ANSAware version, is a system similar to that developed using RPC. This specification when compiled by stubc and when used with a service function, will act as a service interface providing those functions to the other interfaces/clients in the distributed environment.

5.3 Inter Building Block Communications

Step - 1.

When we start up the server program (i.e. the building block), it registers itself with the trading function of ASNAware. Hence it is available to any of the clients through the trading function. The trader keeps track of various interface types (i.e. the various building blocks) that have registered. Any client program or a building block that wishes to access a building block has to import the services of that building block before actually using its operations. This is done through the trader. The servicing building block has to export its services to the world and should also specify the maximum number of requests that it can handle simultaneously.

Step - 2

We start our application program (or through it another building block) that then imports the service of another building block by specifying its name. It can then access any operation provided by the building block by specifying its name and the operation to execute, along with required arguments. Unlike RPC we can send multiple arguments and also receive multiple results from the serving building block.

Step - 3

Since ANSAware also uses concepts of stubs, its mechanism of client request and execution of operation and hence the response is pretty much the same as RPC. But ANSAware being a more sophisticated tool, has better features for distributed computing and hence is a value-added package as compared to pure RPC: In case of delayed response or no response from the server, the client building block will either time-out or will try to look for relocation of the server building block onto some other location on the network; if not, then the request fails within a predefined time limit. Also there is a facility for providing programmer's own exception handling functions, instead of timing out the request.

5.4 Transparency Issues while working with ANSAware

Parameter Passing

Multiple arguments and multiple results are allowed. This is taken care of very neatly in ANSAware. This adds to the flexibility of programming and also reduces lot of effort which would otherwise be required to marshall and unmarshall the data.

Binding

First of all a building block service is asked from a local trader, upon failure of which a master trader is consulted that finds out the actual location of the service and hence will build a contract (IMPORT of building block services) with the required block. Here we do not know the location of the target building block at any point of time. As a matter of fact ANSAware will also look for a possible relocation of a building block in the case of failed/delayed response from the server.

Exception Handling

The PREPC precompiler used by ANSAware provides some exception handling. While coding a operation invocation statement in the client program, if we use the "exception" clause, then we can code an exception handler routine by ourselves using the status codes returned by an operation. Regardless of whether an operation succeeds or fails, or times-out, it will return some status-value, and depending on which of the given status-lists the actual status-value appears in , one of these actions will be taken: Continue, Abort, Signal. In the third case the programmer has to provide the exception handling routine. The return value of this exception-handler determines the final action taken by the program.

Call Semantics

Since ANSAware uses the concept of Interface Refrences/Instances, and each of these instances have a ID associated with it, every time a new instance is created, its ID is changed from the previous one. Thus each of the requests to a same building block service will also have multiple instances having different IDs. In order to realize proper operation of an ANSAware application the programmer has to discard the instances or interface references once he/she is done with it. Thus it is not easy to keep the history of transactions or operations, so that when the need arises we can rollback them back or commit them together as a atomic unit. However, we have not used ANSAware to its fullest capability - it provides transaction mechanisms (THREADS) and certain other system management tools as well as other utilities that can help realize a transaction mechanism in future versions of EIS application.

Now we move on to the description of functionality and its implementation using two different tools. Depending on our discussion here and in the following chapter, we will be presenting conclusions in the Seventh Chapter, which will focus further on the comparison of the two tools used.

CHAPTER 6

REALIZATION AND/OR EVALUATION OF SUGGESTED IMPROVEMENTS

6.1 Detailed Design of EIS.

EIS, Enhanced Information System, is a improved enhanced version of DIS. The source code is completely new, but much of the functionality of DIS has been regrouped and some of it has not been retained , since it did not give us any useful, new results to help us on our goal of studying the systems integration framework.

6.1.1 Entity Relationship Diagram

Figure 6.1, shows the various entities involved in the application and also their relationships with each other. The figure is quite self-explanatory and hence is not explained further.

6.1.2 Data Flow Diagram

Figures 6.2 - 6.6, shows the data flow diagrams at different levels and hence shows in more detail the flow from one building block to the other building block of the application.

6.1.3 Data Dictionary

Here, the abbreviations used in the ER Diagram and the Data flow diagrams have been expanded to make their meanings clear to the reader.

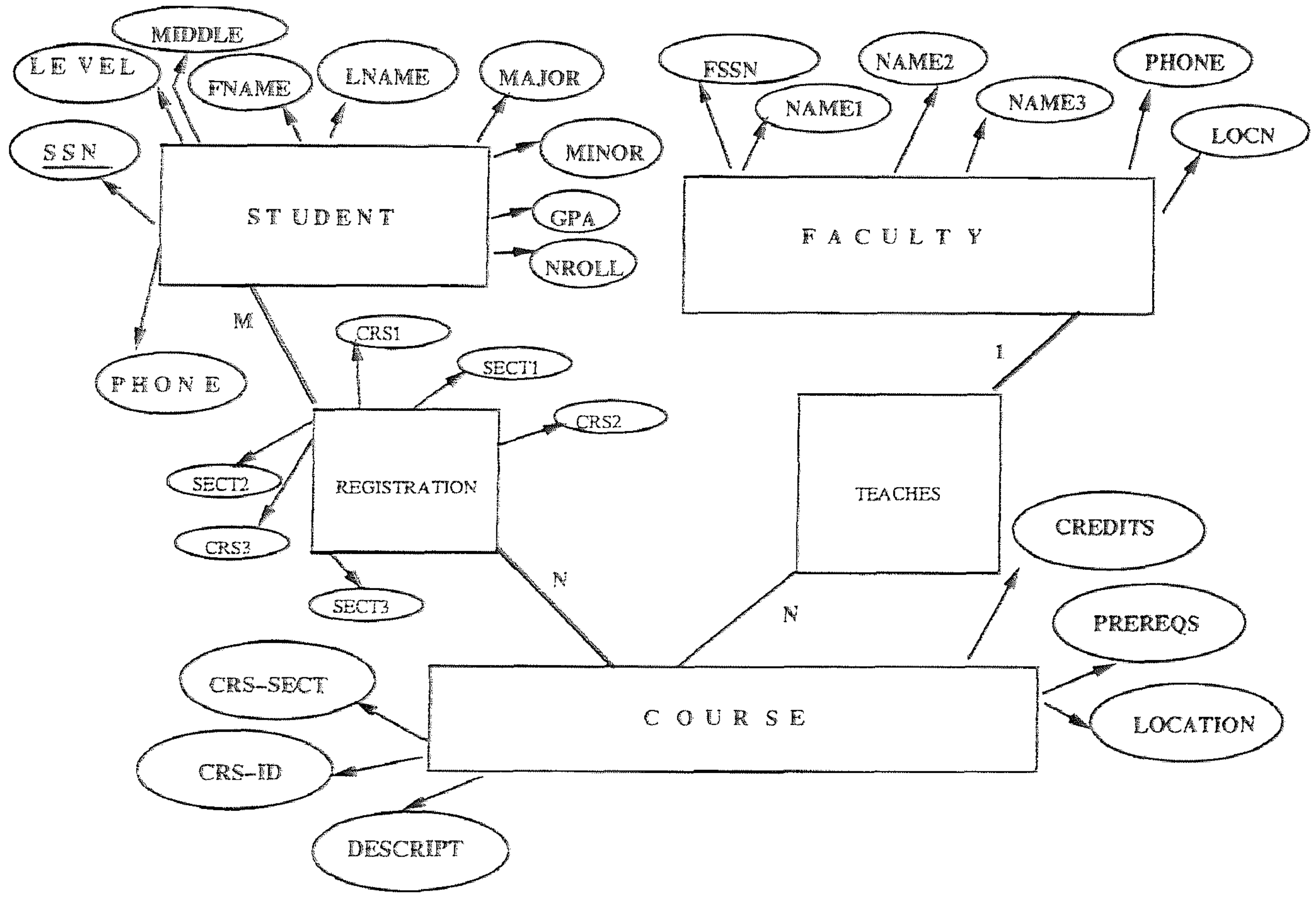

Figure 6.1 Entity Relation Diagram for EIS

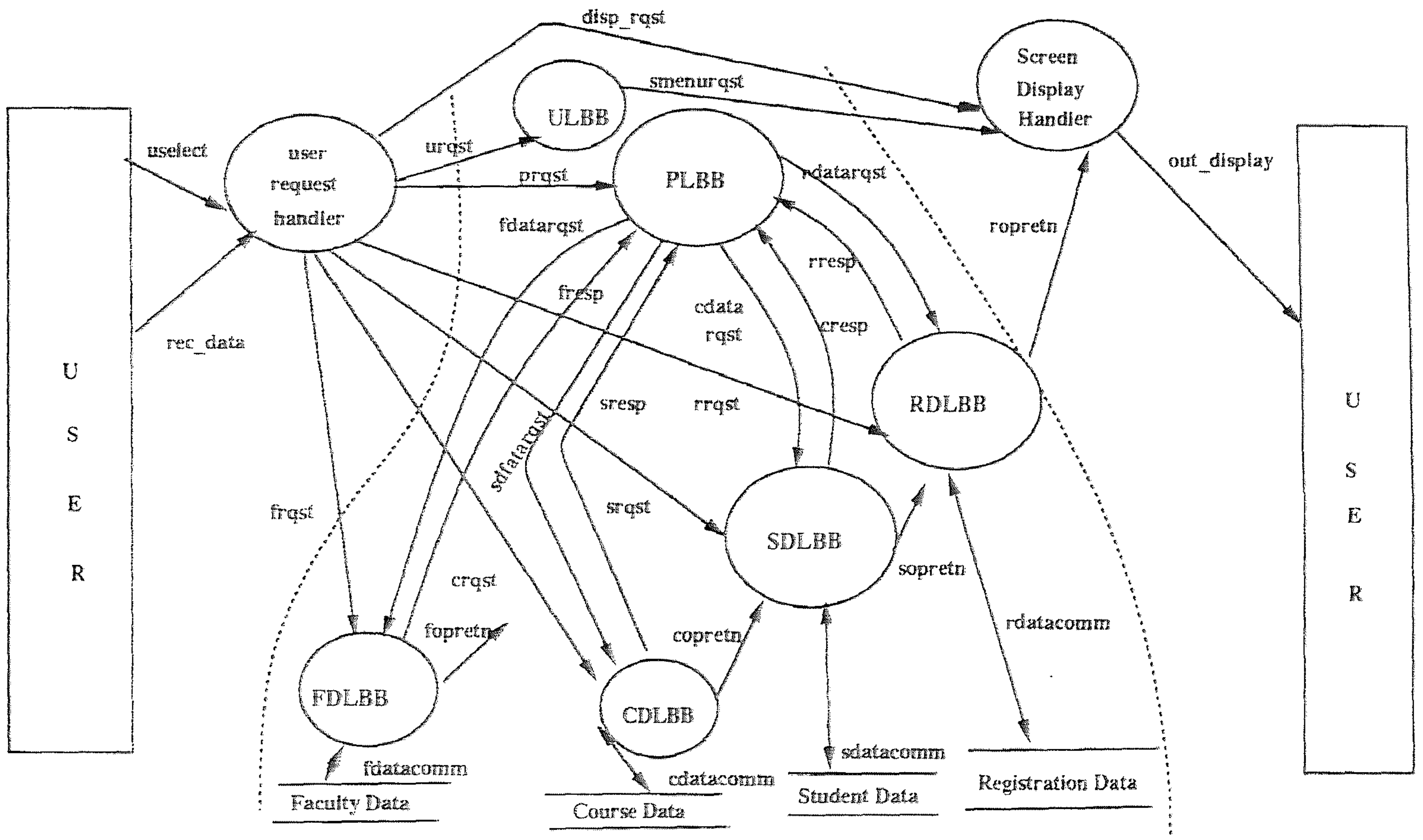

Figure 6.2 Level 1: Data Flow Diagram

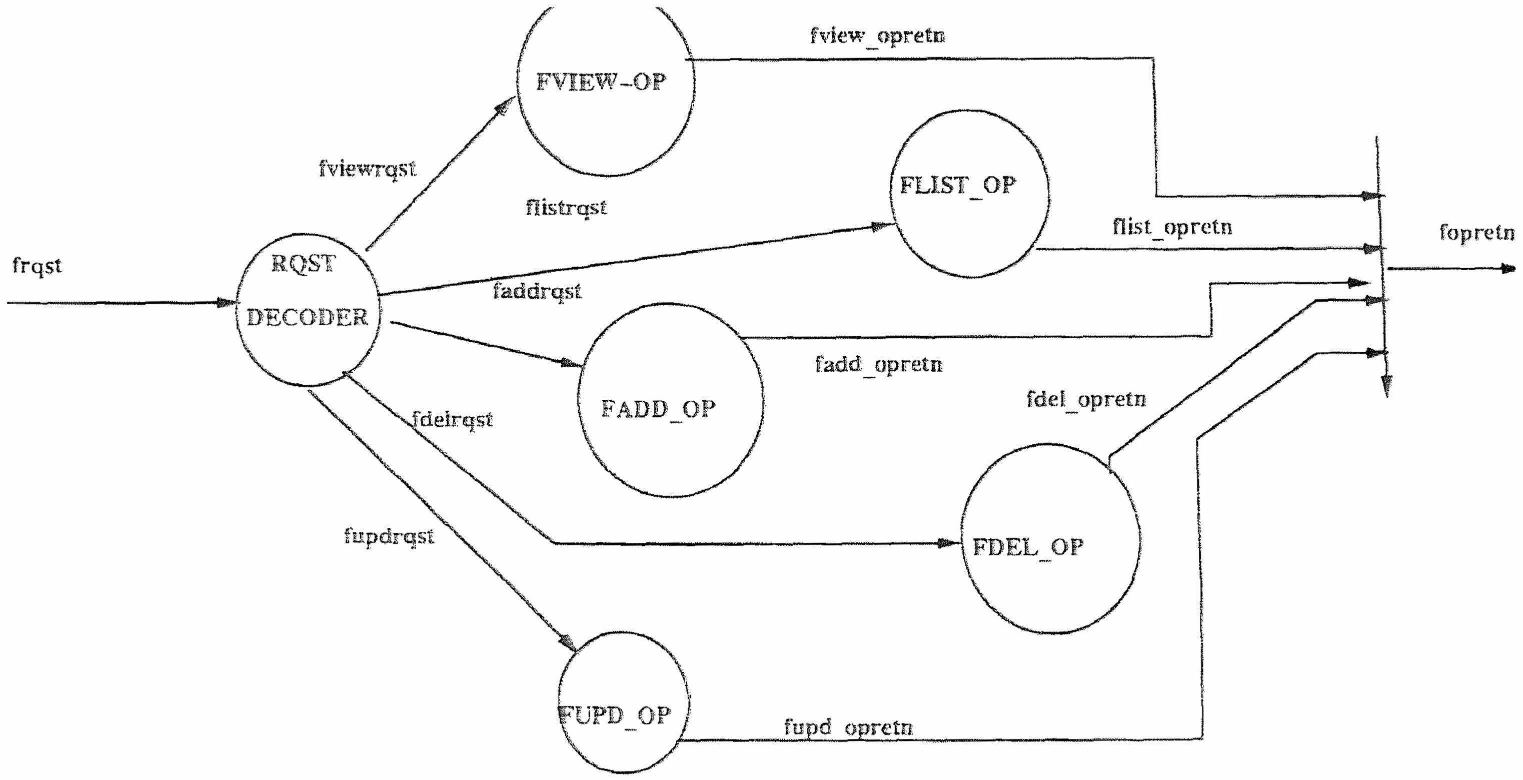

Figure 6.3 Level 2: Data Flow Diagram Explosion of FDLBB

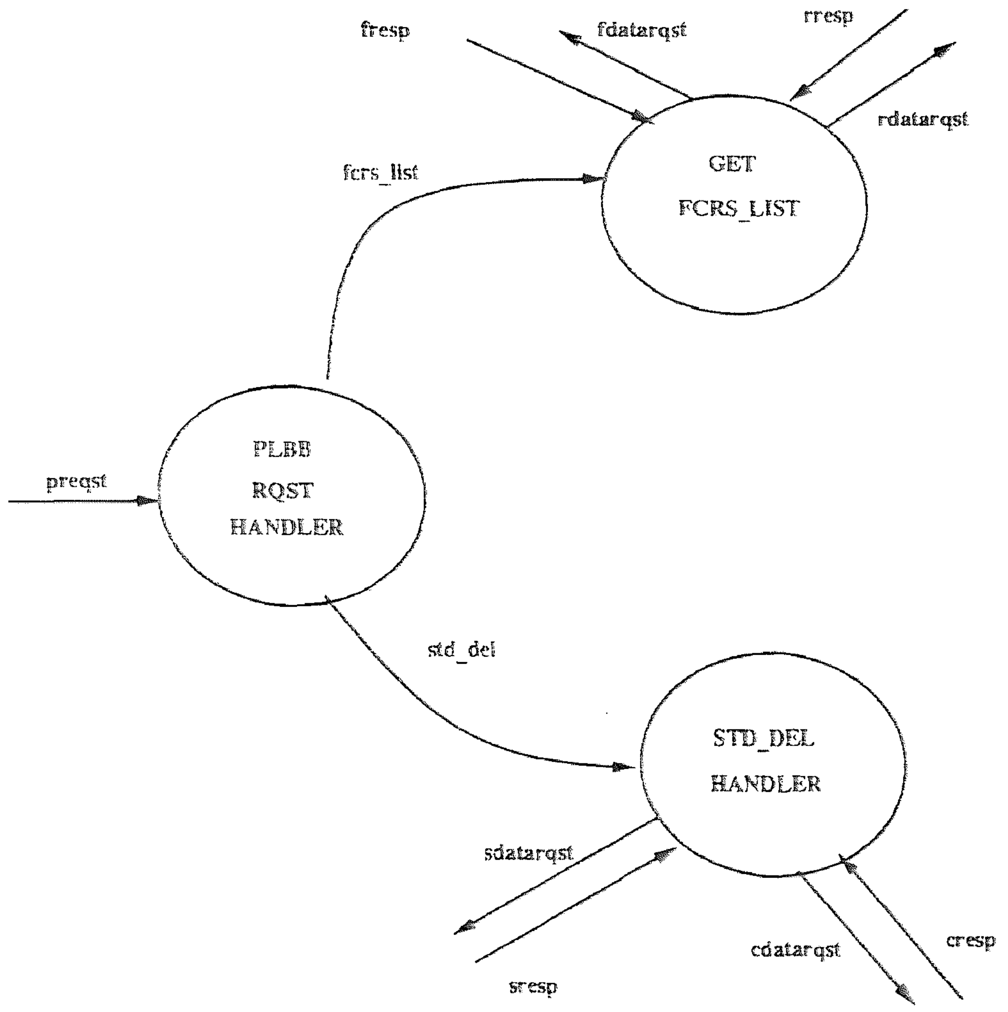

Figure 6.4 Level 2: Data Flow Diagram Explosion of PLBB

DATA DICTIONARY

1) SSN	- Social Security Number of a Student
2) FNAME	- First Name of Student
3) MIDDLE	- Middle name or Initial of a Student.
4) LNAME	- Last Name of a Student
5) GPA	- Grade Point Average
6) NROLL	- Indicating if Enrolled or not
7) FSSN	- Faculty Social Security number
8) NAME1	- First Name of Faculty
9) NAME2	- Middle Name of a Faculty
10) NAME3	- Last Name of Faculty
11) LOCN	- Location of the Office of the Faculty
12) CRS[#]	- Indiactes a Course, # = 1,2,3
13) SECT[#]	- Indicates Section # for a Course, #=1,2,3
14) CRS-SECT	- Course Section
15) CRS-ID	- Course Identification Number
16) DESCRIPT	- Description of a Course
17) PREREQS	- Prerequisites of a Course
18) uselect	- User Select Key for a Menu Option
19) rec_data	- Data Entered by the User for a Data Entry
20) urqst	- User Layer Building Blcok REquest
21) frqst	- Faculty Data Layer Request
22) srqst	- Student Data Layer Request
23) rrqst	- Register Data Layer Request
24) crqst	- Course Data Layer Request
25) disp_rqst	- Display Request for a Menu or Data entry, or Output

6.2 Implementation of ULBB

Here I will explain the improvements made to the user layer building blocks and the logic behind the improvement. We will then discuss two individual implementations using RPC and ANSAWARE respectively. The logic remains the same for both the implementations.

6.2.1 Explanation of the Improvement

As we have discussed before, DIS had a program copied onto two different machines, using it to access the various building blocks of the application. The improvement that we have made here is to have a User Layer Building Block (ULBB) lying on a single server machine, and the client of the application will access it through its own program that will make a request for ULBB via the trader function. This client program will also access all the other building blocks by making a request through the trader. Thus we do not have to duplicate the entire source code onto different machines for being able to access it from different places. Now we can still access it from different machines, with the difference that the user layer building block is installed on only one machine, we call it a server for ULBB.

Thus the programmer wishing to access the functionality provided by the building blocks of the three layers have to write only a small bootstrap program of his/her own that simplifies the access to the numerous building blocks, functionalities which altogether make up a system. Of course, these functionalities should be interrelated in some way and should conform to our integration architecture rules.

6.2.2 Implementation Using RPC

As explained in earlier chapters, we define a building block using Sun RPC's protocol definition file. We therefore define the user layer building block by a similar file as shown in the APPENDIX A. It defines two operations, one is the submenu generator operation and the other is a menu for the processing layer query operations. The client program, through which the user accesses this building block, asks for the submenu or a processing layer menu depending on user request and the ULBB returns the screen information to the client program, which collects it and then properly displays it. It then asks for user selection, from this new menu. And hence depending on user selection takes further actions which may include building contracts with other building blocks.

Having built one ULBB, it makes us think about whether or not to have multiple user layer building blocks. If we have multiple building blocks, we can then have multiple views of the application and hence depending on the access rights of the user, the application intelligently makes contact with only one of those building blocks. This in my view is a very good way of providing views of a particular system and hence also adds to the security features of a particular application. This has been implemented as an experiment.

All these requests and responses come and go through the software bus called the trading function or a trader. The only undesired feature here is that the programmer has to know in advance the location of the user layer building block. Thus in case if its location is changed, a change should be made in the program and hence it has to be recompiled. This is a very important point for comparison with ANSAWARE implementation of the ULBB.

6.2.3 Implementation Using ANSAware

As explained in chapter 5, ANSAware uses the concept of SERVICE-INTERFACE for distributed application development. We make use of this approach to define our building blocks. An example was shown in chapter 5. We have used the IDL to define the building blocks in our system. Please refer to APPENDIX B and look for "ulbb.idl" - that defines the operations, arguments to these operations, and results of these operations of our ULBB.

Except for certain programming syntax and the use of ANSAware's trader mechanism, the logic for ULBB implementation remains the same. A very important point to note here is that the programmer does not have to know the physical whereabouts of the ULBB, it can be found by the local ANSAWARE trader, if the remote machine having the ULBB also has a ANSAware trader installed properly. This approach gives complete distribution transparency and hence is a major difference from our RPC implementation of ULBB. The same is true for any other building block. The programs for ULBB implementation are submenu.c (RPC) and ulbb.dpl (ANSAWARE).

6.3 Data Layer and Processing Layer Building Block

The building blocks in DIS do not conform to the principle of three conceptual layers. The processing layer is not clearly distinguished from the data layer. Also it has misinterpreted the concept of building blocks. It considers the small functions or routines that provide individual functionality such as add, delete, list, view etc. as building blocks, rather than a runnable process that as a whole will provide all of these functionality when asked for it via inter process communication mechanisms, and also communication over a network. Thus, EIS has a proper grouping of functionality and a building block is a process that runs by itself. Requesting program or a building blocks have to make communication with it and asks for a particular function. The building block definitions for the data layer building blocks are given in APPENDICES A and B, for RPC and ANSAWARE implementations respectively. The data layer definitions are in fdatalyr.x, cdatalyr.x, sdatalyr.x, rdatalyr.x for RPC; and fldbb.idl, cdlbb.idl, sdlbb.idl, rdlbb.idl for ANSAware implementation respectively.

Processing layer functionality refers to the functionality that requires access to operations from two or more individual data layer building blocks. Thus we have functionality like -

i) List of Courses Taught by a Faculty Member

This is a query which in database terms basically requires to join two relations. Thus this kind of query can be put in the processing layer. We just have one example here, but we could have many more queries for a full-fledged application.

We first ask the user for a Social Security Number of the Faculty member, with the help of which we can find out whether that faculty SSN is valid or not, by looking into the faculty database (functionality of Faculty Data Layer) and then try to match this SSN with the Faculty SSN of the course database records (functionality of Course Data Layer).

ii) Student Delete Operation

This is because the student delete operation requires a delete from student information as well as his/her registration information. And hence the correct way to delete a student is to delete his/her records from both the data collections at one time. This is the reason we have to keep it in the processing layer. This way it adds to the consistency features of the application.

The processing layer building block (PLBB) definitions can be found in APPENDICES A and B, in sproclyr.x (RPC) and proclyr.idl (ANSAWARE), respectively. The programs are sproclyr.c (RPC) and proclyr.dpl (ANSAWARE).

6.4 Experiments with Partial Results

Here I present some of the experiments that I tried to implement in the EIS, but were done only partially due to lack of time. Although these are partial results, they have provided us with further insight.

6.4.1 A Primitive Commit Protocol

One of our goals for providing consistency of data was to develop something like a transaction, so that in case of any failure or crash we can rollback the transaction. This would have provided recovery from a failure and hence a consistent data collection.

Let us consider the request going from the user layer building block to a data layer building block. And if there are many frequent requests made to the same building block (a DLBB), we want to differentiate them and also want to treat their operations as individual transactions. Meaning we want to preserve the states of operations done by these different request to the DLBB and in case of a failure or a catastrophe we want to reverse their effect and hence recover the original state of the data collection. But as was discussed in Chapter 4, Sun RPC implementation fails to preserve the results of the previous request to a DLBB and hence we can not accumulate these intermediate results. It is true that RPC fails to preserve the results of a RPC call over subsequent calls, and hence prevents us from implementing a primitive commit protocol, but this could be because of the way in which we define our operations. Say for example, that if we define our operations as individual transactions itself, then whenever a building block failure occurs or a crash happens we can take care of these transactions individually. Thus each RPC call that defines a particular operation is atomic by itself and hence we have to design our operations of the application more intelligently and take proper advantage of the characteristic of an RPC call.

6.4.2 Separating the Trader from the ULBB

The trading function as is implemented in DIS is associated with the User Layer Building Block and to the client application program in the EIS (RPC Version). As we know EIS has a differentiation between the client application program and the User Layer Building Block.

Therefore an attempt was made to separate the trader from this level (RPC Implementation) to its a separately running process/service of its own. Due to limitation of Sun RPC's implementation, we are able to pass and receive only single argument. We will have to take care of the packaging of message to be passed into one entity (a structure in C) and then unmarshall (unpack) it at the receiving end. This is doable and if done will be closer to the ideas of systems integration framework and the integration architecture discussed in this work.

What I have experimented with is the marshalling and the sending of a message to the required building block. This was an interesting experiment to do and can be considered as further improvements to the system. However it is important to note that ANSAware provides its own sophisticated trading function, with the help of which we can add, delete, list the building blocks running under that trader and also communicate to the other building blocks through that trader. Thus, this is the functionality that is anyway generically made available to building blocks implemented using ANSAware.

6.5 Installation and Configuration of the EIS system

As was decided in the design phase of the system, the user of the application should be able to access it from any of the three computer systems, viz., Newark, Pluto and Irss. It was decided that the system should have a server for the ULBB and the administrator of the system should be able change the configuration by changing the locations of the building blocks. This can be done by changing the internal directory information on the building blocks and recompile the source code before anybody can use it from its new location. This holds for the RPC implementation of the EIS. The implementation using ANSAware is on a single machine for now, but in the future it can be done on different machines once all the participating systems have ANSAware installed on them. Furthermore, ANSAware is more flexible with regard to relacation of resources and building blocks.

Location of the Building Blocks

The User Layer Building Block is running on the Newark machine. Various Data Layer Building Blocks are running on different locations. The Register Data Layer Building Block is running on Newark. The Course, Faculty and Student Data Layer Building Blocks are running on Pluto. And the bootstrap program for being able to access all building blocks has to be run from IRSS, as per its current configuration. But this can be changed, so that the program can run from any machine and access the functionality of the EIS system. The the EIS system has its building blocks organized in such a way that the user, depending on his/her needs, would have to have a client program that accesses only the required building blocks. Thus, in this way we can provide different views of the system. The processing layer building block is running on Newark, and as explained previously, it accesses the data layer building blocks via the RPC-implementation of the trader.

CHAPTER 7

COMPARISON OF THE TWO IMPLEMENTATIONS

Let us put some light on certain similarities and differences between RPC and ANSAware. This discussion is based on the discussion of Chapters 4 and 5, so the reader is advised to read those two chapters first before reading this chapter.

7.1 Handling of Client-Requests

As we have seen RPC follows a mechanism similar to local procedure call and ANSAware follows a Import-Export concept for handling request from client programs and/or from building blocks. The Import-Export aspect of ANSAware is built on the underlying instantiation mechanism of the provider building block interface. Thus, we may have more than one instances of a building block running at one time, which is not possible to have using RPC directly.

7.2 Message Passing

The Sun RPC implementation used here for EIS limits the sending and receiving message to a single argument/single result mechanism. Multiple arguments and/or results have to be packaged together into a single message. Whereas in ANSAware the building blocks can send or receive multiple arguments/results freely. This takes off lot of marshalling/unmarshalling work from the programmer, which he/she would have done if using RPC.

7.3 Location Transparency of the Building Blocks

While using RPC to develop the a distributed application the programmer has to know the explicit location of the server/provider building block and has to use the programmer does not have to worry about the physical location of a building block as long as it is registered with the ANSAware trader. If it is on a remote machine, the trader on the client/local machine has to communicate to the trader on the remote machine to find the provider building block. This is a significant feature in terms of location transparency of building blocks, even to the programmer, so that he/she can concentrate more on programming rather than networking.

Now let us consider what happens in certain exceptional conditions during the communication of these building blocks that we have been talking about. Here I address a few of them, both, for the RPC and the ANSAware implementation.

7.4 Non-Existence of a Building Block

Let us assume that a building block on one layer is requesting a operation from a building block on the same/different layer. This request as we know has to go through the trading function, which will in turn try to find if the requested building block operation is available or not. Suppose that the provider building block has not registered itself with the trader, then the trader should detect this and let the requester building block know about its non-existence. Now let us see what happens in the two implementations that we have.

A Remote Procedure Call implementation, tries to ping the service provider before actually asking for an operation to be performed. Pinging the server means, checking to see if it really exist or not. Thus if this pinging fails, it can be carried forward to the requester in the form of a message which says that the binding to the server building block has failed.

An ANSAware implementation, tries to import the interface of the provider building block, which is similar to binding to a building block as in

RPC. But in this case it tries to find the server and if not found does more than just informing about its non-existence. What it does more is that it tries to look for a possible dynamic, active or passive relocation of the server building block. If the relocation is detected and the building block is found at some other place, the import is successful; otherwise, a Bind Failure is sent to the requester. One important point to note here is that the ANSAware trader tries to find the server till a specific period of time has passed. This means that the trader looks for the service provider only for a specific time period and then, if not successful, time-out occurs, resulting in an unsuccessful request.

7.5 Delayed Response or No Response

Sometimes, even after binding to a provider building block interface, the requester does not receive a response, before a time out occurs. There can be two situations here. One is that the service operation is taking too long to complete. Another situation could be that due to some malfunctioning of the server or the remote system, the response gets lost. The issue here is that the requester can not wait for a Response eternally, and hence again a time-out approach is adapted by both RPC and ANSAware implementations.

ANSAware has an option similar to exception conditions. It allows for exception handling to be defined in its Distributed Processing Language, compiled using the PREPC compiler. Thus the programmer can define his/her exceptional handling actions while defining the building block interfaces, using IDL.

7.6 Failure of a Client or a Server

Here we will consider the implications of the failure/crash of either a client building block or a server building block while in the middle of an operation that directly changes the state of the data repositories.

Client Failure

If a client building block fails or dies for whatsoever reason while in the middle of a critical operation that directly changes the state of the data being operated upon, then due to distributed nature of the environment, failure of a client building block is not obvious to the server building block. Hence, the failure of client building block, does not affect the execution of the server building block and thus in turn the data repositories. The implication of this failure is just that the client is unable to look at the response from the server and hence the user would not know the result of the operation. But more important thing is that the state of the data remains valid and consistent.

Server Failure

If a server building block fails in the middle of a critical operation, then the consequences are very important to discuss. In this case if the operation was completed and then while sending the results back the building block failed, the data is not affected in any way. But if the operation fails in such way that it has changed the state of the data partially then it makes the information stored inconsistent an hence needs to be checked. In order to avoid this kind of situation a transaction manager kind of tool has to be used to restore the original valid state of the information. But none of our implementations have seen any success in that direction. If ANSAware is used to its fullest capacity with factory and thread-of-transaction mechanism, a future implementation of EIS can see these transaction management features.

7.7 Conclusions

Implementing EIS, though being very simple in functionality, using both RPC and ANSAware has been very interesting and it has been a quite a project to study the various issues that we have discussed. And fun part was in really implementing them to see what really happens and how this reciprocates to the concepts behind the design, framework and the conceptual architecture.

1. Effect of GenSIF and the integration architecture

As was mentioned earlier, one of the goals of this work was to study the effect of the Generic Systems Integration Framework and the example building block architecture on the development process of a distributed application. When I began working on this project, I started designing the system in a standard way around its required operations, grouping the functionality according to the category of operations rather than entities. But once I started to change my preliminary design to incorporate the concepts of the integration architecture I began grouping functionality according to the entities involved and according to the separation of concerns principle that we have discussed earlier. Thus the key factor of the design was in deciding on a building block.

2. Limitations of RPC and its effects

Due to the certain limitations of Sun's RPC, such as single argument message passing, explicit knowledge of the location of the server, and using different IDs for each RPC call we were restricted in implementing very important issues concerning the operations of the building block and also the consistency of the data therein. Thus I feel that Sun's RPC though is a very nice tool to use is not suitable enough for a complete of a system being developed in the given framework under architectural constraints/rules.

3. Sophistication of ANSAware

Although the implementation of EIS using ANSAware is as simple as in RPC, it puts light on some very important features of ANSAware that are of very good interest to the research going on in Systems Integration. These features are : being able to relocate the server building block before actually timing out the request for a remote operation, threads used by ANSAware that make transaction processing doable in such an environment, its ability to instantiate the building block interfaces, and encapsulation of operations and objects. The most important feature of all is the trading function that it provides to add/remove/move building blocks to/from the application, and communicate to any available service without actually knowing its location.

APPENDIX A

Here we present some of the important source code listings for better understanding of the implementation of EIS. Some listings have been ommited since they are not really necessary to understand the concepts behind the implementation.

APPENDIX A

/* cdatalyr.x : protocol definition for course DLBB*/

```
%#define CDATABASE "course.data" /* '%'Passes definition*/
const CMAX_STR = 256;      /* through to header file */
/* <MAX_STR> defines the maximum possible length */
/* No enumerations needed for structure definition */
/*********      Record Structure for course.data *******/
struct crsrec {
            string crsid<6>;
            string crssection<3>;
            string crssemester<6>;
            string crsdescr<30>;
            string location<10>;
            string crsfssn<11>;
            int crscredits;
            string prereqs<40>;
            struct crsrec  *next;
            struct crsrec *prev;
      };

/* No union or typdef needed for program definition */
```

```
program CDATALYRPROG {  /* Can manage multiple servers */
    version CDATALYRVERS {
            string  CLIST_RECORD(void)       = 1;
            string  CVIEW_RECORD(string)       = 2;
            int    CADD_RECORD(crsrec)      = 3;
            int    CDEL_RECORD(string)      = 4;
    } = 1;
} = 0x30000006; /*Program # ranges established by ONC */

/*fdatalyr.x : protocol definition for FDLBB*/
%#define FDATABASE "faculty.data" /* '%'Passes definition */
const MAX_STR = 256;       /* through to header file */
/* <MAX_STR> defines the maximum possible length */

/* No enumerations needed for structure definition */
/*********       Record Structure for Faculty.data *******/
struct record {
            string ssn<MAX_STR>;
            string firstName<MAX_STR>;
            string middleInitial<MAX_STR>;
            string lastName<MAX_STR>;
            string phone<12>;
            string location<MAX_STR>;
            struct record  *next;
            struct record *prev;
       };
/* No union or typdef needed for program definition */
```

```
program FDATALYRPROG {  /* Can manage multiple servers */
    version FDATALYRVERS {
            string  LIST_RECORD(void)       = 1;
            string  VIEW_RECORD(string)        = 2;
            int     ADD_RECORD(record)      = 3;
            int     DEL_RECORD(string)      = 4;
    } = 1;
} = 0x20000006; /*Program # ranges established by ONC */


/* sdatalyr.x : protocol definition for Student DLBB*/
%#define SDATABASE "student.data"      /* '%' Passes definition */
const SMAX_STR = 256;       /* through to header file */


/* <SMAX_STR> defines the maximum possible length */


/* No enumerations needed for structure definition */
/*********       Record Structure for student.data *******/
struct stdrec {
            string stdssn<11>;
            string stdfname<30>;
            string stdmname<30>;
            string stdlname<30>;
            string stdphone<12>;
            string stdbdate<8>;
            string stdlevel<1>;
            string stdmajor<30>;
            string stdminor<30>;
```

```
                string stdgpa<5>;
                string stdenrolled<1>;
                struct stdrec *next;
                struct stdrec *prev;
            } ;
/* No union or typdef needed for program definition */
program SDATALYRPROG {    /* Can manage multiple servers */
     version SDATALYRVERS {
                string  SLIST_RECORD(void)       = 1;
                string  SVIEW_RECORD(string)       = 2;
                int     SADD_RECORD(stdrec)      = 3;
                int     SDEL_RECORD(string)      = 4;
     } = 1;
} = 0x34000006; /*Program # ranges established by ONC */
```

```
/* rdatalyr.x : protocol definition for Registration DLBB*/
%#define RDATABASE "register.data"
/* '%' Passes definition */
const RMAX_STR = 256;    /* through to header file */
/* <RMAX_STR> defines the maximum possible length */
/* No enumerations needed for structure definition */
/*********      Record Structure for student.data *******/
struct rrec {
                string tssn<11>;
                string tcrs1<6>;
                string tsec1<3>;
                string tcrs2<6>;
```

```
            string tsec2<3>;
            string tcrs3<6>;
            string tsec3<3>;
            struct rrec *next;
            struct rrec *prev;
        };
/* No union or typdef needed for program definition */
program RDATALYRPROG {   /* Can manage multiple servers */
    version RDATALYRVERS {
            string  RLIST_RECORD(void)       = 1;
            string  RVIEW_RECORD(string)       = 2;
            int     RADD_RECORD(rrec)       = 3;
            int     RDEL_RECORD(string)       = 4;
    } = 1;
} = 0x37000006; /*Program # ranges established by ONC */


/* submenu.x : RPCL protocol definition for a remote ULBB*/


program ULYRPROG {    /* Can manage multiple servers */
    version ULYRVERS {
            string  AMENU(void)      = 1;
            string  PMENU(void)      = 4;
    } = 1;
} = 0x39000006; /*Program # ranges established by ONC */
/*
**CISMENU.C
```

```
**CALLS SERVICE MENUS and depending on response calls the
proper activity service building blocks.
*/
#include "dis.h"
extern void trader();
main()
{
     char ch;
     int stat = 0;
mstart :
     initscr();
     printf("\n\n\t\t---------------------------------\n");
     printf("\t\t\t\tCIS DEPT. INFO. SYSTEM\n");
     printf("\t\t-----------------------------------\n\n");
     printf("\t\t\t<F/f> Faculty Info. System\n\n");
     printf("\t\t\t<C/c> Course Info. System\n\n");
     printf("\t\t\t<S/s> Student Info. System\n\n");
     printf("\t\t\t<R/r> Registration Info. system\n\n");
     printf("\t\t\t<P/p> sPecialized Queries\n\n");
     printf("\t\t\t<E/e> Exit this menu\n\n");
     printf("\n\t\t\t Enter Selection : ");
     ch = tolower(getresponse());
     if (ch == 'f' || ch == 'c' || ch == 's' || ch == 'r' ||
ch == 'p')
          menu(ch);
     else if (ch == 'e')
     {
        initscr();
        printf("\t\tCIS MENU : Exiting...\n");
        exit(-1);
    }
    else
    {
        error('e',"Invalid Selection, Select Again\n");
    }
    goto mstart;
}
/*
**      menu.c
**      SUB MENU PROGRAM.
*/
#include "dis.h"
void menu(ch)
    char ch;
{
    int init, code;
    static int u = 0, uhndl = 0, p = 0, phndl = 0;
    char title[35], c, **kmenu;
    switch(tolower(ch))
    {
        case 'f' :
            init = FSTART;
            strcpy(title,"Faculty Operations Menu");
            break;
```

```
                case 'r' :
                     init = RSTART;
                     strcpy(title,"Registration Operations Menu");
                     break;
                case 's' :
                     init = SSTART;
                     strcpy(title,"Student Operations Menu");
                     break;
                case 'c' :
                     init = CSTART;
                     strcpy(title,"Course Operations Menu");
                     break;
           } /* switch(ch) */
      mstart :
           initscr();
           if (tolower(ch) != 'p')
           {
                logo(title);
                if (!u)
                {
                     if((uhndl = ulyr_handle()))
                          u = 1;
                }
                if (uhndl)
                {
                     kmenu = amenu_1(NULL,ulyr_clnt);
                     if (!strcmp(*kmenu,""))
```

```
          {
                error('e',"Not Found");
                sleep(5);
          }
          else
                printf("%s\n", *kmenu);
    }
    printf("\t\tEnter Selection : ");
    c = tolower(getresponse());
    printf("\n");
    switch(c)
    {
          case 'a' :
                code = init + ADD;
                break;
          case 'd' :
                code = init + DEL;
                break;
          case 'l' :
                code = init + LIST;
                break;
          case 'v' :
                code = init + VIEW;
                break;
          case 'u' :
                code = init + UPD;
                break;
```

```
                    case 'e':
                         return;
                    default :
                         error('e',"Invalid Selection, Select
Again");
                         ch = 'K';
                         goto mstart;
               } /* switch() */
          }
          else if (ch == 'p')
          {
               logo("Specialized Queries");
               if (!u)
               {
                    if((uhndl = ulyr_handle()))
                         u = 1;
               }
               if (uhndl)
               {
                    kmenu = pmenu_1(NULL,ulyr_clnt);
                    if (!strcmp(*kmenu,""))
                    {
                         error('e',"Not Found");
                         sleep(5);
                    }
                    else
                         printf("%s\n", *kmenu);
```

```
            }
            printf("\t\tEnter Selection : ");
            c = tolower(getresponse());
            printf("\n");
            if (c == 'm')
                 code = MISC_FC;
            else if (c == 'e')
                 return;
       }
       trader(code);
       goto mstart;
}
/*
**          TRADER.C
*/
#include "dis.h"
extern FNODE make_fnode();
extern RNODE make_rnode();
extern SNODE make_snode();
extern CNODE make_cnode();
void trader(code)
       int code;
{
       FNODE rec, new;
       CNODE crec, cnew;
       SNODE srec, snew;
       RNODE trec, rnew;
```

```
	static int fdl = 0, hndl = 0, cdl = 0, chndl = 0, sdl =
0, shndl = 0;
	static int spl = 0, sphndl = 0, rdl = 0, rhndl = 0;
	static int mpl = 0, mphndl = 0;
	int *stat;
	char **lstr, **cstr, **sstr, **mstr, *ssn, *crsid,
*sec;
	if (code == MISC_FC)
	{
		printf("Enter the SSN for Faculty : ");
		ssn = (char *) malloc(15);
		gets(ssn);
		if (!mpl)
		{
			if((mphndl = sproclyr_handle()))
				mpl = 1;
		}
		if (mphndl)
		{
			mstr = misc_fac_1(&ssn, sproclyr_clnt);
			if (!strcmp(*mstr,""))
			{
				error('e',"Not Found");
				sleep(5);
			}
			else
				printf("%s\n", *mstr);
```

```
                }
            } /* MISC_FC() */
        /*
        **    Faculty Information - data Lalyer Bldg. Block
        Functionality.
        */
            if (code >= FADD && code <= FUPD)
            {
                if((rec = (FNODE) malloc(sizeof(fdata))) == NULL)
                {
                    error('e',"Mem. Fault\n");
                    return;
                }
              if (!fdl)
              {
                if((hndl = fdatalyr_handle()))
                    fdl = 1;
              }
              if (hndl)
              {
                switch(code)
                {
                    case FADD :
                        rec = make_fnode(new);
                        stat = add_record_1(rec, fdatalyr_clnt);
                        if (*stat == 0)
                            error('e',"FADD_ERROR::1");
                    break;
                case FLIST :
                    lstr=list_record_1(NULL,fdatalyr_clnt );
                    if (!strcmp(*lstr,""))
                         error('e',"No Match Found");
                    else
                    {
                         system("clear");
                         printf("%s\n", *lstr);
                    }
                    break;
                case FVIEW :
                    printf("Enter SSN: of the Faculty to
View : ");
                    ssn = (char *) malloc(20);
                    ssn = gets();
                    lstr=view_record_1(&ssn,fdatalyr_clnt);
                    if (*lstr == "")
                         error('e',"No Match Found");
                    else
                    {
                         system("clear");
                         printf("%s\n", *lstr);
                    }
                    free(ssn);
                    break;
                case FDEL :
                    printf("Enter SSN: of the Faculty to
Delete: ");
                    ssn = (char *) malloc(20);
                    ssn = gets();
                    ssn[11] = '\0';
                    printf("SSN:%s\n", ssn);
                    stat = del_record_1(&ssn,fdata1yr_clnt);
                    printf("stat = %d\n", *stat);
                    if (*stat == 0)
                         error('e',"FDEL_ERROR::1, Does not
Exist");
                    else
                         printf("SUCCESSFULL DELETION\n");
                    free(ssn);
                    break;
               case FUPD :
               printf("Enter SSN : (999-99-9999) :");
               ssn = (char *) malloc(20);
               ssn = gets();
               ssn[11]='\0';
               sstr = view_record_1(&ssn, fdata1yr_clnt);
               if (*sstr=="")
                    printf("SSN :: UPD :: does not
exist\n");
               else
               {
                    system("clear");
```

```
                        printf("%s\n", *sstr);
                printf("\n\nNow Enter the Data Again for SSN :
%s\n", ssn);
                    sleep(2);
                    }
                    stat = del_record_1(&ssn, fdatalyr_clnt);
                    if (*stat == 0)
                printf("FDL::Unsuccessful Update, Please Check
it\n");
                    rec = make_fnode(new);
                    stat = add_record_1(rec, fdatalyr_clnt);
                    if (*stat == 0)
                printf("FAD::Unsuccessful Update, Please Check
it\n");
                    free(ssn);
                        break;
                    default :
                        error('e',"Invalid Service
Code,Unrecognized");
                        break;
                } /* switch() */
                sleep(2);
              } /* if ..._handle() */
            } /* code >= .. && code <= ... */
 /*     if (code==SADD || code==SVIEW || code==SLIST ||
 code==SUPD)
            {
```

```
/*We are ommitting the portion for Student datalayer
Requests, as it is similar to the faculty data layer
operatrions.*/
     }
     else if (code == SDEL)
     {
     /* Here is a Student  Delete operation that is
     accessed through the processing layer BB.*/
        if (!spl)
        {
          if((sphndl = sproclyr_handle()))
               spl = 1;
        }
        if (sphndl)
        {
          printf("Enter SSN (999-99-9999): ");
          ssn = (char *) malloc(20);
          ssn = gets();
          ssn[11] = '\0';
          printf("ssn = %s\n", ssn);
          stat=psdel_record_1(&ssn,sproclyr_clnt);
          if (*stat == 0)
               error('e',"SDEL_ERROR::1, Does not Exist");
        }
     }
/*Course/Registration data Lalyer Bldg. Block Functionality.
```

```
are similart to FDLBB operations, so they are not listed
here.*/
return;
} /* trader() */
/* Input.c is another which is used by trader() function. It
is a program for Data Entry routines, make_fnode, make_rnode
etc. This is program is merely getting input from the
user,so it is not listed here.*/
/*
**    Fdatalyr.c
**    Remote Database Service Procedures
**    Only The Add_record service is completely detailed;
Please Note that only the Fdatalyr.c is listed here. Other
programs such as sdatalyr.c, cdatalyr.c, rdatalyr.c are the
same
*/
#include <fcntl.h>
#include <string.h>
#include "dis.h"

int *add_record_1(new1)
     record *new1;
{
     static int stat = 0;
     char ssn[256];
     char *newrec, *irec;
     int fd, rqst, found = 0;
```

```
        sprintf(ssn,"%-11s", new1->ssn);
        if ((fd = open(FDATABASE,O_RDWR,0700)) == -1)
        {
                error('e',"Can not Open \n");
                if ((fd=creat(FDATABASE,0700)) == -1)
                {
                        error('e',"Can not Create");
                        exit(-1);
                }
        }
        else
        {
                irec = (char *) malloc(30);
                lseek(fd,0L,0);
                while (read(fd, irec, 11))
                {
                        irec[11] = '\0';
                        if (!(stat = strcmp(irec, ssn)))
                        {
                                error('e',"Duplicate Record
Detected\n");
                                return(&stat);
                        } /* if */
                        lseek(fd,155L,1);
                } /* while */
        } /* else */
```

```
      sleep(5);

      newrec = (char *) malloc(256*6);

      sprintf(newrec,"%-11s|%-30s|%-30s|%-30s|%-30s|%-30s",

      newl->ssn, newl->firstName, newl->middleInitial, newl-
>lastName,

      newl->phone, newl->location);


      rqst = strlen(newrec) - 1;

      lseek(fd, 0L, 2);

      if ((write(fd, newrec, rqst) != rqst))

      {

            printf("\0007");

            stat = 0;

            error('e',"Can not write the requested number");

      }

      else

      {

            stat = 1;

            write(fd, "\n", 1);

      }

      free(newrec);

      if (TORF) printf("ADD done, RETURN THE SUCCESS/FAILURE
CODE\n");

      close(fd);

      return(&stat);

}
```

```
char **list_record_1()
{
	char lstbuf[256];
	static char *outrec = NULL;
	FILE *fp;


	if ((outrec = (char *) malloc(3700)) == NULL)
	{
		error('e',"Mem.Fault");
		return(NULL);
	}

	strcpy(outrec,"");

	if ((fp = fopen(FDATABASE,"r")) == NULL)
	{
		error('e',"Nothing to List");
		strcpy(outrec,"Nothing To List");
		return(&outrec);
	}
	while(fgets(lstbuf,256,fp))
	{
		lstbuf[strlen(lstbuf)-1] = '\0';
		if (!strcmp(lstbuf,"\n"))
			break;
```

```
            strcat(lstbuf,"\n");
            strcat(outrec,lstbuf);
        }
        fclose(fp);
        if (!strcmp(outrec,""))
        {
            strcpy(outrec,"");
            return(&outrec);
        }
        else
            return(&outrec);
}
char **view_record_1(ssn)
        char **ssn;
{
        int stat = 1, i = 0;
        static char *retstr;
        char *irec, rssn[15], *flds[256];
        FILE *fp;
        sprintf(rssn,"%-11s", *ssn);
        if ((fp = fopen(FDATABASE,"r")) == NULL)
        {
            error('e',"Can not Open \n");
            strcpy(retstr,"");
            return(&retstr);
        }
        else
        {
              retstr = (char *) malloc(356);
              while ((stat) && (fgets(retstr,256,fp)))
              {
                    irec = (char *) malloc(356);
                    strcpy(irec, retstr);
                    for(i=0;i<6;i++)
                          if((flds[i]=strtok(irec,"|")) != NULL)
                                irec = NULL;
                    stat = strcmp(rssn, flds[0]);
              } /* while */
        } /* else */
        if(!stat)
        {
              fclose(fp);
              return(&retstr);
        }
        else
        {
              fclose(fp);
              strcpy(retstr,"");
              return(&retstr);
        }
  }
  int *del_record_1(ssn)
        char **ssn;
  {
```

```
static int ret = 1;
FILE *fp, *ofp;
int stat = 0, i;
char dssn[15], *flds[256], *drec, s[256];
if ((fp=fopen(FDATABASE,"r")) == NULL)
{
     error('e',"Can not Open");
     ret = 0;
     return(&ret);
}
if ((ofp=fopen("tempfac.data","a+")) == NULL)
{
     error('e',"Can not Open");
     ret = 0;
     return(&ret);
}
printf("DSSN=%s\n", dssn);
sprintf(dssn,"%-11s", *ssn);
drec = (char *) malloc(256);
while(fgets(drec,256,fp))
{
     printf("DREC=%s\n", drec);
     strcpy(s,drec);
     for(i=0;i<MAX;i++)
          if((flds[i]=strtok(drec,"|")) != NULL)
               drec = NULL;
          else
```

```
                    break;
                if (!(stat=strcmp(dssn,flds[0])))
                    continue;
                fputs(s,ofp);
                drec = (char *) malloc(256);
        } /* while() */
        if (stat != 0)
                ret = 0;
        fclose(fp);
        fclose(ofp);
        remove(FDATABASE);
        rename("tempfac.data",FDATABASE);
        return(&ret);
}
/*
**      Sproclyr.c
**      Remote Database Service Procedures  This Illustrates
the Processing layer fuynctionality
*/
#include <fcntl.h>
#include <string.h>
#include "dis.h"
/*
**      This is a Function (Service) to Delete a Student.
*/
int *psdel_record_1(ssn)
        char **ssn;
```

```
{
     static int stat = 0;
     char *tssn;
     tssn = (char *) malloc(20);
     strcpy(tssn, *ssn);
     if (sdatalyr_handle())
          stat = *sdel_record_1(&tssn, sdatalyr_clnt);
     if (!stat)
          return(&stat);
     if(rdatalyr_handle())
          stat = *rdel_record_1(&tssn, rdatalyr_clnt);
     if (!stat)
          return(&stat);
}
char **misc_fac_1(ssn)
     char **ssn;
{
     int i =0, k = 0, j = 0;
     static char *outrec;
     char **cstr, *cistr[256];
     char *fssn, *clist, *crec[656], *flds[256], *crecptr;
     char **fview;
     if((outrec = (char *) malloc(5000)) == NULL)
     {
          strcpy(outrec,"");
          return(&outrec);
     }
```

```
        clist = (char *) malloc(5000);

        crecptr = (char *) malloc(300);

        fssn = (char *) malloc(20);

        strcpy(outrec,"");

        strcpy(fssn, *ssn);

        if (fdatalyr_handle())

        {

              fview = view_record_1(&fssn, fdatalyr_clnt);

              if (!strcmp(*fview,""))

              {

                    strcpy(outrec,"Invalid Faculty SSN :: ");

                    return(&outrec);

              }

              strcpy(outrec,"Information on Requested Faculty is

  : \n");

              strcat(outrec, *fview);

              strcat(outrec,"\nNow Displaying The Info. on

  his/her Courses\n");

        }

        if (cdatalyr_handle())

        {

              cstr = clist_record_1(NULL, cdatalyr_clnt);

              if(!strcmp(*cstr,""))

              {

                    strcat(outrec,"\nNo Course Taken by Requested

  Faculty");

                    return(&outrec);
        }
        strcpy(clist, *cstr);
        for (i=0;i<MAX;i++)
        {
                if((crec[i]=strtok(clist,"\n")) != NULL)
                        clist = NULL;
                else
                        break;
        }
        for(k=i-1;k>=0;k--)
        {
                strcpy(crecptr,crec[k]);
                for(j=0;j<MAX;j++)
                {
                        if((flds[j]=strtok(crecptr,"|")) !=
NULL)
                                crecptr = NULL;
                        else
                                break;
                }
                if(!strcmp(flds[5], fssn))
                {
                        strcat(outrec,"\n");
                        strcat(outrec,crec[k]);
                }
                crecptr = (char *) malloc(300);
        } /* () */
```

```
     } /* cdat.._han..() */
     return(&outrec);
} /* misc_fac() */
/*

**   submenu.c
**   Remote Database Service Procedures
*/
#include <string.h>
#include "ulyr.h"
char **amenu_1()
{
     static char *mnu;
     mnu = (char *) malloc(400);
     strcpy(mnu,"");
     strcat(mnu,"\n\t\t\t\t<A/a>  Add Information\n");
     strcat(mnu,"\n\t\t\t\t<D/d>  Delete Information\n");
     strcat(mnu,"\n\t\t\t\t<L/l>  List Information\n");
     strcat(mnu,"\n\t\t\t\t<V/v>  View Information\n");
     strcat(mnu,"\n\t\t\t\t<U/u>  Update Information\n");
     strcat(mnu,"\n\t\t\t\t<E/e>  Exit This Menu\n");
     return(&mnu);
}
char **pmenu_1()
{
     static char *mnu;
     mnu = (char *) malloc(400);
```

```
        strcpy(mnu,"");
        strcat(mnu,"\n\t\t\t\t<M/m>  Faculty Course List\n");
        strcat(mnu,"\n\t\t\t\t<E/e>  Exit This Menu\n");
        return(&mnu);
}
```

/* Now We List the Makefiles */

/*Faculty Building Block Makefile */

```
FDLSERVOBJS = fdatalyr.o fdatalyr_svc.c fdatalyr_xdr.o
error.o init.o input.c
HEADERFILES = dis.h fdatalyr.h
FDATALYRGEN = fdatalyr_clnt.c fdatalyr_svc.c fdatalyr_xdr.c
fdatalyr.h
fdatalyrbb : $(FDLSERVOBJS) $(HEADERFILES)
        cc -g -o fdatalyrbb $(FDLSERVOBJS)
fdatalyr.o : fdatalyr.c
        cc -g -c -o fdatalyr.o fdatalyr.c
fdatalyr_svc.o : fdatalyr_svc.c
        cc -g -c -o fdatalyr_svc.o fdatalyr_svc.c
input.o : input.c
        cc -g -c -o input.o input.c
fdatalyr_xdr.o : fdatalyr_xdr.c
        cc -g -c -o fdatalyr_xdr.o fdatalyr_xdr.c
error.o : error.c
        cc -g -c -o error.o error.c
init.o : init.c
        cc -g -c -o init.o init.c
$(FDATALYRGEN) : fdatalyr.x
```

```
     rpcgen fdatalyr.x

/* Processing Layer BB makefile*/

SPROCSERVOBJS = sproclyr.o sproclyr_svc.o sproclyr_xdr.o
error.o init.o prochandle.o sdatalyr_clnt.o rdatalyr_clnt.o
sdatalyr_xdr.o  rdatalyr_xdr.o cdatalyr_clnt.o
cdatalyr_xdr.o fdatalyr_clnt.o fdatalyr_xdr.o
HEADERFILES = dis.h sproclyr.h sdatalyr.h rdatalyr.h
cdatalyr.h
SPROCLYRGEN = sproclyr_clnt.c sproclyr_svc.c sproclyr_xdr.c
sproclyr.h
SDATALYRGEN = sdatalyr_clnt.c sdatalyr_svc.c sdatalyr_xdr.c
sdatalyr.h
RDATALYRGEN = rdatalyr_clnt.c rdatalyr_svc.c rdatalyr_xdr.c
rdatalyr.h
CDATALYRGEN = cdatalyr_clnt.c cdatalyr_svc.c cdatalyr_xdr.c
cdatalyr.h
FDATALYRGEN = fdatalyr_clnt.c fdatalyr_svc.c fdatalyr_xdr.c
fdatalyr.h
sproclyrbb : $(SPROCSERVOBJS) $(HEADERFILES)
     cc -g -o sproclyrbb $(SPROCSERVOBJS)
sproclyr.o : sproclyr.c
     cc -g -c -o sproclyr.o sproclyr.c
sproclyr_svc.o : sproclyr_svc.c
     cc -g -c -o sproclyr_svc.o sproclyr_svc.c
sproclyr_xdr.o : sproclyr_xdr.c
```

```
     cc -g -c -o sproclyr_xdr.o sproclyr_xdr.c
sdatalyr_clnt.o : sdatalyr_clnt.c
     cc -g -c -o sdatalyr_clnt.o sdatalyr_clnt.c
rdatalyr_clnt.o  : rdatalyr_clnt.c
     cc -g -c -o  rdatalyr_clnt.o rdatalyr_clnt.c
sdatalyr_xdr.o : sdatalyr_xdr.c
     cc -g -c -o sdatalyr_xdr.o sdatalyr_xdr.c
rdatalyr_xdr.o : rdatalyr_xdr.c
     cc -g -c -o rdatalyr_xdr.o rdatalyr_xdr.c
cdatalyr_clnt.o : cdatalyr_clnt.c
     cc -g -c -o cdatalyr_clnt.o cdatalyr_clnt.c
cdatalyr_xdr.o : cdatalyr_xdr.c
     cc -g -c -o cdatalyr_xdr.o cdatalyr_xdr.c
fdatalyr_clnt.o : fdatalyr_clnt.c
     cc -g -c -o fdatalyr_clnt.o fdatalyr_clnt.c
fdatalyr_xdr.o : fdatalyr_xdr.c
     cc -g -c -o fdatalyr_xdr.o fdatalyr_xdr.c
error.o : error.c
     cc -g -c -o error.o error.c
init.o : init.c
     cc -g -c -o init.o init.c
prochandle.o : prochandle.c
     cc -g -c -o prochandle.o prochandle.c
$(SPROCLYRGEN) : sproclyr.x
     rpcgen sproclyr.x
$(RDATALYRGEN) : rdatalyr.x
     rpcgen rdatalyr.x
```

```
$(SDATALYRGEN) : sdatalyr.x
     rpcgen sdatalyr.x
$(CDATALYRGEN) : cdatalyr.x
     rpcgen cdatalyr.x
$(FDATALYRGEN) : fdatalyr.x
     rpcgen fdatalyr.x

/* The User Layer BB Makefile*/
NEWMENUOBJS = submenu.o ulyr_svc.c error.o init.o
HEADERFILES = ulyr.h
ULYRGEN = ulyr_clnt.c ulyr_svc.c ulyr.h
newsbb : $(NEWMENUOBJS) $(HEADERFILES)
     cc -g -o newsbb $(NEWMENUOBJS)
submenu.o : submenu.c
     cc -g -c -o submenu.o submenu.c
ulyr_svc.o : ulyr_svc.c
     cc -g -c -o ulyr_svc.o ulyr_svc.c
error.o : error.c
     cc -g -c -o error.o error.c
init.o : init.c
     cc -g -c -o init.o init.c
$(ULYRGEN) : ulyr.x
     rpcgen ulyr.x

/* The Header File for EIS is DIS.H, which as follows */
#include <stdio.h>
#include <string.h>
```

```
#include <ctype.h>
#include <rpc/rpc.h>
#include "fdatalyr.h" /* on newark remove this */
#define TORF   1
#define COMMIT      1
#define MAX    256
#define MAXFLDS 20
#define        SEEK_SET  0
#define        SEEK_CUR  1
#define        SEEK_END  2
#define   FSTART    0
#define CSTART 4
#define SSTART 8
#define RSTART 12
#define ADD    1
#define DEL    2
#define   VIEW 3
#define   LIST 4
#define   FADD 1
#define   FDEL 2
#define        FVIEW     3
#define        FLIST     4
#define   CADD 5
#define        CDEL 6
#define   CVIEW     7
#define        CLIST     8
#define        SADD 9
```

```
#define         SDEL 10
#define         SVIEW     11
#define         SLIST     12
#define         RADD 13
#define         RDEL 14
#define         RVIEW     15
#define         RLIST     16
#define    MISC_FC 17
#define    EXIT 21
#define    FDATALYRSERV   "pluto" /* pluto */
CLIENT *fdatalyr_clnt; /* remove on NEWARK, only on PLUTO */
struct record fdata;
typedef struct record *FNODE;
/* This is all for the RPc Version of EIS*
The next Appendix  illustrates the application developed by
using ANSAware*/
```

APPENDIX B

Now, here we illustrate the ANSAware Implementation of the EIS system. Here we list the source codes of the Interface Definition Languagefiles for each of the building bblocks and the Distributed Processing Language routines for each of them.

```
/* Faculty Data Layer BB, "fdlbb.idl and "fdlbb.dpl*./
fdlbb : INTERFACE =
BEGIN
     FRECORD : TYPE = RECORD [
      FSSN   : STRING,
      FNAME : STRING,
      MNAME : STRING,
      LNAME : STRING,
      PHONE : STRING,
      LOCATION : STRING
      ];

      fadd : OPERATION [fnew : FRECORD ] RETURNS [fstat :
INTEGER ];
      fdel : OPERATION [fssn : STRING ] RETURNS  [fstat :
INTEGER ];
      fview : OPERATION [fssn : STRING] RETURNS [fstat :
STRING ];
      flist : OPERATION [] RETURNS [flist : STRING ];
END.
/* Faculty Data Layer BB, Operations Implementation
   fdlbb.dpl.*/
/*
 * Generated by `stubc $Revision: 1.15 $'
 *    from `fdlbb.idl'
 *    on `Sun Nov  8 13:56:24 1992'
 */
#include "ansa.h"
#include "dis.h"
#include <string.h>
#define TORF 1
```

```
#define PROPSIZE 1024
char propbuf[PROPSIZE];
! USE fdlbb
! USE Trader
! DECLARE { fir } : fdlbb SERVER
void body(argc, argv, envp)
int argc;
char *argv[];
char *envp[];
{
     ansa_InterfaceRef fir;
!    {fir} :: fdlbb$Create(10)
     (void) system_init_properties(propbuf, PROPSIZE, argc,
argv);
!    {} <- traderRef$Export("fdlbb", "/", propbuf, fir)
}
int fdlbb_fadd( _attr, fnew, fstat )
     ansa_InterfaceAttr *_attr;
     FRECORD fnew;
     ansa_Integer *fstat;
{
     char ssn[256];
     char *newrec, *irec, *buf;
     int  rqst, found = 0, stat = 1;
     FILE *fp;
     sprintf(ssn,"%-11s", fnew.FSSN);
ropen:
```

```
if ((fp = fopen(FDATABASE,"r+")) == NULL)
{
     error('e',"Can not Open \n");
     if ((fp = fopen(FDATABASE,"a")) == NULL)
     {
          error('e',"Can Not Create\n");
          sleep(2);
          *fstat=0;
          return(1);
     }
     else
     {
          fclose(fp);
          goto ropen;
     }
}
else
{
     irec = (char *) malloc(11);
     buf = (char *) malloc(256);
     while (fgets(buf,256,fp))
     {
          buf[11] = '\0';
          sprintf(irec,"%-11s",buf);
          if (!strcmp(irec, ssn))
          {
               printf("DUPLICATE RECORD\n");
```

```
                    *fstat = 0;
                    free(buf);
                    free(irec);
                    return(1);
               } /* if */
          } /* while */
     } /* else */
     newrec = (char *) malloc(256*6);
     sprintf(newrec,"%-11s|%s|%s|%s|%s|%s\n",
     fnew.FSSN, fnew.FNAME, fnew.MNAME, fnew.LNAME,
     fnew.PHONE, fnew.LOCATION);
     if (fprintf(fp,"%s",newrec))
          *fstat = 1;
     else
     {
          printf("\0007");
          *fstat = 0;
          error('e',"Can not ADD the requested
Information");
     }
     fclose(fp);
     free(newrec);
     free(buf);
     free(irec);
     return(1);
}
int fdlbb_fdel( _attr, fssn, fstat )
```

```
ansa_InterfaceAttr *_attr;
ansa_String fssn;
ansa_Integer *fstat;

FILE *fp, *ofp;
int stat = 0, i, found = 0;
char dssn[15], *flds[256], *drec, s[256];
if ((fp=fopen(FDATABASE,"r")) == NULL)
{
     error('e',"Can not Open");
     *fstat = 0;
     return(1);
}
if ((ofp=fopen("tempfac.data","w")) == NULL)
{
     error('e',"Can not Open");
     *fstat = 0;
     return(1);
}
sprintf(dssn,"%-11s", fssn);
drec = (char *) malloc(256);
while(fgets(drec,256,fp))
{
     strcpy(s,drec);
     for(i=0;i<MAX;i++)
          if((flds[i]=strtok(drec,"|")) != NULL)
               drec = NULL;
```

```
                    else
                         break;
               drec = (char *) malloc(256);
               if (!strcmp(dssn,flds[0]))
               {
                    found = 1;
                    continue;
               }
               fprintf(ofp,"%s", s);
          } /* while() */
          if (!found)
               *fstat = 0;
          else
               *fstat = 1;
          fclose(fp);
          fclose(ofp);
          remove(FDATABASE);
          rename("tempfac.data",FDATABASE);
          return(1);
     }
     int fdlbb_fview( _attr, fssn, fstat )
          ansa_InterfaceAttr *_attr;
          ansa_String fssn;
          ansa_String *fstat;
     {
          int stat = 1, i = 0;
          char *retstr;
```

```
        char *irec, rssn[15], *flds[256];
        FILE *fp;
        sprintf(rssn,"%-11s", fssn);
        if ((fp = fopen(FDATABASE,"r")) == NULL)
        {
                strcpy(*fstat,"Faculty Database Empty, Nothing to
View\n");
                return(1);
        }
        else
        {
                retstr = (char *) malloc(356);
                while ((stat) && (fgets(retstr,256,fp)))
                {
                        irec = (char *) malloc(356);
                        strcpy(irec, retstr);
                        for(i=0;i<6;i++)
                                if((flds[i]=strtok(irec,"|")) != NULL)
                                        irec = NULL;
                        stat = strcmp(rssn, flds[0]);
                        if (stat == 0)
                                *fstat = retstr;
                } /* while */
        } /* else */
        if (stat != 0)
                *fstat = "Not Found";
        fclose(fp);
```

```
     return(1);
}
int fdlbb_flist( _attr, flist )
     ansa_InterfaceAttr *_attr;
     ansa_String *flist;
{
     char lstbuf[256];
     char *outrec = NULL;
     FILE *fp;
     if ((fp = fopen(FDATABASE,"r")) == NULL)
     {
          strcpy(*flist,"Faculty Database Is Empty, Nothing
To List\n");
          return(1);
     }
     if ((outrec = (char *) malloc(3700)) == NULL)
     {
          strcpy(*flist,"Memory Fault::FLIST\n");
          return(1);
     }
     strcpy(outrec,"");
     while(fgets(lstbuf,256,fp))
     {
          lstbuf[strlen(lstbuf)-1] = '\0';
          if (!strcmp(lstbuf,"\n"))
               break;
          strcat(lstbuf,"\n");
          strcat(outrec,lstbuf);
     }
     fclose(fp);
     if (!strcmp(outrec,""))
          *flist = "DataBase FACULTY :: EMPTY";
     else
          *flist = outrec;
     return(1);
}
/* Course Data Layer Building block */
ulbb : INTERFACE =
BEGIN
     submenu : OPERATION [opcode : INTEGER ] RETURNS [smenu
: STRING ];
     procmenu : OPERATION [] RETURNS [pmenu : STRING ];
END.

#include "ansa.h"
#include "tulbb.h"
#include "dis.h"
#define PROPSIZE 1024
char propbuf[PROPSIZE];

! USE ulbb

! USE Trader
! DECLARE { ir } : ulbb SERVER
```

```
void body(argc, argv, envp)
int argc;
char *argv[];
char *envp[];
{
     ansa_InterfaceRef ir;

!    {ir} :: ulbb$Create(10)
     (void) system_init_properties(propbuf, PROPSIZE, argc,
argv);
!    {} <- traderRef$Export("ulbb", "/", propbuf, ir)
}

int ulbb_submenu( _attr, opcode, smenu )
     ansa_InterfaceAttr *_attr;
     ansa_Integer opcode;
     ansa_String *smenu;
{

     char *rqmnu;
     char title[40];

     rqmnu = (char *) malloc(400);

     switch(opcode)
     {
            case FSTART :
                 strcpy(title,"\t\t\t\tFACULTY OPERATIONS
MENU");
                 break;
            case CSTRT :
                 strcpy(title,"\t\t\t\tCOURSE OPERATIONS
MENU");
                 break;
            case SSTART :
                 strcpy(title,"\t\t\t\tSTUDENT OPERATIONS
MENU");
                 break;
            case RSTART :
                 strcpy(title,"\t\t\t\tREGISTRATION MENU");
                 break;

       }

       if (opcode == FSTART || opcode == CSTRT || opcode ==
SSTART || opcode == RSTART)
       {

       strcpy(rqmnu,"");
       strcat(rqmnu,"\n\t\t-----------------------------------
-------------------\n");
       strcat(rqmnu, title);
```

```
     strcat(rqmnu,"\n\t\t------------------------------------
-------------------");
     strcat(rqmnu,"\n\n\t\t\t\t<A/a>  Add Information\n");
     strcat(rqmnu,"\n\t\t\t\t<D/d>  Delete Information\n");
     strcat(rqmnu,"\n\t\t\t\t<U/u>  Update Information\n");
     strcat(rqmnu,"\n\t\t\t\t<L/l>  List Information\n");
     strcat(rqmnu,"\n\t\t\t\t<V/v>  View Information\n");
     strcat(rqmnu,"\n\t\t\t\t<E/e>  Exit This Menu\n");
     *smenu = rqmnu;
     return(1);
     }
     else
     {
     strcat(rqmnu,"INVALID SELECTION\n");
     *smenu = rqmnu;
     return 1;
     }
}

int ulbb_procmenu( _attr, pmenu )
     ansa_InterfaceAttr *_attr;
     ansa_String *pmenu;
{
     char *rqmnu;
     char title[40];

     rqmnu = (char *) malloc(400);
     strcpy(rqmnu,"");
     strcpy(rqmnu,"\n\n");
     strcat(rqmnu,"\t\t-------------------------------------
-----------------\n");
     strcat(rqmnu,"\t\t\t Processing Layer Functions\n");
     strcat(rqmnu,"\t\t-------------------------------------
-----------------");
     strcat(rqmnu,"\n\n\t\t\t<1>  List of Courses Taught By
a Faculty\n");
     strcat(rqmnu,"\n\t\t\t<2>  Student/Regtr. Delte of
Student info.\n");
     strcat(rqmnu,"\n\t\t\t<E/e>  Exit This Menu\n");
     *pmenu = rqmnu;
     return(1);
}

/* Student Data Layer Building BBlock */
sdlbb : INTERFACE =
BEGIN
     SRECORD : TYPE = RECORD [
     SSSN   : STRING,
     FNAME : STRING,
     MNAME : STRING,
     LNAME : STRING,
     SPHONE : STRING,
     SBDATE : STRING,
     SLEVEL : STRING,
     SMAJOR : STRING,
     SMINOR : STRING,
     SGPA :  STRING,
     SENROLL :STRING
     ];

     sadd : OPERATION [snew : SRECORD ] RETURNS [sstat :
INTEGER ];
     sdel : OPERATION [sssn : STRING ] RETURNS  [sstat :
INTEGER ];
     sview : OPERATION [sssn : STRING] RETURNS [sstat :
STRING ];
     slist : OPERATION [] RETURNS [slist : STRING ];
END.
/*
 * Generated by `stubc $Revision: 1.15 $'
 *    from `sdlbb.idl'
 *    on `Sun Nov  8 14:28:42 1992'
 */
#include "dis.h"
#include <string.h>
#include "ansa.h"

#define PROPSIZE 1024
char propbuf[PROPSIZE];
```

```
!USE sdlbb

!USE Trader

!DECLARE {sir} : sdlbb SERVER


void body(argc, argv, envp)

int argc;

char *argv[];

char *envp[];

{

     ansa_InterfaceRef sir;


!    {sir} :: sdlbb$Create(10)

     (void) system_init_properties(propbuf, PROPSIZE, argc,

argv);

!    {} <- traderRef$Export("sdlbb", "/", propbuf, sir)


}


int sdlbb_sadd( _attr, snew, sstat )

     ansa_InterfaceAttr *_attr;

     SRECORD snew;

     ansa_Integer *sstat;

{

          int stat = 1;

          char ssn[256];

          char *newrec, *irec, *buf;

          int fd, rqst, found = 0;
```

```
            FILE *fp;


         printf("The Student SSN is %s\n",snew.SSSN);
            sprintf(ssn,"%-11s", snew.SSSN);
    sopen:
            if ((fp = fopen(SDATABASE,"r+")) == NULL)
         {
              error('e',"Cannot open \n");
              if ((fp = fopen(SDATABASE,"a")) == NULL)
              {
                   error('e',"Cannot create\n");
                   *sstat=0;
                   return(1);
                    }
              else
              {
                   fclose(fp);
                   goto sopen;
              }
          }
          else
          {
              irec = (char *) malloc(11);
              buf= (char *) malloc(256);
              while (fgets(buf,256,fp))
              {
                   buf[11] = '\0';
```

```
                sprintf(irec,"%-11s",buf);
                if(!strcmp(irec,ssn))
                {
                     printf("DUPLICATE RECORD\n");
                     *sstat = 0;
                     free(buf);
                     free(irec);
                     return(1);
                         } /* if */
                  } /* while */
             }/*else*/
      newrec= (char *) malloc(256*11);
      sprintf(newrec,"%-11s|%s|%s|%s|%s|%s|%s|%s|%s|%s|%s\n",
      snew.SSSN,snew.FNAME,snew.MNAME,snew.LNAME,snew.SPHONE,
      snew.SBDATE,snew.SLEVEL,snew.SMAJOR,snew.SMINOR,snew.SG
PA,
      snew.SENROLL);
      if (fprintf(fp,"%s",newrec))
           *sstat = 1;
      else
      {
           printf("\0007");
           *sstat = 0;
           error('e',"Can not ADD the requested
Information");
      }
      fclose(fp);
```

```
        free(newrec);
        free(buf);
        free(irec);
        return(1);
}

int sdlbb_sdel( _attr, sssn, sstat )
        ansa_InterfaceAttr *_attr;
        ansa_String sssn;
        ansa_Integer *sstat;
{
        FILE *fp, *ofp;
        int stat = 0, i, found = 0;
        char ssn[15], *flds[256], *drec, s[256];

        if ((fp=fopen(SDATABASE,"r")) == NULL)
        {
                error('e',"Can not Open");
                *sstat = 0;
                return(1);
        }
        if ((ofp=fopen("tempstd.data","w")) == NULL)
        {
                error('e',"Can not Open");
                *sstat = 0;
                return(1);
}

sprintf(ssn,"%-11s", sssn);
drec = (char *) malloc(256);
while(fgets(drec,256,fp))
{
     strcpy(s,drec);
     for(i=0;i<MAX;i++)
          if((flds[i]=strtok(drec,"|")) != NULL)
               drec = NULL;
          else
               break;
     drec = (char *) malloc(256);
     if (!(stat=strcmp(ssn,flds[0])))
     {
          found = 1;
          continue;
     }
     fprintf(ofp,"%s",s);
} /* while() */
if (!found)
     *sstat = 0;
else
     *sstat = 1;
fclose(fp);
fclose(ofp);
remove(SDATABASE);
```

```
     rename("tempstd.data",SDATABASE);
     return(1);
}


int sdlbb_sview( _attr, sssn, sstat )
     ansa_InterfaceAttr *_attr;
     ansa_String sssn;
     ansa_String *sstat;
{
     int stat = 1, i = 0;
      char *retstr;
     char *irec, rssn[15], *flds[256];
     FILE *fp;

     sprintf(rssn,"%-11s",sssn);
     if ((fp = fopen(SDATABASE,"r")) == NULL)
     {
          error('e',"Student Database Empty \n");
          return(1);
     }
     else
     {
          retstr = (char *) malloc(356);
          while ((stat) && (fgets(retstr,256,fp)))
          {
               irec = (char *) malloc(356);
               strcpy(irec, retstr);
```

```
                for(i=0;i<11;i++)
                     if((flds[i]=strtok(irec,"|")) != NULL)
                          irec = NULL;
                stat = strcmp(rssn, flds[0]);
           } /* while */
     } /* else */
       fclose(fp);
     if(!stat)
     {
             *sstat=retstr;
          return(1);
     }
     else
     {
          *sstat="Not Found, Does not Exist";
          return(1);
     }
}

int sdlbb_slist( _attr, slist )
     ansa_InterfaceAttr *_attr;
     ansa_String *slist;

{
     char lstbuf[256];
     char *outrec = NULL;
     FILE *fp;
```

```
    if ((fp = fopen(SDATABASE,"r")) == NULL)
    {
         strcpy(*slist," Student Database Empty\n");
         return(1);
    }
    if ((outrec = (char *) malloc(3700)) == NULL)
    {
         strcpy(*slist,"Memory fault:SLIST\n");
         return(1);
    }
    strcpy(outrec,""); /* this is for each new invocation
of slist_record() */
    while(fgets(lstbuf,256,fp))
    {
         lstbuf[strlen(lstbuf)-1]='\0';
         if (!strcmp(lstbuf,"\n"))
              break;
         strcat(lstbuf,"\n");
         strcat(outrec, lstbuf);
    }
    fclose(fp);
    if (!strcmp(outrec,""))
         *slist="Database STUDENT :EMPTY";
    else
         *slist=outrec;
    return(1);
```

```
  }


/* Registration Data Layer Building Block*/
rdlbb : INTERFACE =
BEGIN
     RRECORD : TYPE = RECORD[
     SSN    : STRING,
     CRS1   : STRING,
     SEC1   : STRING,
     CRS2   : STRING,
     SEC2   : STRING,
     CRS3   : STRING,
     SEC3   : STRING
     ];

     radd : OPERATION [rnew : RRECORD ] RETURNS [rstat :
INTEGER ];
     rdel : OPERATION [rssn : STRING ] RETURNS  [rstat :
INTEGER ];
     rview : OPERATION [rssn : STRING] RETURNS [rstat :
STRING ];
     rlist : OPERATION [] RETURNS [rlist : STRING ];
END.

#include "ansa.h"
#include "dis.h"
#include <string.h>
```

```
#define TORF 1

#define PROPSIZE 1024
char propbuf[PROPSIZE];

! USE rdlbb
! USE Trader
! DECLARE { rir } : rdlbb SERVER

void body(argc, argv, envp)
int argc;
char *argv[];
char *envp[];
{
     ansa_InterfaceRef rir;

!    {rir} :: rdlbb$Create(10)
     (void) system_init_properties(propbuf, PROPSIZE, argc,
argv);
!    {} <- traderRef$Export("rdlbb", "/", propbuf, rir)
}

int rdlbb_radd( _attr, rnew, rstat )
     ansa_InterfaceAttr *_attr;
     RRECORD rnew;
     ansa_Integer *rstat;
{
        char ssn[60];
        char *newrec, *irec, *buf;
        int rqst, found = 0;
        FILE *fp;

        sprintf(ssn,"%-lls", rnew.SSN);
        printf("SSN = %s\n",ssn);

    ropen:
        if ((fp = fopen(RDATABASE,"r+")) == NULL)
        {
            error('e',"Can not Open \n");
            if ((fp = fopen(RDATABASE,"a")) == NULL)
            {
                error('e',"Can Not Create\n");
                sleep(2);
                *rstat=0;
                return(1);
            }
            else
            {
                fclose(fp);
                goto ropen;
            }
        }
        else
        {
```

```
        irec = (char *) malloc(11);
        buf = (char *) malloc(256);
        while(fgets(buf,256,fp))
        {
             buf[11] = '\0';
             printf("BUF = %s\n", buf);
             sprintf(irec,"%-11s",buf);
             printf("IREC = %s\n", irec);
             if (!strcmp(irec,ssn))
             {
                  printf("DUPLICATE RECORD\n");
                  *rstat = 0;
                  free(buf);
                  free(irec);
                  return(1);
             }
        } /*while*/
     } /*else*/
   newrec = (char *) malloc(256*7);
   sprintf(newrec,"%s|%s|%s|%s|%s|%s|%s\n",
   rnew.SSN, rnew.CRS1, rnew.SEC1, rnew.CRS2, rnew.SEC2,
   rnew.CRS3, rnew.SEC3);

   if (fprintf(fp,"%s",newrec))
        *rstat = 1;
   else
   {
          printf("\0007");
          *rstat = 0;
          error('e',"Can not Add the requested
Information");
     }
     fclose(fp);

     free(newrec);
     free(buf);
     free(irec);
     return(1);
}

int rdlbb_rdel( _attr, rssn, rstat )
     ansa_InterfaceAttr *_attr;
     ansa_String rssn;
     ansa_Integer *rstat;
{
     FILE *fp, *ofp;
     int stat = 0, i, found = 0;
     char ssn[15], *flds[256], *drec, s[256];

     if ((fp=fopen(RDATABASE,"r")) == NULL)
     {
          error('e',"Can not Open");
          return(1);
     }
```

```
if ((ofp=fopen("temprgstr.data","w")) == NULL)
{
      error('e',"Can not Open");
      return(1);
}

sprintf(ssn,"%-11s", rssn);
drec = (char *) malloc(256);
while(fgets(drec,256,fp))
{
      strcpy(s,drec);
      for(i=0;i<MAX;i++)
            if((flds[i]=strtok(drec,"|")) != NULL)
                  drec = NULL;
            else
                  break;
      drec = (char *) malloc(256);
      if (!(stat=strcmp(ssn,flds[0])))
      {
            found = 1;
            continue;
      }
      fprintf(ofp,"%s",s);
} /* while() */
if (!found)
      *rstat = 0;
else
```

```
            *rstat = 1;
        fclose(fp);
        fclose(ofp);
        remove(RDATABASE);
        rename("temprgstr.data",RDATABASE);

        return(1);
}

int rdlbb_rview( _attr, rssn, rstat )
        ansa_InterfaceAttr *_attr;
        ansa_String rssn;
        ansa_String *rstat;
{
        int stat = 1, i = 0;
        char *retstr;
        char *irec, regssn[11], *flds[256];
        FILE *fp;

        sprintf(regssn,"%-11s", rssn);
        if ((fp = fopen(RDATABASE,"r")) == NULL)
        {
            strcpy(*rstat,"Data Base :: REGISTER :: EMPTY\n");
            return(1);
        }
        else
        {
          retstr = (char *) malloc(356);
          while ((stat) && (fgets(retstr,256,fp)))
          {
               irec = (char *) malloc(356);
               strcpy(irec, retstr);
               for(i=0;i<MAX;i++)
                    if((flds[i]=strtok(irec,"|")) != NULL)
                         irec = NULL;
               stat = strcmp(regssn, flds[0]);
               printf("retstr = %s\n", retstr);
               if (stat == 0)
                    *rstat = retstr;
          } /* while */
     } /* else */
     if(stat != 0)
          *rstat = "Not Found, dos Not
Exist::REGISTER::DATABASE";
        fclose(fp);
     return(1);
}


int rdlbb_rlist( _attr, rlist )
     ansa_InterfaceAttr *_attr;
     ansa_String *rlist;
{
     char lstbuf[256];
     char *outrec = NULL;
     FILE *fp;

     if ((fp = fopen(RDATABASE,"r")) == NULL)
     {
          strcpy(*rlist,"Register Database Is Empty, Nothing
To List\n");
          return(1);
     }

     if ((outrec = (char *) malloc(3700)) == NULL)
     {
          strcpy(*rlist,"Memory Fault::RLIST\n");
          return(1);
     }
     strcpy(outrec,"");
     while(fgets(lstbuf,256,fp))
     {
          lstbuf[strlen(lstbuf)-1] = '\0';
          if (!strcmp(lstbuf,"\n"))
               break;
          strcat(lstbuf,"\n");
          strcat(outrec,lstbuf);
     }
     fclose(fp);
     if (!strcmp(outrec,""))
          *rlist = "DATABASE REGISTER :: EMPTY";
```

```
    else
         *rlist = outrec;
    return(1);
}

/* Processing Layer BB */
proclyr:INTERFACE =
NEEDS fdlbb;
NEEDS cdlbb;
NEEDS sdlbb;
NEEDS rdlbb;
BEGIN
      stdelete: OPERATION [ opcode : STRING] RETURNS [
result: INTEGER];

      listfac : OPERATION [ opcode : STRING] RETURNS [
result :STRING];

END.
/*
 * Generated by `stubc $Revision: 1.15 $'
 *    from `proclyr.idl'
 *    on `Mon Nov 23 12:43:57 1992'
 */

#include "ansa.h"
#include "dis.h"
```

```
#define PROPSIZE 1024
char propbuf[PROPSIZE];

! USE fdlbb
! DECLARE {facir} : fdlbb CLIENT

! USE sdlbb
! DECLARE {stir} : sdlbb CLIENT

! USE rdlbb
! DECLARE {regir} : rdlbb CLIENT

! USE cdlbb
! DECLARE {coir} : cdlbb CLIENT

! USE proclyr
! DECLARE {pir} : proclyr SERVER

! USE Trader

     ansa_InterfaceRef stir;
     ansa_InterfaceRef regir;
     ansa_InterfaceRef coir;
     ansa_InterfaceRef facir;

void body(argc, argv, envp)
```

```
int argc;
char *argv[];
char *envp[];
{
        ansa_InterfaceRef pir;

!       {pir} :: proclyr$Create(10)
        (void) system_init_properties(propbuf, PROPSIZE, argc,
argv);
!       {} <- traderRef$Export("proclyr", "/", propbuf, pir)
}




int proclyr_stdelete( _attr, opcode, result )
        ansa_InterfaceAttr *_attr;
        ansa_String opcode;
        ansa_Integer *result;
{
        static int stat = 0;
        char *tssn;
        long int sdelete = 0, rdelete = 0;


        tssn = (char *) malloc(20);
        strcpy(tssn, opcode);
!    {stir} <- traderRef$Import("sdlbb","/","")

!       {sdelete} <- stir$sdel(tssn)

     if (!sdelete)
     {
          *result = sdelete;
!         stir$Discard
          return(1);
     }

!    {regir} <- traderRef$Import("rdlbb","/","")

!       {rdelete} <- regir$rdel(tssn)

     if (!rdelete)
     {
          *result = rdelete;
!         regir$Discard
          return(1);
     }
!         stir$Discard
!    regir$Discard
     return(1);
}
```

```
int proclyr_listfac( _attr, opcode, result )
      ansa_InterfaceAttr *_attr;
      ansa_String opcode;
      ansa_String *result;
{
      int stat = 1;
      int i = 0, k = 0, j = 0;
      char *outrec;
      char *cstr, *cistr[256];
      char *fssn, *list, *crec[656], *flds[256], *crecptr;
      char *view;

      if((outrec = (char *) malloc(5000)) == NULL)
      {
            *result = "PROC::LISTFAC::Cannot Allocate output
Buffer";
            return(1);
      }

      list = (char *) malloc(5000);
      view = (char *) malloc(5000);
      fssn = (char *) malloc(20);

      strcpy(outrec,"");
      *result = "";

      sprintf(fssn,"%-11s",opcode);
```

```
!     {facir} <- traderRef$Import("fdlbb","/","")


!        {view} <- facir$fview(fssn)
      if(!(stat=strcmp(view,"Not Found")))
      {
           *result = " FACULTY DOES NOT EXIST : INVALID SSN";
           return(1);
      }
      strcpy(outrec,"Faculty Detail is : \n");
      strcat(outrec, view);

      cstr = (char *) malloc(800);
!     {coir} <- traderRef$Import("cdlbb","/","")
!     {cstr} <- coir$clist()
      if(!(stat=strcmp(cstr,"")))
      {
           strcat(outrec,"\nNo Courses Taken By the Requested
Faculty\n");
           *result = outrec;
           return(1);
      }

      strcat(outrec,"\nThe Courses Taken By The Faculty
Are");
      strcpy(list, cstr);
      for (i=0;i<MAX;i++)
      {
        if((crec[i]=strtok(list,"\n")) != NULL)
             list = NULL;
        else
             break;
    }

    for(k=i-1;k>=0;k--)
    {
        crecptr = (char *) malloc(300);
        strcpy(crecptr,crec[k]);
        for(j=0;j<MAX;j++)
             if((flds[j]=strtok(crecptr,"|")) != NULL)
                  crecptr = NULL;
             else
                  break;
        if(!(strcmp(flds[5], fssn)))
        {
             strcat(outrec,"\n");
             strcat(outrec,crec[k]);
        }
    }
    *result = outrec;
!   coir$Discard
!   facir$Discard
    return(1);
}
```

```
/* User Layer Building Block */
ulbb : INTERFACE =
BEGIN
     submenu : OPERATION [opcode : INTEGER ] RETURNS [smenu
: STRING ];
     procmenu : OPERATION [] RETURNS [pmenu : STRING ];
END.

/*
 * Generated by `stubc $Revision: 1.15 $'
 *   from `ulbb.idl'
 *   on `Sat Nov  7 14:13:38 1992'
 */

#include "ansa.h"
#include "tulbb.h"
#include "dis.h"

#define PROPSIZE 1024
char propbuf[PROPSIZE];

! USE ulbb

! USE Trader
! DECLARE { ir } : ulbb SERVER

void body(argc, argv, envp)
```

```
int argc;
char *argv[];
char *envp[];
{
     ansa_InterfaceRef ir;

!    {ir} :: ulbb$Create(10)
     (void) system_init_properties(propbuf, PROPSIZE, argc,
argv);
!    {} <- traderRef$Export("ulbb", "/", propbuf, ir)
}

int ulbb_submenu( _attr, opcode, smenu )
     ansa_InterfaceAttr *_attr;
     ansa_Integer opcode;
     ansa_String *smenu;
{

     char *rqmnu;
     char title[40];

     rqmnu = (char *) malloc(400);

     switch(opcode)
     {
          case FSTART :
```

```
                        strcpy(title,"\t\t\t\tFACULTY OPERATIONS
MENU");
                        break;
                    case CSTRT :
                        strcpy(title,"\t\t\t\tCOURSE OPERATIONS
MENU");
                        break;
                    case SSTART :
                        strcpy(title,"\t\t\t\tSTUDENT OPERATIONS
MENU");
                        break;
                    case RSTART :
                        strcpy(title,"\t\t\t\tREGISTRATION MENU");
                        break;

                }

                if (opcode == FSTART || opcode == CSTRT || opcode ==
SSTART || opcode == RSTART)
                {

                strcpy(rqmnu,"");
                strcat(rqmnu,"\n\t\t-------------------------------------
-------------------\n");
                strcat(rqmnu, title);
                strcat(rqmnu,"\n\t\t-------------------------------------
-------------------");
```

```
          strcat(rqmnu,"\n\n\t\t\t\t<A/a>  Add Information\n");
          strcat(rqmnu,"\n\t\t\t\t<D/d>  Delete Information\n");
          strcat(rqmnu,"\n\t\t\t\t<U/u>  Update Information\n");
          strcat(rqmnu,"\n\t\t\t\t<L/l>  List Information\n");
          strcat(rqmnu,"\n\t\t\t\t<V/v>  View Information\n");
          strcat(rqmnu,"\n\t\t\t\t<E/e>  Exit This Menu\n");
          *smenu = rqmnu;
          return(1);
          }
          else
          {
          strcat(rqmnu,"INVALID SELECTION\n");
          *smenu = rqmnu;
          return 1;
          }
}

int ulbb_procmenu( _attr, pmenu )
          ansa_InterfaceAttr *_attr;
          ansa_String *pmenu;
{
          char *rqmnu;
          char title[40];

          rqmnu = (char *) malloc(400);

          strcpy(rqmnu,"");
```

```
     strcpy(rqmnu,"\n\n");
     strcat(rqmnu,"\t\t--------------------------------------
-----------------\n");
     strcat(rqmnu,"\t\t\t Processing Layer Functions\n");
     strcat(rqmnu,"\t\t--------------------------------------
-----------------");
     strcat(rqmnu,"\n\n\t\t\t<1>  List of Courses Taught By
a Faculty\n");
     strcat(rqmnu,"\n\t\t\t<2>  Student/Regtr. Delte of
Student info.\n");
     strcat(rqmnu,"\n\t\t\t<E/e>  Exit This Menu\n");
     *pmenu = rqmnu;
     return(1);
}
```

Thus, this is the IDL and DPL files for the building blocks that make up the EIS. Now finally, we list the Imakefile that is used to define the dependencies among all the files.

REFERENCES

1. Rossak, W. "Integration Architectures, A Concept and a Tool to Support Integrated Systems Development." (1992): 6-10.

2. Rossak, W., and P. Ng. "System Development with Integration Architectures." Proceedings of The Second International Conference on Systems Integration. (1992): 1-8.

3. Wilhelm, R., and T. Zemel. "A Two-Level Process Model for Integrated System Development." (1992): 1-20

4. Zemel, T. "A Mega-System Development Framework." A Ph.D. Thesis, Department of Computer and information Sciences, New Jersey Institute of Techonology, Systems Integration Laboratory,in work. (1993).

5. Mills, J. "An OSCA Architecture Characeterization of Network Functionality and Data." (1991): 1-19.

6. Technical Reference. The Bellcore OSCA^{TM} Architecture. Issue 1, (1992): 1-80.

7. Masand, B. "Development of Prototype Distributed Information System.". Master's Thesis, Department of Computer and Information Sciences, New Jersey Institute of Technology, Systems Integration. (1992): 10-90.

8. Bloomer, J. "Ticket to Ride, Remote Procedure calls in a Network Environment." An article in SunWorld. (1991): 39-55.

9. ANSAware Application Programmer's Manual.