A STUDY IN SYSTEMS INTEGRATION ARCHITECTURE

By

SASHIDHAR M. PRASAD

Adviser

Prof. Wilhelm Rossak
Assistant Professor

This thesis is submitted to the Faculty of the Graduate School of the New Jersey Institute Of Technology in partial fulfillment of the requirement for the Degree of Master Of Science in Computer and Information Science

December 1991

APPROVAL SHEET

Title of Thesis: A Study in System Integration Architecture

Name of Candidate: SASHIDHAR M. PRASAD

Master of Science in Computer
and Information Science, 1991.

Thesis and Abstract Approved :

Dr. Wilhelm Rossak   Date
Assistant Professor
Computer and Information Science Department

Dr. Lonnie R. Welch   Date
Assistant Professor
Computer and Information Science Department

VITA

Name:	SASHIDHAR M. PRASAD

Secondary Education:	St. George's Grammar School - 1977
Colleges Attended:	

New Jersey Institute of Technology:	09/89 - 12/91	M.S (CIS)	10/91

Major:	Computer and Information Science
Positions Held:	Finance Executive, REFRATEK INDUSTRIES (P) Ltd.

ABSTRACT

Title of Thesis: A Study In Systems Integration Architecture

Name of Candidate: Prasad M. Sashidhar

Thesis Advisor: Dr Wilhelm Rossak
Assistant Professor
Computer Information Science Department.

This Thesis studies the two architectures OSCA and ANSA which support the ODPSE principle in the first two parts. In the third part the framework for integrating these two architectures is described. The idea of integration architectures in relation to open architectures is studied using the enabling technologies.

ACKNOWLEDGEMENTS

The author wishes to thank Dr. Wilhelm Rossak for his valuable suggestions and technical assistance during the preparation of this report.Also the author wishes to thank Mr. Michael Bohomelec who proof read this report and gave his valuable suggestions.

Contents

List of Figures

Chapter 1

OSCA^{TM} Architecture

1.1 Introduction and Overview

1.1.1 What is OSCA^{TM} Architecture ?

The OSCA^{TM} Architecture has been designed by Bellcore to enable and enhance the interoperability among software systems [OSCA89]. OSCA^{TM} architecture is a strategic architecture intended to be used by Bellcore Client Companies (BCC's) to provide software interoperability. This means software products of various suppliers will be able to operate with the existing software products used by the Bellcore Client Companies. It also allows the Bellcore Client Company Operating Systems that are distributed over a wide range of computing environments such as a wide range of database management systems, work stations and application architectures to be compatible with a wide variety of communication architectures and data structure which should be able to interoperate. The OSCA^{TM} architecture is a system design framework which is intended to give the Bellcore Client Companies (BCC) the flexibility to combine various software products in various ways that satisfy their business needs. This is known as **"Interoperability"**. Interoperability is the ability of OSCA^{TM} to interconnect various software products of different manufacturers which consists of large number of programs, transactions and databases. It also providea access to the corporate data by any authorized user and to maintain this access and interconnection even if there are changes in the suppliers and the vintages.

One of the fundamental ideas of this architecture is the notion of **"SEPARATION OF CONCERN"**. This means the application functionality, be grouped into three layers or domains, namely, Data Layer, Processing Layer and User Layer. The software which implements the functionality in these layers is partitioned into "**building blocks**", and these have to work under certain fixed guidelines or principles. These building blocks are interconnected by means of interfaces known as "**CONTRACTS**" which also must adhere to certain rules and guidelines. A layer is the union of functionality defined as either Corporate Data functionality, Processing Functionality or User Functionality.

When we refer to "**Separation of Concerns**" in the OSCA^{TM} layers we mean that the application functions are allocated to building blocks within a layer and that no building block contains application functions for more than one layer. It should also be noted interfaces amongst the functions in one building block are well defined and contain well formed interfaces, so that the functions of one layer are decoupled from the functions of another layer.

The Corporate Data layer provides the functionality to support the semantic integrity of the Corporate Data. The User layer provides the functionality to support human users. The Processing layer provides the functionality for business processes. Each layer is made up of one or more well defined functional units called

Building Blocks. The building blocks are interconnected with each other by means of interfaces called "**CONTRACTS**". These building blocks are the software product deliverables. Any building block can communicate with any other building block that provides a function that this building block requires. All these building blocks are tied together by means of a "**Communication Software Fabric**".

To promote interoperability, OSCA^{TM} architecture provides the goals that are given below :-

- Separation of application processing in a software product from the corporate data processing used by a software product.

  To be interoperable, the architecture must have its corporate data resources defined. This must be made available to, and be shared by end users, and authorized software products, thus enabling data to be treated as a Corporate resource. Other business functions are processed separately and semantic integrity for the data is provided. Access is allowed to authorized end users in a uniform manner, irrespective of the type of the database or the communication network that is in use.

- Regulation of Redundant Corporate Data.

  Quite often software products duplicate common corporate data in their own private databases a number of times. This data becomes redundant as a result of this multiple duplication. This multiple duplication or redundancy is eliminated in OSCA^{TM} architecture by the software products which view the corporate data as a shared, corporate resource and access is got through uniform standard methods. OSCA^{TM} sees to it that the redundant copies of the data be managed such that the consistency and integrity are maintained.

- Separation of the application processing in the software products from device handling facilities.

  Under OSCA^{TM} architecture, the software products are directly not responsible for the end user devices, nor do they know the characteristics of the end user devices when they are communicating with an end user device or any other software products.

- Ability for the software product selection to take place.

  To meet its local needs, OSCA^{TM} allows the Bellcore Client Companies (BCC's) to configure a software product.

- Provision of a framework for interconnecting software products from device handling responsibilities.

  All participants of the architecture are required to develop their software products in the form of building blocks. This should conform to the principles and guidelines as set forth by OSCA^{TM} architecture.

- National and International standards are used to ease software product intercommunication and connection.

  The OSCA^{TM} architecture allows the use of national and international standards to open the interfaces between software products. This would help a large number of vendors to be able to participate in this architecture provided the software products they develop would be consistent with these standards.

1.1.2 Building Blocks

The OSCA^{TM} architecture utilizes a building block approach to the software product development, in which, each layer of the architecture is composed of many building blocks that are working together. Each building block is a set of computer programs, data schemas and other related software which possesses well defined functionality and interfaces. A description of a building block consists of inputs, outputs, syntax, semantics and pragmatics which unambiguously define the functions they provide to other building blocks. This feature allows building blocks to be provided by multiple suppliers. A building block need not have the knowledge of the internal structure of other building blocks and similarly OSCA^{TM} is not concerned with the internal structure of the building blocks. OSCA^{TM} specifies the guidelines that the interfaces of a building block should meet, to interact with other building blocks within a BCC configuration.

Interoperability is at the building block level. For a building block to be interoperable, all of its inter-building block interfaces and its functions must be documented and fully supported. One implication of an interoperable architecture is that a building block could be substitutable. If there are interfaces to a building block that are not documented, then they are not substitutable and thus do not conform to OSCA^{TM}. Other building blocks could use the building blocks through their documented interfaces; but if the building blocks were substituted, the undocumented interfaces could become unsupported. This becomes a violation of the principle of interoperability.

A building block utilizes hardware and software services from various sources, like services provided by the operating systems, the communication software fabric and the database management systems for it, to be able to perform its work. Since the definition of the relationship between the services and the building blocks is an intra building block concern, such definition is outside the scope of OSCA^{TM}.

The building blocks can co-operate in a number of ways. For example, a User Layer Building block could communicate with the Process Layer Building Block, by fetching and manipulating data from a Data Layer building block for the end user. Or a User layer building block can inquire directly into a Data Layer building block or a Process Layer building block can access more than one Data Layer Building Block or more than one instance of a Data Layer building Block. Whereas a user layer building block could access many Processing Layer Building Blocks, which in turn, each can have access to the data layer building blocks.

A User Layer Building Block can access many Process Layer Building Blocks or instances of a Process Layer Building Block which in turn access a single Data Layer Building Block. Building Blocks of any layer could communicate with building blocks of the same layer. Shown on the next page are a few figures showing how building blocks could cooperate with each other.

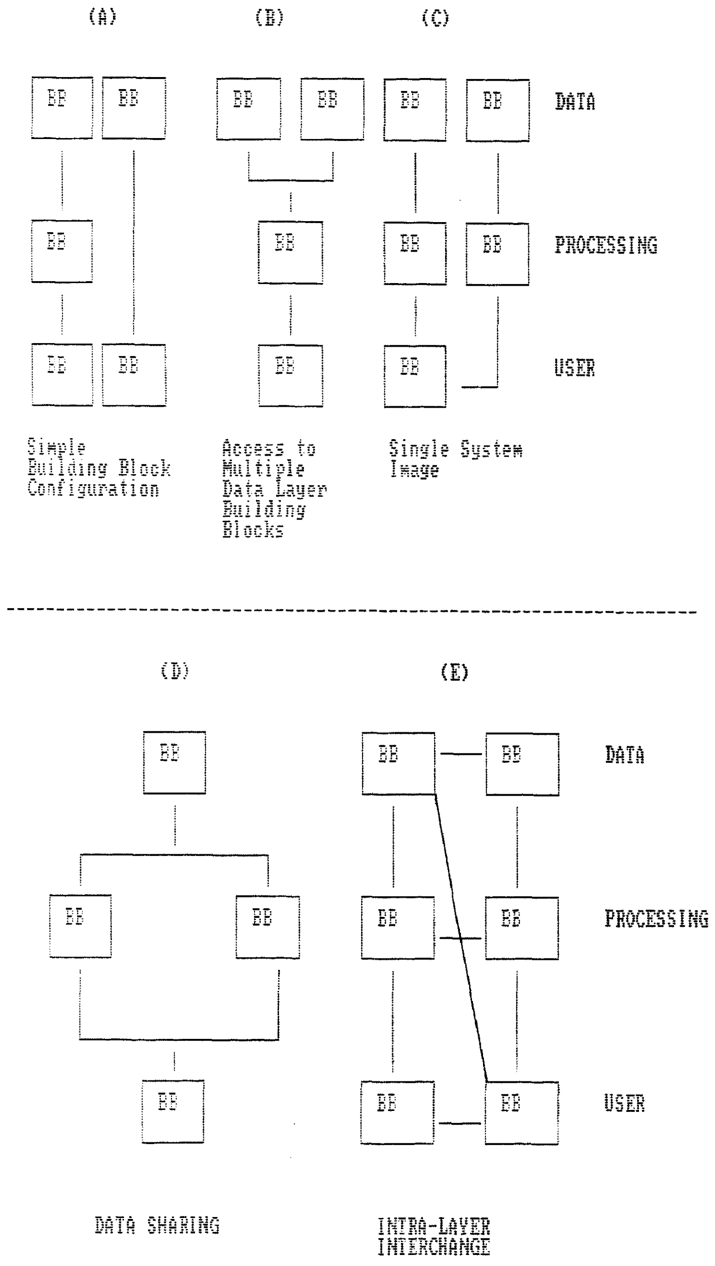

Figure 1.1: **Building Blocks Configuration**

1.1.3 Contracts

Normally large software and its components interact with one another by means of " Interfaces "also known as " Protocols" in the communications world. In OSCA^{TM} these interfaces are known as "**Contracts**".

A **contract** is a well defined set of functionality that is available to the software in all other building blocks and these should adhere to contract principles. A building block could support one or more contracts and this set of all contracts that a building block supports is termed as the "**Contract Set**" of that particular building block.

In general a contract invoker is not concerned where the contract is installed or with which other contracts it is grouped to. There are no defined set of rules for deciding which contract or contracts, a particular building block supports. A building block could support a particular contract based on the rules given below:

1. High software cohesion and low coupling between contract sets;
2. Ease of release independence. Contracts which are expected to have interdependencies with each other should be grouped together.
3. Related business functions that the contracts provide.
4. The architecture of the data which the building block acts upon, accesses, or stewards.
5. Implementation issues such as performance, proper synchronization scope, availability requirements etc.

There are two aspects for a building block,

1. the "**released**" unit of the software from the developer and

2. the "**installed**" unit of the software.

It is generally possible for a single "**released**" unit of the software to be "**installed**" at several different locations. To distinguish the "released" unit of the software from the "installed" units of software is known as **building block instance.**

1.1.4 Invocation Of Contracts

Building blocks which interact with one another may for some unforeseen condition may not do so due to communication loss, software/hardware problem etc. When such a loss occurs, the building block software should be capable of handling the unavailability in a way that is consistent and meaningful to the application. The non availability of a contract should not cause a building block to terminate abnormally or suspend other work which is being processed.

The contract principles of a well defined interface has two aspects. One set of aspects are the functionality and the characteristics that the developer of the contract has under his control and the other set of aspects are the characteristics that are imparted due to particular building block deployment instance.

1.1.5 Data

Corporate and Local Data

Corporate Data is defined by the Bellcore Client Company (BCC) as a means of data architecture and the modelling efforts. This is required for the performance of the BCC business processes and it contains all the information about the Company. Corporate data is stewarded by the Data layer Building Blocks. These data are not owned by any one organization or software components. Since these are a Corporate resource, they can only be stewarded and not owned by any single Data Layer Building Block.

In addition to the Corporate Data, OSCA^{TM} allows for local data. Local data is owned and not stewarded, this may occur in any layer of the architecture including the Corporate Data Layer. There are two types of local data:

1. **Copies of Corporate Data or redundant Corporate Data:** If there are a multiple instances of a piece of corporate data, only one instance is Corporate Data. This Corporate Data is stewarded by only one data layer building block and is always assumed to be correct. All other instances are Local Data.

2. Working data is temporary, transient and which does not have value outside the present using building block. It is likely that, once a data modelling effort is done, a minimal amount of this type of local data will remain.

1.2 Principles Of Building Blocks

Building blocks must adhere to the principles given below:

1. release independence;
2. physical database independence;
3. no accessibility assumptions between building blocks;
4. logical building block addressing;
5. execution in only one recoverable environment;
6. interactions among building blocks are defined by contracts;
7. secure environment;

The above given principles apply to building blocks. They do not apply to programs or transactions within a single building block, although the architecture does not preclude this. For example, although two building blocks must be release independent, the programs within a single building block could and probably be release dependent. The above mentioned principles are explained below:

1. **Release Independence**

   To maximize the deployment flexibility of the building blocks, each instance of a building block must be able to be installed and upgraded independently of other instances of itself and other building blocks. A building block must operate with a negotiated or predefined number of prior versions of other instances of itself and other building blocks. New building blocks must be compatible with other building blocks. To obtain new functionality, it may be required that new versions of several building blocks be installed over a period of time before the new functionality is available. Such installation is not required to take place simultaneously.

Release Independence implies that:

(a) If a software product consists of multiple building blocks, a new version of it may be installed piece meal. It is up to the suppliers to furnish the documentation to the customer, about the details which the configuration will deliver to the required functionality.

(b) An agreed set of rules that are consistently release dependent across all building blocks is needed.

2. **Physical Database Independence**

In order to ensure the Bellcore Client Companies (BCC's) deployment flexibility, two or more building blocks or instances of building blocks can use the services of the same database management system. This means that their data could be housed in the same database management system, only if release independence could be maintained amongst them. One and only one instance of a data layer building block which is the steward of any given instance of Corporate Data, uses the native access to that data. All other building blocks that access and share the data, do so through contracts supported by the stewarding building block and not via the native access. A building block could have local data which it maintains and no other building block can access it.

The purpose of this principle are:

(a) any physical reorganization of one building block's data will not affect the operation of other building blocks which are accessing and using that data or the Database Management System;

(b) any changes made to a database in the Database Management System for one building block will not affect other building blocks, since the stewarding block is responsible for buffering other building blocks from such changes;

(c) only the stewarding building block can run on the same hardware as and when it is required, to use the services of the databases and the vendor Database Management System.

3. **No Accessibility Assumptions Between Building Blocks**

One instance of a building block communication with another building block instance should be prepared to deal with the unavailability of the receiving building block instance. The sending building block must be able to handle the absence of another instance of itself or another building block in a manner that preserves its own availability. A few major implications of this principles are:

(a) Building Blocks communicate in a manner such that the delayed or non-existent response to one request will not impact the processing of other requests.

(b) If the interface between two building blocks is interactive when a session is established, a mechanism is required to detect the failure of either of the partners during the interactive session and necessary action is required.

(c) Some of the building blocks could be required by other building blocks. Which one is required depends on the functionality and customer need. However, no accessibility assumption applies, because any instance of a building block or the communication links to it may fail and the mandatory building blocks cannot be assumed to be available. Thus the building blocks, must be capable of handling the non-availability of the required building blocks in a way that is appropriate to the application.

(d) A building block must have the functionality to manage the circumstances where another building block is unavailable, including the procedures to be followed, if a building block is inaccessible or if a building block itself has become accessible after an outage, including the resynchronization of the data when appropriate.

4. **Logical Building Block Addressing**

One instance of a building block must not assume that another building block resides at a specific network location and such assumption limits the BCC's ability to manage its software products deployment and support. Building blocks should identify other building blocks by their logical addresses that are independent of the network location. The communication software fabric performs the logical to network address mapping.

5. **Execution In Only One Recoverable Environment**

A recoverable environment is one where a physical machine is running on one operating environment, or multiple machines. These are made to appear as if they are one environment from the view of the operating system, messages, databases and transaction recovery, and this is done with the use of vendor hardware and software, such as distributed database management system or a database machine.

No instances of any building block may be required to be installed on the same recoverable environment, as any other building block or any other instance of itself. Moreover, it cannot be restricted from being installed on the same recoverable environment as another building block or instance of itself. This means that a building block is installed in a recoverable environment independent of where the other building blocks are installed, in that working environment.

6. **Interactions Among Building Blocks Defined By Contracts**

Following principles must be observed to attain interoperability, which are discussed below:

(a) building blocks must not have any contracts tailored specifically for use by a particular interfacing building block, instead of being defined for a more general use;

(b) the syntax encoding must be widely used and have general industry acceptance, utilizing appropriate available national and international standards, so that no building block specifies its own private syntax encoding;

(c) relevant communication software fabric services will adhere to a commonly agreed upon set of standards;

(d) the agreed upon set of standards should be targeted towards the industry and emerging national and international standards.

7. **Secure Environment**

A building block must provide a secure environment, provide for the recognition of authorized users and audit relevant security events as per the guidelines discussed below:

(a) There must be no entry points into a building block, other than those defined in its "**contract specification**"or in case of a User Layer Building Block, those provided for authorized secure user access.

(b) Resources which are returned to the recoverable environment must not contain any sensitive information.

(c) The identity of the invoking user must be maintained and passed through the building blocks.

(d) Depending on their security requirements, building blocks may need to re-authorize the identity of the invoking users or the building blocks.

(e) All interfaces from one building block to another must be authorized by control mechanisms.

(f) Access to all local data and functions within a building block must be limited to authorized users. This authorization may be provided by the underlying recoverable environment.

(g) Depending on the security requirements, fine grained access controls that restrict users access to local or corporate data based on the data content of a specific field, attribute, tuple or record of any database may be required of any building block.

(h) Also contract dependent access controls that restrict processing based on the requesting user or building block identity, may be required by any building block.

(i) All building blocks provide audit information at a level, co-incident with security requirements.

1.3 Contracts

A Contract is a commitment by a building block to provide a well defined set of functionality to all other building blocks, in such a way that it adheres to the contract principles. A building block may support, one or more Contracts. The set of all contracts that a building block supports is termed the **Contract Set** of that building block. Interactions between OSCA^{TM} building blocks are accomplished by interfaces which adhere to contract principles that are discussed below in detail:

1. **Use Of Standards**: The Contracts should use applicable national, international and industry standards.

2. **Restricted Set of Syntax Encoding**: Contracts shall use a commonly agreed upon and restricted set of syntax encoding and supporting communications software fabric services. Common agreement should be obtained through a standard operating environment process which ensures that appropriate syntaxes and standards are selected.

3. **Isolation from Building Block Internals:** Contracts should be able to isolate the invoker from the internal building block details, like the type of processor or the type of database used, etc. If the underlying database or the processor that a building block uses should change, such change should remain transparent to the invokers of the contracts that the building block provides.

4. **Release Independence:** Changes to a contract shall occur only with adequate notice, and when a co-ordination process, is in place to affect that change.

Contracts shall limit the impact for the release changes that will have upon the invoker of the contract, by making these changes in both upward and downward compatible fashion. All contracts shall use the same single mechanism or set of rules that make it possible to release different versions of the building block independently of the building blocks that invoke the contracts.

5. **Equality Of Invocation:** The use of Contracts by building blocks may not have assumptions that would preclude or make it unreasonably difficult. Except for security reasons, a contract when invoked by building block "A" shall function exactly in the same way as a contract that is invoked by building block "B".

6. **Well Defined Interfaces:** A contract should be defined by a contract specification, which is a textual documentation that contains the way in which the functionality provided by the building block is invoked. Contracts should be well defined in terms of interface, response time and availability.

7. **Logical Addressing:** Contracts shall be identified and invoked in a way that is independent of the physical location of a building block that provides the contract. Each contract that a building block provides should be uniquely identifiable and accessible with respect to other contracts.

8. **No Contract Accessibility Assumption:** Building blocks that interact with one another, might be unable to do so, for various reasons, like communication loss, security etc. When such a loss of interaction occurs, the building block software should have the capability to handle this non-availability in a way that is consistent and meaningful to the application. The building block software

should be specifically capable of handling,

   (a) to determine whether or not a contract was successfully invoked;

   (b) or losing contact with the building block in the "middle" of the contract when the contracts are spanning several interactions.

9. **Recognition Of Authorized Users**: The implementation of contract shall perform actions only on behalf of authorized users.

10. **Maintain Identity of the Invoking User**: The invoking user's identity must be maintained and passed through, to any other building block with which the contract has to be established.

11. **Minimum Trust Of Invoking User**: A contract specification must describe the rationale for "trusting" the identity of the building block or the user as part of its security policy.

12. **Security Audits**: Contracts must incorporate enough information to allow a security audit.

Apart from the separation of User, Processing and Data functionality and building block principles, there are no specific architectural rules for grouping of contracts into the building blocks. Also there are no architectural restrictions on the type of communication service or information exchange or invocation paradigm that must be used, the number or type of exchanges, the direction or flow of exchanges or the time within which a contract must be performed, provided the layering, building block principles and contract principles are not violated. It is possible that individual building blocks or contracts could impose restrictions, but the architecture does not.

1.4 Layers

1.4.1 Corporate Data Layer Building Blocks

The fundamental principle of OSCA^{TM} is the separation of Corporate Data from processing functions and this separation, makes it easier to provide:

- Open access to corporate data by an authorized system or an end user;
- Access to a wide variety of database management systems;
- Uniform interfaces between the corporate data layer and other layers.

The separation of corporate data from the processing functions provides a framework whereby future technological advances such as semantic and knowledge based database management systems (DBMS) could be utilized without major software product rewrites, beyond the affected Data Layer Building Block (DLBB).

The Corporate Data is responsible for maintaining and providing access to all corporate data, but the data layer is just not merely a data access layer. The data layer contains much more functionality than just data access to corporate data. It provides sufficient and necessary functionality to provide users with update operations that will preserve the semantic integrity of the corporate data, including appropriate security measures, to ensure that, only authorized users are allowed to execute relevant contracts.

Data Layer building blocks (DLBB) are building blocks in the corporate data layer that steward the corporate data. Stewarding means, that the DLBB takes the responsibility for the corporate data and for all the users of this data. A DLBB stewards one or more pieces of corporate data and a piece of corporate data is always stewarded by one and only one instance of a DLBB. A piece of corporate data is an instance of a data attribute, relationship or information that describe the grouping of pieces of data at any one point in time.

The OSCA^{TM} architecture does not:

- Specify the Database Management System to be used;
- Or specify the data storage method like relation, heirarchical etc.,
- Or specify the partitioning of corporate data among Data Layer Building Block.

A Data Layer Building Block stewards the corporate data for all authorized users, and thus in OSCA^{TM} architecture, the Corporate data can be accessed by other building blocks and end users, through the contracts of the stewarding data layer building block and not directly through the native access.

The Corporate Data is determined by separate data architecture efforts. When there are inconsistencies between corporate data and other data, corporate data is always assumed to be correct. Corporate Data and consequently a data layer building block, do not belong to a project or an organization, but to the company as a whole. If two or more data layer building blocks use the service of the same database management system, a change in the database management system used for one building block may not require a corresponding change in other building blocks.

A Data Layer Building Block adheres to the principles that are discussed below in detail:

1. **Separation** contains only the functionality enumerated in these principles and guarantees a support of its contracts over the internal changes. This functionality eliminates the processing and user functionality and provides query, integrity and redundancy management functions for corporate data. A data layer building block provides access to the stewarded data through the contracts which are kept invariant across the internal structural and database changes. This ensures that the changes in the Data Layer Building Block are not a result of the new functionality and this will not be visible to other building blocks.

2. **Onlyness:** A DLBB provides means such that the corporate data is only updatable and readable through the stewarding DLBB. The DLBB provides the functions of create, retrieve, update and delete capabilities. This prevents unknown sources trying to read or damage the data.

3. **Openness:** The DLBB has the capability to provide query contracts. The building block allows adhoc query of all corporate data, that, it stewards for all authorized users. The queries are based on an implementation data model and cast into an industry accepted standard query language. It also provides other predefined queries and alternate views. Thus the principle insures that all the data is accessible by authorized users via ad-hoc queries and it also provides for other predefined queries and alternate views, thus ensuring that all the data is accessible by authorized users only through ad-hoc queries. An accepted industry standard is used and the language will change as the technology advances. Such changes have to conform to the contract principles. At

present the industry accepted query language is SQL.

The DLBB may provide the contracts with present predefined queries and alternate views. These are not accessed through the ad-hoc query language and all the data may not be available to them.

4. **Integrity**

   The DLBB ensures the semantic integrity of the data it stewards. It ensures that the corporate data is valid for all states intrinsic to the data. Semantic integrity is described as:

   (a) **Constraint Scope**: The database space may be constrained because of

      i. Entity Constraints: It constrains the contents of the database entity, such as a record or segment;

      ii. Intra-entity Constraints: The database entities of the same class is constrained;

      iii. Inter-entity: Constrains the database entities of different classes.

   (b) **Temporal Class** is the span of time over which the constraint applies,

      i. static constrains in a database;

      ii. dynamic constrains are the transition between the database states;

      iii. historic constrains the specific time period.

5. **Redundancy**

The Corporate Data may be duplicated in any of the three OSCA^{TM} layers as **Local Data**. The data could be duplicated into the corporate data layer itself, to provide alternate views of the data or to help to provide alternate views of the data. Or also to provide help for adequate performance and availability. Appropriate functionality to create and update the alternate views is provided in the appropriate DLBB. Corporate Data may be duplicated in the Processing and User layers to provide alternate views or enhance the performance.

A DLBB provides means whereby updates to redundant copies of the data that it stewards can be provided to the building block owning copies of the data either on request or automatically. If there is an inconsistency in the redundant copies, the corporate copy stewarded by the DLBB is assumed to be correct.

1.4.2 Processing Layer Building Blocks

The processing layer is fundamentally related to the business processes. The Processing Layer Building Blocks (PLBB) numerous functions include crunching numbers, construct report contents, do complex queries and control process flows. The PLBB's provide value added services using the Corporate Data, that, an end user requires. It doesn't own or steward corporate data, whereas it uses and produces corporate data. PLBB's can obtain data from and send data to the data layer. Whenever corporate data is required, the PLBB's request the appropriate Data Layer Building Block which is responsible for accessing the data and returns the required results. The PLBB's adhere to the principles given below:

1. **Functionality**: Any functionality that is not expressly forbidden in the following principles and that which cannot be construed as communication software fabric, data layer or user layer functionality and which guarantees support of its contract over internal changes;

2. **Corporate Data**: Does not steward corporate data;

3. **User Layer Functionality**: Does not perform any function enumerated exclusively for a User Layer Building Block.

1.4.3 User Layer Building Blocks

The OSCA^{TM} architecture requires that the separation of data management and processing functions from physical presentation functions. This separation promotes:-

- freedom of applications from concern with particular user devices, potentially providing wide spread availability, facilitating adaption and development of new technologies and prompting efficient access to building blocks by intelligent workstations;

- isolation of deleterious effects resulting from changes in the business functions, underlying database or user interface of an operating system;

- incorporation of user interface tool kits;

- simplified procedures for customization of the user interface;

- deployment flexibility, so as to facilitate distributed processing and co-operative work;

- User Layer Building Block principles promote standardization.

The User Layer is the user's gateway to request services of multiple building blocks. It serves a human user with whom it interacts directly or indirectly when, used through printers or ULBB, and relieves other building blocks that perform the function of database and/or business functions from dependencies on end user devices such as terminals and printers.

Functionality of the User Layer

The essential functions of the user layer is to map semantic information into a physical presentation and to map user actions back into a semantic representation. This layer is responsible for maintaining a predictable, intelligible environment for the user, so as to insulate the user to the degree possible from failures external to itself, either in other building blocks or the communication software fabric.

There are three groups of functions in the User Layer that are divided into required, additional and discretionary functions. Required functions are those that the User layer must supply on behalf of the user. Additional functions are optimal capabilities that deal with data presentation and collection, if supported they belong to the User Layer. Discretionary functions are those that provide capabilities within the User layer although these capabilities may not directly relate to data presentation or collection.

1.4.4 User Layer Functions

The main functions which OSCA^{TM} lists are not mutually exclusive and some could be, the sub sets of a higher level functions. They are:

1. **Required User Layer Functions:** These functions must be performed on behalf of the user in the User Layer; however not every User Layer Building Block needs to provide all of these required functions. The required functions are:

   (a) Physical Mapping;

   (b) Presentation Structuring;

   (c) User Message Delivery;

   (d) User Authentication.

   Additional User Layer Functions are those which deal with the physical aspects of the presentation and/or need to be in close proximity to the end user. These are optional and important to all interfaces. These functions are:

   (a) Data Entry and modification;

   (b) Command Parsing/Processing;

   (c) Display Processing;

   (d) Menu Processing;

   (e) Report Formatting;

   (f) Help prevention;

   (g) Interactive text and design tools;

(h) Session and Window Management and

(i) User Customization.

The discretionary User Layer functions are the ones that reside in the User Layer Building Block. These functions are :

(a) Local Validations;

(b) Translation;

(c) Arithmetic Calculations.

2. User Layer Contracts

The user layer is the user's "**gateway**" to perform business function in OSCA^{TM}. The user layer serves as the user's agent to invoke contracts in other building blocks (BB's) and to receive, appropriate requested data and messages which it passes back to the user. Therefore the user layer is primarily a consumer of contracts executing external to it.

3. User Layer Building Block Principles

These are intentionally made flexible to allow for a wide variety of implementations. The internal structure of the user layer building block is not prescribed or constrained by OSCA^{TM} as long as the general building block principles and user layer building block principles are maintained. The major principles are

(a) perform atleast one required or additional user layer function;

(b) may support contracts for required or additional, but not discretionary user layer functions;

(c) does not steward corporate data nor perform any function or service enumerated exclusively for a data or processing layer building block.

4. User Layer Principle

In addition to the building block principles, OSCA^{TM} stipulates one higher level principle for the user layer, which is the combined service of the user layer building blocks accessed by a user, that must provide all the required functions of the user layer.

5. General Building Block Principles

Some of the important principles that have a peculiar effect on the user layer are:

- release independence;
- a building block can execute in only one recoverable environment;
- secure environment.

1.5 Communication Software Fabric

This provides the software services and infrastructure for the inter-building block communication, building block availability and performance management. In a distributed environment, such as the one promoted by OSCA^{TM}, no single building block provides all the services required by a user. In such an environment, multiple building blocks operate together in a co-operative effort and communicate their results to each other. The communication software fabric is the "**glue**" that ties the building blocks together. It also allows this co-operative effort in providing services to the Users.

1.5.1 Composition

Internally, it consists of a communication network that provides the basic connectivity and services, that are provided to the application logic and which reside with in a building block. Some of this functionality is co-located with the building block at the "**edge**" of the network, as part of shareable services. Other functionality is provided from "with-in" the network. This is accessed through the connectivity provided by the communication software fabric. These communication services are able to foster and ease conformance.

1.5.2 Communication Software Fabric Service Classes

The typical Commmunication fabric service includes the following:

(a) **Communication Services:** These services allow contracting among the building blocks by moving the data from one building block to the other.

(b) **Communication Support Service:** These assist both the building blocks and the Communication Services to get the data back to its destination.

(c) **BB Management Service:** This service manages the distributed computing environment that OSCA^{TM} architecture specifies.

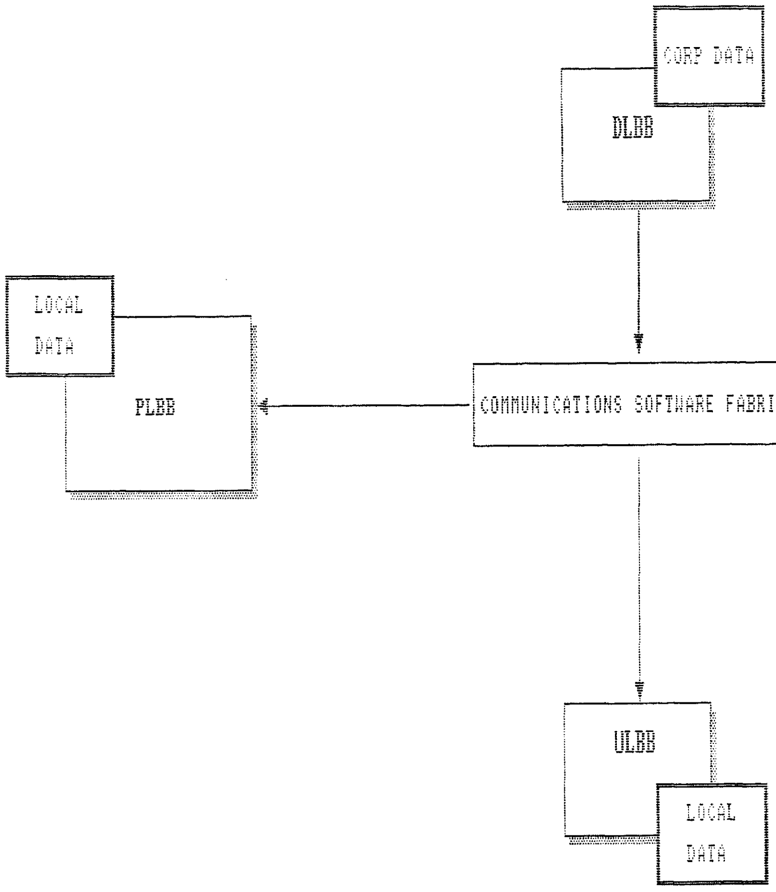

Figure 1.4: **Target Architecture**

1.6 Summary and Conclusions

The objective of OSCA^{TM} architecture is to be built upon the ISO and CCITT standards as far as they are available and commonly agreed upon. These standards are targeted to interfaces among building blocks and to the interfaces with in the communications software fabric.

The network functionality is allocatable to the data, processing and user layers, and this data is either Corporate or Local. The operating systems and the networking elements can thus operate as peers within OSCA^{TM} architecture.

A number of concerns that must be addressed to apply the OSCA^{TM} architecture to the network functionality are

- A data architecture analysis is needed to determine what is corporate data. For example, some operating systems traditionally, hold a view that, the data is what the customer has ordered and if so, are these of the same views or are there different views, and if so, what is their relationship?

- If the networking environment hosts a data layer building block, how will the required performance of the computing environment be maintained and how will the availability, required for the service process be attained. When will the ad-hoc query and business oriented semantically valid updates be provided?

- In the networking environment, to what extent can OSCA^{TM} principles be applied to provide the principles of interoperability with other functionality?

- Can future technological advances such as semantic and knowledge based DBMS and commercial grade heterogenous DBMS be utilized, without major software product rewrites, beyond the affected data layer building within OSCA^{TM}'s principles?

Chapter 2

ANSA Architecture

2.1 Introduction and Overview

2.1.1 What is ANSA Architecture

ANSA stands for **ADVANCED NETWORKS SYSTEM ARCHITECTURE** used as a part of the ODPSE architecture [ANSA89]. ANSA was originally designed in 1986 by various European companies such as British Telecom, DEC, GEC/Marconi, GPT, HP, ICL, ITL, Olivetti, Plessey etc within the U.K Alvey Information Technology Program. In 1989 a company by name Architecture Projects Management Ltd (APM Ltd) was floated, to continue further work on ANSA for the sponsors at Cambridge, England.

ANSA Project receives its funds through the Commission of the European Communities (CEC) Espirit II Program within a project called Integrated System Architecture (ISA). There are many sponsors for APM Ltd., which are companies such as AEG, CTI-Patras, Erricson Telecom, Televerket, Phillips, Siemens and France

Telecom.

2.1.2 Objectives and Goals

The main goal of ANSA is to provide an architecture for distributed systems that satisfies the objectives given below:

- the architecture should be generic to many fields of application;
- the technical content contains state of art technology;
- the architecture is portable across a wide range of operating systems and programming languages;
- it should be operateable in heterogeneous and multivendor environments;
- it is modular in structure, with maximum opportunity for reuse of the existing functionality;
- it supports for a range of distribution naming, concurrency and fault handling policies;
- is applicable to a wide range of computers and network topologies with no constraints on size;
- oriented towards the requirements of application programmers;
- it focuses the open distributed processing for support environment (ODPSE) part of architecture.

2.2 Concepts

The concepts associated with ANSA are:

Architecture: Is used for building distributed systems in the form of an integrated set of structures, functions, design recipes and implementation guidelines.

Testbench: The software is developed to demonstrate and validate the architecture. This known as the **ANSA Test bench**. This test bench can be operated on various operating systems and machines such as HPUX, SunOs, Ultrix, VMS, MSDOS etc.

Standards: The results contributed by ANSA should confirm to International standards.

Technology Transfer: The transfer of the architecture and the testbench as a technology to both the sponsors of ANSA and to the EDP community at large.

2.2.1 Standardization

ANSA has been designed on the concept, that the architecture should adopt and not conflict with the current open standards where ever possible. Aspects which fall outside the scope of current open standards are taken into the open standard process. The ANSA team participates actively in ISO/IEC JTC1 SC21 WG7 on the standardization of a reference model for Open Distributed Processing and with ECMA TC32-TG2 as a Open Distributed Processing for Support Environment (ODPSE). The ANSA team participates to a lesser extent in a number of other standards groups[ANSA 89].

2.3 Architectural Philosophy

The philosophy and goal of the ANSA project has primarily been to develop an architecture that provides the simplest concepts necessary to build distributed systems. This philosophy had profound effect on the design of the architecture and the Testbench.

2.3.1 Viewpoints On Distributed Processing

A study had been conducted by the ANSA scientists and it was revealed that to make up the '**distributed processing**', five viewpoints were dominant. Each viewpoint in some way or the other acknowledges the concerns addressed in other view points, but with a lesser priority. As a result, the description of ANSA is structured into models representing five viewpoints. The models that describe these view points are enterprise, information, computation, engineering and technology. A distributed system could be described using any of these models of ANSA.

1. **Enterprise Model**

   The purpose of the enterprise model is to provide a framework for explaining and justifying the role of an information processing system within an organization. An enterprise is one that describes the overall objectives of the system in terms of roles for people, actions, goals and policies. It specifies the activities that take place within an organization using the system, the roles that people play in the organization and the interactions between the organization, the system and the environment in which the system and the organization are placed.

2. **Information Model**

   This model provides a framework to describe the information requirements of a system. An information description of a system is made up of structures of information elements in a system, constraints on the information elements and the rules.

3. **Computation Model**

   This model provides a framework for modelling the operations of information transfer, retrieval, transformation and management necessary to automate information processing. The mechanism required to support the computation model thus defined are specified in the engineering projection of the system. The computation description of a system, partitions the required transformations among processing objects as necessary to achieve the complete set of transformation. The partition thus defined is logical and not location dependent.

4. **Engineering Model**

   This model provides the framework for describing how to mechanize an application definition identified using the computation model. This support will include definition of physical distribution to realize the partitioning defined in the computation projection.

5. **Technology Model**

   This model provides the framework for describing the technical artifacts from which the distributed system is built. This could include OSI and proprietary standards as needed. It shows how the hardware and software that comprise the local operating systems, the input/output devices, storage, points of access to communications are mapped onto the mechanisms identified in the engineering model.

6. **Distribution Transparency**

   Various forms of distribution transparency are available to the programmers which they can choose from. These transparencies determine the extent to which the programmers need to be concerned with and have control over the integration of disparately located pieces of application programs.

The various types of transparencies supported by ANSA are :-

- **Access transparency** provides identical invocation semantics for both local and remote criterion;
- **Location transparency** hides the exact location of a program component from any other component that interacts with it;
- **Concurrency transparency** hides the existence of the concurrent users of a service;
- **Failure transparency** hides the effects of partially completed interactions that fail for whatever reason;

- **Replication transparency** hides the effects of having multiple copies of program components to provide for an increase in dependability or availability;
- **Migration transparency** is a dynamic form of location transparency that hides the effect of a program component being moved from one location to another while it is being used by another component.

Currently ANSA testbench software supports - Access, Location and Concurrency transparencies.

2.4 Design Of ANSA

ANSA is an architecture for building distributed systems that can operate as a unified unit such that the fact of distribution is transparent to the application programmers and the users. ANSA allows full advantage to be taken of the inherent concurrency and separation of distributed systems in order to increase the performance, decentralization and reliability, while masking their disadvantages such communication errors, partial failures etc.

ANSA follows a programming language view, which means that the distributed computing concepts should be represented by extra syntactic constructs that could be added to the existing programming languages. These could directly be compiled into calls at the system level. The main advantages of this system are :

- a simple programming model for application programmers;
- checking at compile times;
- independence of the application programmers view from the systems point of view which makes the applications and systems compatible with future modifications.

2.4.1 Model For Computation For Ansa

This model is a framework of programming structures and program development tools that should be made available to distributed application programmers, irrespective of the application programming language they choose to use. This model addresses the topics given below:

- modularity of distributed application;
- access transparent invocation of operations in interfaces;
- parameter passing scheme;
- configuration and location transparency of interfaces;
- replication constraints on interfaces;
- extending existing languages to support distributed computing.

This concept specifies that maximum engineering flexibility is obtained if all computation requirements of an application are expressed declaratively. This allows tools to be applied to the specifications to generate the code satisfying the declared requirements. It allows a clean separation between application programmers by stating the requirements and system programmers by providing the tools. These tools use the requirements in the environment, the appropriate quality to the task in hand. This means that by making this separation it is possible to identify the different forms of transparency that are required by a distributed application. This technique should be able to choose the appropriate technique for providing the required transparency for each application.

2.4.2 Engineering Model For Ansa

The Engineering Model is a framework of the compiler and the operating system component for realizing the computation in heterogeneous environments which are as follows:

- thread and task management - a thread is a function to be carried out, and is allocated to a task which will execute it;
- Address space management;
- Inter address space communication;
- Distributed application protocols;
- Network protocols;
- Interface locator - identifies the location of a specific interface;
- Interface traders - provide directory facilities for identification of interfaces, both imported and exported;
- Configuration managers;
- Atomic operation manager;
- Replicated interface manager;

This model provides the system designer with a view of engineering trade-offs that are available, when providing a mechanism for a particular function as defined in the computation model.

The implementor may vary the quality attributes of a system by making trade-offs in terms of its dependability (reliability, availability, safety, security) and performance without disturbing its function. This is an important function of ANSA since it decouples application design from technology to a certain extent.

In the computation model, the programmer is given the guarantee that his program will be able to operate in a variety of different quality environments without modification of the source.

The Engineering model gives the system implementor a toolbox for building an environment of the appropriate quality to the task in hand, which means that by making this separation it is possible to identify what forms of transparency are required by a distributed application and be able to choose the most appropriate technique for providing the required transparency for each application.

2.4.3 Overall Structure

The way in which the various components of ANSA fit with one another is shown in the three figures shown. We are only interested in figures (1) and (2) more than figure (3).

Figure (1) shows two ANSA systems. Each system is running several applications shown as **'A'** in the figure. These are linked together with a trader **'T'** and a configuration manager 'C'. The trader provides a directory structure that can be searched by path name, property values or by combination of the two. A server can export an interface reference to the trader to make it accessible to other applications. An import operation is provided to clients so that they can retrieve interfaces from the trader. The configuration manager provides the means to start new application components executing in an ANSA system. To provide federation between the systems, the two traders are also linked together. This enables an application to export an object which the other imports, such that to the user the distributed system appears to be running on a single host.

Figure 2 shows us nucleus components. These are represented to us as **'N'** or **'NUCLEUS'** along with the Trader and the Configuration Manager which are a part of the ANSA platform. These take the basic resources of the local infrastructure and build on them to provide the basic distributed computing environment common to each host, which are shown as **'host systems'** in the third level. These nucleus components are then able to work together, along with the trader and configuration manager (which may themselves be distributed) to provide a basic support platform for distributed computing.

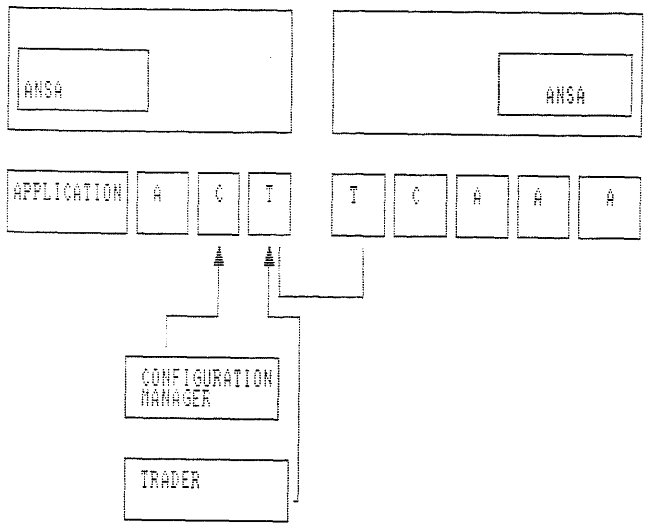

Figure 2.1: **A Federation Of ANSA Systems**

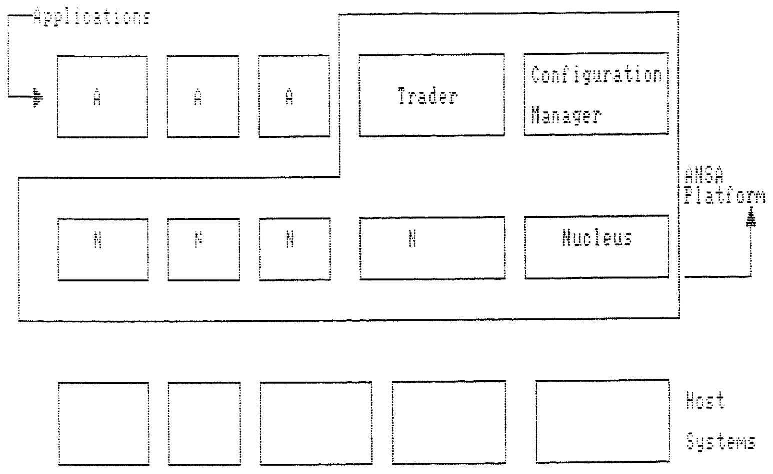

Figure 2.2: **ANSA System**

Figure 3 is an expansion of the nucleus components in both the directions. the transparency components '**T**' provide additional functions which enable the various aspects of distribution to be made transparent to the applications. Below the nucleus there are components to provide Execution Protocols '**E**' and Message Passing Protocols '**M**'. If interworking between the heterogeneous systems is not required, either or both of these could be replaced by local equivalents. Below these will be the local cpu management '**P**', communications '**C**', memory management '**S**' and other local functions '**F**'.

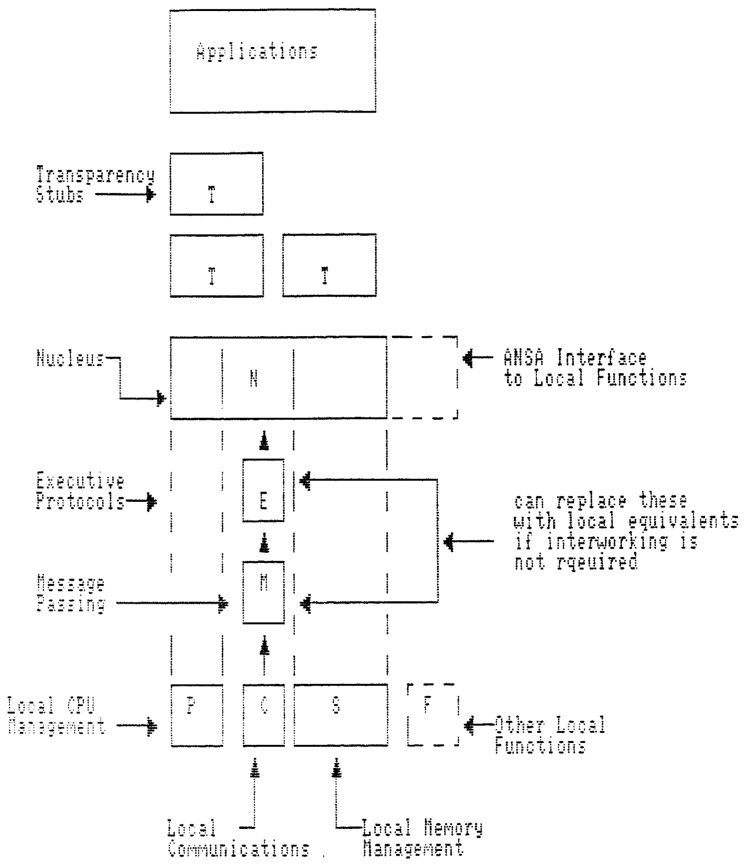

Figure 2.3: **ANSA CAPSULE**

2.4.4 Implementation

The ANSA Testbench software is a suite a ANSI C Programs that conform to the architecture. These represent an instantiation of the results of the architecture intended for porting across the current generation of operating systems and network protocols. In particular, the standard distribution includes porting instructions for SunOS, HPUX, Ultrix, VMS, MSDOS.

The Testbench software consists of modules that are discussed below:

1. A threads management policy is to provide for concurrency within an address space, if it is not provided by the host. Concurrency is needed so that the servers can respond to multiple clients in parallel. In addition so that the clients can distribute the computation in time to perform parallel tasks. As well as perform remote tasks or run remote tasks in parallel.

2. The function of the address space management package is to complement the threads package with facilities for managing multiple stacks, communication buffers and a shared heap within a single address space. To sup port true concurrency multiple stacks are necessary.

3. An inter address space communication package also known as the interpreter is to provide an implementation independent standard interface for interactions between threads in separate address spaces.

4. A remote execution protocol (REX) provides for the messages to be transported to implement the communications requirements of the inter address space communications package. This provides functions of transport, error

recovery, fragmentation of large messages and control of optional end to end connections.

5. An interface description language (IDL) is used to describe the interfaces between the application components.

6. An IDL processor reads interface descriptions and generates libraries of stub procedures in C language. These procedures handle the packing/ unpacking of arguments or results into/from buffers of transmission and exchanging buffers between the distributed portions of the application. These functions are known as marshalling, unmarshalling and communications respectively.

7. An application description language (DPL) for C. The DPL preprocessor for C extracts statements that augment an ANSI Standard C program to connect to interfaces and invoke remote operations. These statements are then translated into calls to the appropriate stub procedures and interaddress space communication package calls.

8. A Trader is a distributed application component which acts as a directory and management facility for distributed application components.

9. A Configuration Manager is also a distributed application component which provides a means to instantiate application components above the platform.

The ANSA architecture is not restricted to any particular programming language, operating system, network or hardware platform.

2.5 ANSA's Relationship To ODPSE

The ISO ODP scope, states:

"This standard will be concerned with, and limited to, the general aspects and common features of distributed systems". It will provide for:

(a) common definitions of concepts and terms for distributed processing;

(b) a generalized model of distributed processing using these concepts and terms;

(c) a general framework for identifying and relating together open distributed processing standards.

As per the above paragraphs ANSA is based on the principles of ODPSE. ODPSE stands for Open Distributed Processing for Support Environment. Then the question comes what does ODP stand for ?

ODP stands for Open Distributed Processing which is:

(a) an attempt to establish a consistent framework for networked distributed application;

(b) the support environment is required to enable the implementation of distributed application in an open vendor independent way [ODPSE].

An Open Distributed Application is a set of software components which cooperate independently irrespective of their location within a global network (which consists of several vendor networks). These components can run on any vendor application environment that contain an ODPSE. Given on next page is figure 4, that shows the Systems and Vendor Networks.

A Distributed Operating System tackles the task of distributed applications in a homogeneous environment by establishing the distribution platform at the Operating System Level. The main idea of an identical operating system is that it has to run on all systems that participate in a distributed application environment. The Distributed Operating Systems do not unify information processing on an enterprise, wide scale as to a large extent the enterprise information processing system are heterogeneous.

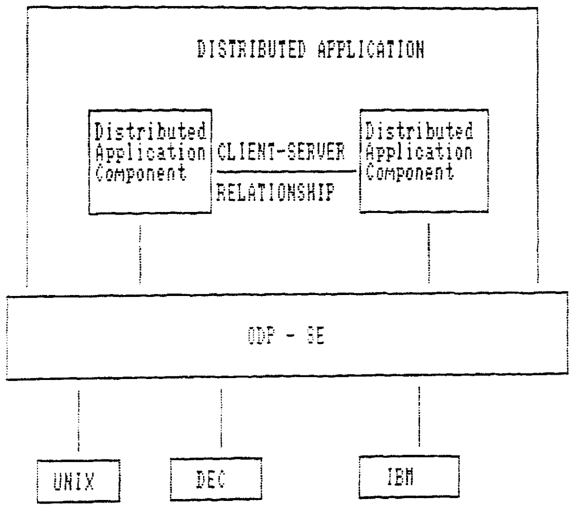
DISTRIBUTED APPLICATION
Distributed Application Component
CLIENT-SERVER RELATIONSHIP
Distributed Application Component
ODP - SE
UNIX
DEC
IBM

Figure 2.4: **System and Vendor Networks**

2.6 Summary and Conclusions

The ANSA architecture follows the principles of object oriented design, this is carried through to an object oriented implementation for the Testbench within the limitations of C language.

The architecture emphasizes the concept of generic functions and narrow interfaces as the key to minimizing concepts and enabling maximum reuse of the components.

This Project has a strong commitment to see that its work is placed in the public domain by actively participating in relevant standardization activities.

Chapter 3

Thoughts On Integration Of OSCA^{TM} And ANSA

3.1 Integration Architectures

3.1.1 Introduction

This part of my thesis presents conceptual ideas of how OSCA^{TM} could be integrated with ANSA. As said earlier in Chapters 1 and 2, both these architectures support the ODPSE architecture either directly or indirectly.

A distributed operating system is a program or a set of programs running on various types of computing environment that are interconnected by a network. The distributed operating system unifies computers of different architectures into a single integrated compute and storage resource. Depending upon the facility it provides, a distributed operating system is classified either as general purpose, real time or embedded [CLOUDS91]. ODP or Open Distributed Processing is one step ahead of distributed operating system [ODPSE].

It is:

(a) an attempt made to establish a consistent framework for networked distributed applications as well as;

(b) the support environment required to enable the implementation of distributed applications in an open vendor independent way.

As networked computing environments have become popular these days, powerful computing systems have become more affordable. Most computing environments consist of combinations of various types of architectures like workstations, main frames, minis, and personal computers. In the network environments, the fact of distribution is kept transparent to the user without affecting his productivity and the complete environment must appear like a centralized pool of resources. Also in a network environment there is an increasing level of interoperability which results because

1. customers will need access to operations and information that span multiple discipline oriented operating systems and network elements;

2. many operating systems and network elements can be spanned when a new service is implemented;

3. flexible environment is needed to develop new services to access network element data and functionality.

A distributed system is one where it makes a collection of different computers look and feel like one centralized system, yet keeps the advantages of distribution intact. There are two paradigms that fall in to this category of distributed systems. These are (1) Channel-based or message based and (2) Object Based. In my thesis the paradigm of Channel based or message based system comes in the view of OSCA^{TM} and Object based system for ANSA. As said in Chapter1, the Contracts support local communication with in the same level of building blocks or at different levels.

In an object based system, services and resources are encapsulated into entities called "**Objects**". ANSA which falls under this system uses the approach of Object Orientedness through its computation model. Objects are similar to the instances of abstract data types. They are written in individual modules composed of specific operations that define the module interfaces [CLOUDS91]. The framework for an integration model in general consists of three major components mentioned below:

1. Enabling Technologies;
2. Integration Architectures and
3. Global Integration.

1. **Enabling Technologies** form the basis for system integration by providing the required building blocks to begin with. It talks about the mechanisms, tools and systems that could be used for system integration. This addresses the mechanisms, systems etc., which could be used as a basis for system integration.

2. **Integration Architectures** describe the use of these building blocks that help in forming a system which is integrated internally and allows for future expansion. It refers to the idea of an open architecture which is implemented on the basis of enabling technologies.

The elements of an integration framework that are described within an integration architecture should be contributed by the :

(a) conceptual layout of the architecture that consists of specification of standards and restrictions of components of modules, communication and data storage;

(b) the mapping of the domain model into the architecture;

(c) the applied standards and

(d) the guidelines for implementation.

In contrast to the standards for an integration architecture, the standards for implementation describe the technological basis on which the generic integration architecture is implemented. The standards for communication are specified on top of the application model,e.g. using the ISO/OSI model at the level of the integration architecture.

3. **Global Integration** describes the coordination and fine tuning of the system on its semantic and interface levels.

   NOTE:

   This part of my thesis talks about a framework of integration which is based on integrating the OSCA^{TM} Architecture of Bellcore [OSCA90] and ANSA architecture of APM Ltd [ANSA89]. (See also Chapters I and II).

3.2 OSCA^{TM} And The Integral Framework

OSCA^{TM} of Bellcore [OSCA90] as discussed in Chapter I of this thesis is an example of an open system approach at the company level to guide the process of integration. OSCA^{TM} specifies a "strategic architecture to be used by the Bellcore Client Companies to provide software interoperability."

Interoperability is the ability of the building blocks to communicate with each other, and the users to communicate with any building block irrespective of the internal architectures and the environments on which these building blocks reside. OSCA$^{TM's}$ approach is targeted towards the definition of an integration architecture or a "meta-architecture by providing the guidelines and constraints for product specific architectures", instead of a process model for the management of integration.

3.2.1 Channel Based Approach

From our point of view, OSCA^{TM} follows a modified channel based approach for its basic layout. We assume that a channel is existing and it is transparent. The channel provides the means of working on different machines. A Channel based approach separates the "**passive**" communication components from the "**active**" functional components. The building blocks of OSCA^{TM} form the "**active**" components and the Communications Software Fabric is the "**passive**" channel component. This provides a communication infrastructure to transmit standardized messages on the application level, from the "**active**" components which constitute the functionality and data storage capacity of the system. The channel and the passive components transport data and files in the form of standardized messages "**CONTRACTS**" at the system level. The existence of the communications software fabric in OSCA^{TM} hides the networking effort underneath the specified channel interface. The channel acts transparently for the building blocks and delivers messages or returns error messages as a guaranteed service.

The Channel Based approach goes with the concept of distributed data storage. Therefore, message passing integration architectures must take into consideration data integrity by specifying the communication protocols and the interaction among the distributed components which handle the data in the system.

OSCA^{TM} takes care of this problem, by introducing the concept of "**stewarding**". Stewarding means that the Data layer of the OSCA^{TM} provides sufficient and necessary functionality to

1. update and access Corporate Data,
2. preserve the semantic integrity of the Corporate Data and
3. allow appropriate security measures [Mills 90].

The Message passing systems focus their concern on the communication side of the architecture and leave the distributed handling of data to the distributed components of the system which behaves like data capsules. As said in the Introduction of this part of my thesis, we find that there are two classes of message passing systems - Object Oriented systems and Channel based systems. (**Remark**: The Channel based approach acts like an object oriented approach with regards to message passing, but does not include hierarchy of classes and inheritance mechanism.) As said earlier OSCA^{TM} falls into the Channel based system approach.

While OSCA^{TM} talks in detail about the three levels or layers of functionality namely, the data, the processing and user layers, it is relatively silent on its channel the "**Communication Software Fabric**". This fabric connects the three levels of functionality along with their building blocks. It allows the building blocks to communicate with each other with in the same level or at other different levels.

This communication software fabric, as specified in [OSCA90], doesn't specify the communications networks but provides only the architectural guidelines which the chosen network must satisfy.

3.2.2 Domain Model

OSCA^{TM} uses the concept of "**separation of concerns**" in the model, where every component in the system is related to one of the three layers of functionality, namely:

- the data layer;
- the processing layer;
- the user layer.

Every functionality in OSCA^{TM} Architecture relies on this abstract model of "**Separation Of Concern**". This is done by defining the handling of the user, processing and data activities as layers of functionality. The architecture maps this very general domain model onto a model at the next lower level by placing the building blocks, exactly on one level of functionality and by explaining the dynamic behavior of the system in terms of those layers.

However, OSCA^{TM} never mentions something like a domain model explicitly. The Three layers of functionality are chosen to allow all different sorts of application systems to be handled within the framework of OSCA^{TM} [MILLS 90]. From this point of view, OSCA^{TM} has no real model of an application domain as a basis, but relies simply on separation of groups of functionality.

3.2.3 System Architecture

Open Architecture:

Specification design and eventually implementation details should be made available to every user or a vendor of the system. An open system should define the system parts given below:

1. basic architecture;
2. user interface;
3. data storage and representation;
4. system function;
5. data transfer and
6. using enabling technologies.

The concepts used in this thesis are based on the layered models of architectures like OSCA^{TM}. These architectures talk about the basic technologies, specify the integration architecture and continue on the problems of concept integration over multiple system parts. System Integration is achieved by pasting the existing parts together as in a jigsaw puzzle through an integrated approach. To co-ordinate the integration process either the top down or bottom up approach is taken into consideration.

3.3 Other Aspects of OSCA^{TM}

As said earlier, OSCA^{TM} Architecture relies on an abstract model of "**separation of concerns**", by defining the handling of user, processing and data activities as layers of functionality. The architecture maps this process model onto a model at the next lower level by placing the building blocks on exactly one level of functionality and by explaining the dynamic behavior of the system in terms of those layers.

OSCA^{TM} follows a standard format where messages are sent via a predefined communication system to other building blocks at the same level or at different levels. This scheme of message passing is a variation of the object oriented approach/ channel based approach. As I mentioned in section 3.2.1 the building blocks of OSCA^{TM} are the "active" components. Objects are said to be "active" when an active object has one or more processes associated with it that communicate with the external world with other building blocks at the same level or different levels, and handles the task it is supposed to do internal to the object. For example, a process can monitor an object's environment and can inform some other entity (another object) when the event has occurred. Conceptually, an object is an encapsulation of data and a set of operations on the data. The operations are performed by invoking the object and can range from simple data-manipulation routines to complex algorithms, from shared library accesses to elaborate system services. Objects can gather data from a device without knowing about the mechanisms involved in accessing it or its locations.

Objects are a simple concept with a major impact. They can be used for almost every need - from general purpose programming to specialized purposes, but yet provide a simple procedural interface to the rest of the system. However, OSCA^{TM} is relatively silent on the object oriented aspects of the architecture. It could be assumed that the building blocks at different levels of functionality are the "active" components which do a similar function like objects.

The separation of corporate data is one of the main feature of the OSCA^{TM} Architecture. This data is accessible by the users of the system who are widely distributed geographically with in the BCC environment. It allows for the failures to be reported which occur during communication and partial failures that occur during execution of a program. The Building Blocks act like objects and encapsulate the data. This provides the functionality when a question is asked and the answer is got back.

OSCA^{TM} favors a "top down " approach which allows a more traditional phased model of system integration. This allows a system like OSCA^{TM} to provide a basis not only for technical integration but also for the management of organizational, budgetary and legal aspects.

This allows us to integrate the already existing components such as the building blocks, the contracts, the communications software fabric and to provide a framework for interface definitions, performance and cost evaluation. The architecture is generic and robust, enough to provide the flexibility and adaptability for changes in requirements and technologies.

3.3.1 Conclusions for OSCA^{TM}

The global functions of OSCA^{TM} like "**user interface**", "**stewarding of corporate data**" and "**functional processing**" are derived from the application domain. The global integration activities such as semantic integration can be handled in a top down fashion. While OSCA^{TM} talks in detail about the three levels or layers of functionality namely, the data, the processing and user layers, it remains relatively silent on the "**Communication Software Fabric**" implementation.

The principles of Architecting and Archetyping are used to overcome the difficulties of system development and integration [EISN90]. The phase of archetyping, is independent from the subsequent building phase and constitutes the phased top down approach. This is further divided into two steps

1. architecting;
2. prototyping.

Architecting uses a top level design approach to construct a model of the system. With respect to OSCA^{TM}, the Building Blocks the Contracts specification and the design of the Software Fabric falls with in the principles of Architecting. Whereas the prototypes of the building blocks, the implementation of the contracts, the implementation and testing of the Communication Software fabric fall into the principles of Archetyping [ROSSAK91].

3.4 ANSA And The Integral Framework

ANSA favors a bottom up or the "**POST FACTO**"approach . ANSA basically focuses on a programming language which is used for interconnecting various parts in a heterogeneous system. When an architecture follows a bottom up or POST FACTO approach then it has the following properties:

- parts design precede system design;
- system is heterogeneous;
- multilingual;
- loosely coupled;
- parts are typically medium to large size;
- non standardized reuse.

In ANSA, the above mentioned properties can be seen in one or all of the five models (the Enterprise Model, Information Model, Computation Model, Engineering Model and the Technology Model) [ANSA89].

An Enterprise model follows the property that the system is heterogeneous, because the purpose of this model is to provide a framework for explaining and justifying the role of an information processing system with in an organization. It describes the overall objectives in terms of roles the people or users play or the actions, goals and policies that they do.

The Information Model follows the property that the system is heterogeneous ,because this model provides a framework to describe the system which is made up of structures of information elements. It also states the rules and constraints that state the relationship about the elements of the Information System. It also follows the property that the parts are typically medium to large size because this model shows how the information is partitioned across logical boundaries and has required quality attributes. This model also has the property that it is loosely coupled, because it does not have to differentiate between parts that are to be automated or performed manually.

The Computation Model follows the rule that the parts design precede the system design. This is because it provides a framework for modelling the operations of information transfer, retrieval, transformation and management that is required to automate information processing. This model also concentrates on the problems and opportunities presented by the execution of applications of many loosely coupled computer systems. ANSA plays follows the function of multilinguality. As ANSA is an architecture for open systems and it is not viable to impose a single language. Heterogeneity is another important property of ANSA, where various parts of a large application could be written in different languages for various reasons like history or suitability. The property that " parts design precede system design" could be seen in detail in the Computation Model. The design philosophy of this model is to find the smallest number of concepts that are needed to describe distributed computations and to propose a declarative formulation for each concept rather than imperative formulation [ANSA89].

The Engineering Model of ANSA follows the property of non standard reuse because it provides a framework to describe how to mechanize an application definition used in conjunction with the Computation Model.

The Technology Model follows the principle or property that parts design precede system design because this model provides the framework for describing the technical or realized components from which the distributed system is built. It shows how the hardware and software make up the local operating systems, the Input/Output device storage, points of access to communication are all mapped to mechanisms in the Engineering Model.

ANSA uses concepts similar to Object Orientedness through the Computational Model. It takes the scoping and encapsulation mechanism down to the level of single data structures and data types. Embedding or encapsulating the programming components by wrapping them up in the necessary distribution transparency is accomplished by the use of Object Oriented philosophy and by the separation of the interface specification from object definition in the ANSA's computation model. The Computation Model includes syntactic structures and a flexible type system which permits a wide range of checks that are to be made statically at compile time, without compromising the ability of the programmer to defer some decisions to run time through the use of explicit control.

ANSA doesn't speak about the object orientedness but tells us how these objects communicate. ANSA implements mechanisms to send messages even though it is not concerned that these objects would use it's services. All data is stored in objects and accessed indirectly through interfaces. This is dealt by the Computation Model which deals only with interface references. The Computation Model of ANSA specializes in distribution, by packaging sets of operations into interfaces. This is to restrict the scope of operation names as tightly as possible and by always accessing interfaces indirectly, so that location transparency is maintained [ANSA89].

3.5 Ideas In Integrating OSCA^{TM} And ANSA

To integrate OSCA^{TM} and ANSA the following aspects should be taken into account. OSCA^{TM} talks about the layers, contracts and the building blocks in detail but is silent on the communication software fabric. So here in this section we see how ANSA could be integrated as the communications software fabric of OSCA^{TM}.

ANSA supports the ODPSE type of architecture [ANSA89]. OSCA^{TM} doesn't mention about the ODPSE architecture. It could be a part of the ODPSE architecture, because users and data are located at different places within the BCC configuration [OSCA89]. This feature of users and data being distributed widely geographically is a typical feature of the ODPSE architecture.

ANSA follows the programming language paradigm in its test bench to integrate its modules. Whereas in OSCA^{TM}, the programming aspects are handled inside the building blocks, which are hidden behind the interfaces. Since ANSA is an architecture for open system, it is not viable to impose a single language as the conformance criterion for the computation model [ANSA89].

OSCA^{TM} follows a top down approach in a networked environment and stays at a much higher level, whereas ANSA follows a bottom up approach. If OSCA^{TM} and ANSA are to be integrated, it will be necessary to apply a mixture of both both "top down " and "bottom up" approach. Since OSCA^{TM} is a channel based architecture, ANSA could be integrated as OSCA^{TM}'s Communication Software Fabric.

For ANSA to function successfully as the Communication Software Fabric of OSCA^{TM} it has to follow the following properties:

1. The Enterprise Model of ANSA provides a framework for the role of information processing. It could also play an important role in the Data layer and Processing layer of OSCA^{TM} for the transfer of data efficiently through the Communication Software Fabric. When it is integrated, it should have the capability of describing the overall objectives of the complete system in terms of the actions, goals and policies that take place during operation within the BCC environment.

   The role that ANSA's Information Model plays when it is integrated with OSCA^{TM} is to provide a framework that describes the information requirements of the three layers of OSCA^{TM} - the data, the user and the processing layers.

   The Computation Model of ANSA concentrates on the problems and opportunities that are present on many loosely coupled computer systems. It provides the programming language features for the Communication Software Fabric.

   As both ANSA and OSCA^{TM} are multiple-purpose system architectures, it is not viable and possible to impose a single and a particular programming language. The other important aspect is a program, application or data of the integrated system should be capable of being ported on to any machine using the SunOs, HPUX, ULTRIX, VMS or MSDOS, to keep this integrated system running. The trader of ANSA acts as a directory and management facility for distributed application. The configuration manager which is also a distributed application component provides means to instantiate applications components above the ANSA platform [ANSA89]. It could be recommended that ANSA could be integrated with OSCA^{TM} to play the role of Communication Software Fabric.

As OSCA^{TM} maintains company or corporate data, it is necessary that a user of the system should be able to access data that only pertains to him. Different users of the complete system of OSCA^{TM} and ANSA could write a distributed program in a high level programming language for the various building blocks of OSCA^{TM}'s corporate data layer. ANSA would provide various forms of distribution transparency that support access, location and concurrency transparency to maintain secrecy of the Corporate Data.

3.6 Summary

The total integrated functionality is allocated to data, processing and user layers of OSCA^{TM} and the data is either local or corporate. In the distributed environment, data entities with global importance within the BCC environment have to be decentralized in different data layer blocks. To guarantee data consistency and to provide access flexibility, these data blocks act very much like abstract data types, protecting the data and granting access only through standard set of actions. Thus when ANSA is integrated with OSCA^{TM} it should be able to interoperate as a peer within OSCA^{TM}'s interoperable architecture.

Enabling technologies are not only identified but are also classified with respect to an integration architecture with regards to basic elements of technology. These basic elements of technology could be hardware systems, operating systems, basic networking etc.

The concepts of semantic integrity concepts and the application domain have been discussed with respect to OSCA^{TM} and ANSA. As we have seen earlier the communication between the building blocks of OSCA^{TM} is limited through the contracts. These contracts are handled by the communication software fabric which provides transparent access to the building blocks of OSCA^{TM}. ANSA, when integrated should be able to do this job, when it functions as the communications software fabric of OSCA^{TM}.

John Mills suggests in his paper [Mills 90], that the drivers of distributed functions tend to move the architectures towards separation of concerns and accommodation of the OSCA^{TM} building block principles. The distributed architectures must be concerned about location transparency and standardized messages. This fact of location transparency is given by ANSA which provides for distribution transparency. Standardization of messages in ANSA is the first step in moving towards contracts.

Thus we can see that if OSCA^{TM} and ANSA are integrated, the applications of OSCA^{TM} might raise the level of interoperability enhancing the Bellcore Client Company's ability to offer advance services efficiently and rapidly.

Chapter 4

References

(a) [ANSA89] ANSA: An Engineer's Introduction to the Architecture. Release [TR.03.02], November 1989. M/s Architecture Projects Ltd., Poseidon House, Castle Park, Cambridge, U.K.

(b) [ARM89] The ANSA Reference Manual. Architecture Projects Management Ltd., Poseidon House, Castle Park, Cambridge, U.K.

(c) [EISN 90] H. Eisner, "The New Process of Archetyping for Large Scale Systems Integration", submitted to the Journal of System Integration, November 1990.

(d) [Mills91] John A. Mills, Lecture delivered at NJIT on 4/17/91 on OSCA^{TM}.

(e) Mills90] John A. Mills, "An OSCA^{TM} Architecture Characterization of Network Functionality and Data". Bellcore, 444 Hoes Lane, CN - 1300, Piscataway, NJ 08854-4182, August 27, 1990.

(f) [ODPSE] Project Reference to the Open Distributed Processing Support Environment, ALCATEL/ELIN Research Center, Vienna, Austria. Annexure to ANSA: An Engineer's Introduction to the Architecture. Release [TR.03.02], November 1989. M/s Architecture Projects Ltd., Poseidon House, Castle Park, Cambridge, U.K.

(g) [ROSSAK91] Wilhelm Rossak, " Some Thoughts on Systems Integration: A Conceptual Framework", Journal On System Integration, Vol I/1, 1991.

(h) [OSCA90] The Bellcore OSCA^{TM} Architecture; Technical Advisory [TA-STS-000915]; Issue 2; 2/27/90.

(i) [CLOUDS91] Partha Dasgupta and Richard J. LeBlance, "The Clouds Distributed Operating System", Computer, November 1991.

NOTE:

The figures shown in Chapter1 and 2 of this thesis are taken from [OSCA2:90] and [ANSA89].

OSCA^{TM} is the registered trademark of Bellcore and ANSA "**Advanced Networked Systems Architecture**" is the registered trademark of APM Ltd.

Chapter 5

Glossary

(a) **Building Blocks or BB's:** A set of computer programs, data schemas and other related software that have interfaces with and whose functionality is independently releasable and installable. These executable software components adhere to OSCA^{TM} architecture's separation of concerns and building block principles.

(b) **DLBB**: Data Layer Building Block stewards the Corporate Data and allows data access by other building blocks.

(c) **PLBB**: Processing Layer Building Block. A building block in the Processing Layer.

(d) **ULBB**: User Layer Building Block in the User Layer.

(e) **Steward**: The Data Layer Building Block responsible for some set of Corporate Data.

(f) **DBMS**: Database Management System.

(g) **ODPSE**: Open Distributed Processing For Support Environment.

(h) **ANSA**: Advanced Networked Systems Architecture.

(i) **APM Ltd.**: Architecture Projects Management Ltd.

(j) **BCC**: Bellcore Client Companies.