Fall 1992

A comparison of integration architectures

Amar Mahidadia
New Jersey Institute of Technology

Copyright Warning & Restrictions

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be "used for any purpose other than private study, scholarship, or research." If a, user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of "fair use" that user may be liable for copyright infringement,

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation

Printing note: If you do not wish to print this page, then select "Pages from: first page # to: last page #" on the print dialog screen

ABSTRACT

A Comparison of Integration Architectures

by
Amar Mahidadia

This paper presents GenSIF, a Generic Systems Integration Framework. GenSIF features a pre-planned development process on a domain-wide basis and facilitates system integration and project coordination for very large, complex and distributed systems. Domain analysis, integration architecture design and infrastructure design are identified as the three main components of GenSIF. In the next step we map Bellcore's OSCA interoperability architecture, ANSA, IBM's SAA and Bull's DCM into GenSIF. Using the GenSIF concepts we compare each of these architectures.

GenSIF serves as a general framework to evaluate and position specific architecture. The OSCA architecture is used to discuss the impact of vendor architectures on application development.

All opinions expressed in this paper, especially with regard to the OSCA architecture, are the opinions of the author and do not necessarily reflect the point of view of any of the mentioned companies.

A COMPARISON OF INTEGRATION ARCHITECTURES

by
Amar Mahidadia

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

Department of Computer and Information Science

January, 1993

APPROVAL PAGE

A COMPARISON OF INTEGRATION ARCHITECTURES

by
Amar Mahidadia

Dr. Wilhelm Rossak, Thesis Adviser
Assistant Professor of Computer and Information Science, NJIT

Dr. Peter Ng, Committee Member
Chairperson and Professor of Computer and Information Science, NJIT

BIOGRAPHICAL SKETCH

Author: Amar Mahidadia

Degree: Master of Science in Computer and Information Science

Date: January, 1993

Undergraduate and Graduate Education:

- Master of Science in Computer and Information Science, New Jersey Institute of Technology, Newark, New Jersey, 1993
- Bachelor of Science in Chemical Engineering, The Gujarat University India, 1983

Major: Computer and Information Science.

This thesis is dedicated to
my parents and Dr. Wilhelm Rossak

ACKNOWLEDGMENT

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

Special thanks to Dr. Peter Ng, for serving as members of the committee. The author is grateful to Mr. John Mills, for his timely suggestions and comments for this work. The author appreciates the cooperation of the systems integration research group under Dr.Wilhelm Rossak at NJIT. Special thanks to Sunitha Reddy for helping on ANSA chapter and providing review and text.

And finally, a thank you to Dr. Bill Huang, Rikesh Dave, Anirban Sharma and Ketan Adhvaryu for thier help and to my wife, Smruthi.

TABLE OF CONTENTS

LIST OF FIGURES

Figures **Page**

CHAPTER 1

1.0 Introduction

Computer science and modern technology have provided services to users in very diverse application areas of research, development and industry. System design methods and tools normally focus on a particular problem and the resulting system functions according to a specification targeted to a specific project with definite boundaries. The evolution of commercial information systems has reached a stage where the productivity and availability of the electronic assets of the enterprise rivals the importance of the mainstream business activity. The efficient management of the commercial information system is now one of the organization's highest priorities. Administrators need to manage three often conflicting requirements. Firstly, they must reduce system resource costs. Secondly, they must reduce complexity and maximize availability, reliability and flexibility. Thirdly and of major importance, they must also ensure the appropriate level of security.

Considering the immature nature of the field, the first task of an administrator is to design a process and its supporting elements before tool support and distinct applications. As soon as the process is defined and implemented within the supporting environment, the product engineer can handle the development of applications in an integrated manner. Presently, there is no common, standardized process model or reference process architecture for systems integration which can be seen as a basic or generic model in this field.

We suggest GenSIF, a Generic Systems Integration Framework, as a first step

in this direction. GenSIF supports a pre-planned development process on a domain-wide basis and facilitates system integration and project coordination.

The purpose of this paper is to relate GenSIF to the OSCA architecture, Bellcore's interoperability architecture, and to compare other architectures to this framework.

CHAPTER 2

GENSIF - THE GENERIC SYSTEMS INTEGRATION FRAMEWORK

2.1 The GenSIF Perspective

Over the past years, systems development has become increasingly demanding. We have to recognize the fact that current methodologies and development strategies are unable to deal efficiently with very large, distributed systems. These systems are built by assorted contractors at different times as relatively independent projects, but still are expected to work in an integrated manner.

In Mittermeir (11), this problem has been characterized by naming a set of critical factors which are not supported by traditional technology and the available state-of-the-art process models. These critical factors can be summarized as follows:

Development support goes beyond software support.

More than a single client is involved.

More than a single producer is involved.

More than a single project is involved.

Most of the existing system development methodologies can adapt only insufficiently to these challenges and focus mainly on the development of one system for one specific problem, instead of providing means to develop an integrated group of systems which provides a coordinated domain-wide solution. The result is that we try to develop huge systems to cover the requirements of an ever growing problem space, a situation similar to the one when large monolithic "spaghetti" programs developed to cope with upgraded user requirements. In analogy to the concepts of structured system

development suggested at that point in time, many researchers and practitioners suggested taking a step beyond the "huge system approach" as practiced today. The idea is to treat the application domain as a unified problem space, but to decompose the solution into highly independent systems which are integrated to form a comprehensive solution.

Based on these considerations, we propose the development of so called mega-systems to cover the needs of application domains in a structured way. We consider an application domain to be a comprehensive, internally coherent, relatively self-contained area or business enterprise supported by software systems Zemel (28). A mega-system covers at least a major part of an application domain and is defined as a class of software systems with at least one of the characteristics of being constructed from more than one system, by more than one developer and for more than one customer. Mega-systems include huge systems, generic systems, package systems and systems of systems Zemel (26).

In this report we will focus on systems of systems. Eisner, Marciniak, and McMillan Eisner (4) define a system of systems as a set of several independently acquired systems, each under a nominal systems engineering process; these systems are interdependent and form in their combined operation a multi-functional solution to an overall coherent mission. The optimization of each system does not guarantee the optimization of the overall system of systems.

To guarantee the integration, optimization and consistent user semantics of the (final) product, coordination and control of the autonomously running projects is

mandatory Eisner (4), Mittermeir (11), Rossak (19). Going beyond software engineering, we want to reach this level of systems engineering by pre-planning and organizing the integration process. This includes the task to integrate applications into a larger system framework like GenSIF (Generic Systems Integration Framework) Rossak (19) and to use interoperability architectures, like the Bellcore OSCA[1] architecture OSCA (14).

The basic suggestion is to introduce a conceptual integration architecture as a level of control and management above the independent projects. The integration management of all the projects is handled on the basis of this architecture which also supports long-term strategies and standards. The necessary environmental infrastructure such as networking, data storage, operating systems, hardware, etc. are supported on the tool level. The establishment of this meta-level of thinking and coordination also recognizes the need to analyze and specify the application domain which encompasses the application environment of the integrated system Prieto-Diaz (16), Zemel (27) and to derive a fitting integration architecture from the domain model Rossak (18).

2.2. The Two Levels of System Development in GenSIF

GenSIF is oriented towards a solution for a specific application domain. Systems development is project oriented. The framework strives to leave the project teams as much freedom as possible but provides domain wide integration measures for project coordination. Each project can pursue its own organization and development effort as

[1] OSCA is a trademark of Bellcore.

long as it complies to the integration framework that is expressed in different models and standards, which guarantee the interoperability of the developed systems.

The focus is on planned-for integration Zemel (28) which prepares systems for later integration without assuming to know all (future) parts of the mega-system nor all possible requirements in the domain. This is opposed to other approaches like post-facto integration Power (15) which strives to resolve problems in systems integration of uncoordinated, non interoperable systems and complete pre-facto integration which assumes to be able to plan everything in advance and, thus, relies on complete requirements and fully detailed system design. Variations of post-facto integration are seen as a part of GenSIF in resolving the problem of legacy systems in the application domain but are not further discussed in this paper.

In GenSIF structured requirements analysis on a domain wide, even though possibly incomplete basis and a mega-system design framework play a major role. The focus is much more on these high level concepts and much less on the technologies and tools which finally implement the concepts. Our goal is to be concept driven and to avoid the problems of an uncoordinated technology driven approach. GenSIF deals with technologies and tools from the basis of a firm set of domain wide conceptual models which allow one to evaluate new opportunities provided by the fast pace of technology development and to pick the best and compatible technical solution on the basis of explicitly spelled out prerequisites and decisions. Figure 2.1 illustrates this concept by relating system development on the project level to mega-system development in GenSIF. This model of a two-level development approach is the backbone of the

presented process model and discussed in more detail later on. At present we want to focus only on the similarities between project and mega-system level to give an intuitive introduction to the concept.

While projects look only at their limited problem space inside the domain, the mega-system level takes the whole application domain into account. The requirements for a project is a domain model for a mega-system. Single systems are built according to a specialized design; mega-systems define an integration architecture, i.e. a framework to interrelate systems to form a mega-system and a set of rules to make sure that these systems are developed in a compatible way. Implementation of systems is centered around local environments while a solution for a domain must provide a standardized infrastructure as a tool platform which accommodates and standardizes every project on the tool level (20).

The elements on the mega-system level coordinate the development on the project/system level. This means standardization of certain tools and methodologies as well as limitation of choices for projects. However, all elements of the mega-system level are kept generic and general enough to avoid the problems of a pre-facto integration approach and are to be seen rather as a framework that helps projects to find their way in the domain, to position themselves relative to other projects, to plan for interoperability with other systems, and to take a common position towards heterogeneous technical environments.

On the mega-system level, the domain model is the basis to derive the integration architecture. The architecture is then used to decide on a useful

infrastructure. So far a kind of "top-down" approach has been used and the picture changes once the framework is established. Domain modeling, integration architecture design, and infrastructure design remain active during the lifetime of the application domain and the related mega-systems. By steadily adapting to a changing environment and by reacting to changes in the domain and in the mega-system itself, they keep the framework up-to-date.

The following section describes the process model from an activity and flow oriented point of view.

2.3 The Process Model

The process model, as presented here, is a simplified version of what is defined in Zemel (28). It includes a discussion of the most important tasks involved in the development framework and has been brought into a compacted format regarding data and control flows. We limit the presentation to the identification of tasks and subtasks and the communication of results/documents between tasks; we do not discuss related organizational issues regarding staffing, team structure, documentation standards, etc., nor do we go beyond the engineering point of view to cover social, legal, or other aspects.

The method we use to describe the structure of the process model is similar to SADT Ross (17). The main element are tasks. A task is defined as an activity with a specific objective and schedule. Tasks are connected by flows of data and/or control. Several tasks performed to achieve a particular purpose constitute a process. A complex

task may be decomposed into several sub-tasks. However, when a task delivers a result, this does not necessarily mean that it ends its activity. Thus, our model does not define steps or phases which execute for a limited time and in a highly specific sequence, but rather a network of interdependent and steadily active tasks which are interrelated by a coordinated exchange of information.

In our diagrams tasks are shown as boxes. A task which runs in several copies is drawn as a set of overlapping boxes (a box with "shadows"). Scheduling of tasks is achieved by a special type of task, so called control tasks. While simple tasks are shown as boxes with solid lines, control tasks are shown as boxes with dashed lines. Flows entering a box from the left side are interpreted as inputs. Flows leaving a box on the right side are outputs/results. Execution mechanisms supporting tasks in their activities are shown as flows going to the bottom of a box. Unconsumed reference information, not used as a direct input for processing, is depicted as a flow to the top of a box. Inputs, outputs, execution mechanisms and references are shown as solid lines. Control/scheduling information from a control task is shown as a dashed line connected to the top of a task box.

2.3.1 An Overview of the Process Model

The basic principle of systems integration in GenSIF is to upgrade the currently used software engineering life cycle models to overcome the problems encountered during the development of large and complex systems. The major point is not to revolutionize the way systems are developed but rather to lead to a gentle evolution of concepts,

preserving knowledge and tools which exist for traditional development processes and environments, while introducing at the same time an additional level of reasoning and control above the single project. Figure 2.2 describes a basic process model for such an integrated development effort.

In a development process for a mega-system, e.g. a system of systems, multiple system tasks (projects) can be active at the same point in time. System tasks represent the traditional system development process, transforming requirements into user-specific systems. To support the single projects and to guarantee an integrated result (a system of systems and not a set of unrelated systems), a domain model, a conceptual integration architecture and an infrastructure are used. All projects are scheduled and controlled by a meta-management task activity.

The domain model allows the project team to handle requirements and to communicate knowledge about the domain on the basis of a standardized common model. The conceptual integration architecture provides a design reference model used to guide projects during their internal design activities. The technical infrastructure supports the implementation phase of the project by providing a standardized set of tools and technologies which are shared by all projects.

Integration architecture, domain model, and infrastructure are specified and maintained by the mega-system tasks. These mega-system tasks complement the system tasks by establishing an additional level of coordination in an application domain. The goal is to keep projects as independent as possible and to specify only the necessary additional information models and flows to enable an integrated approach where a

synthesis activity can derive the mega-system from the set of systems produced by the projects.

The following sections discuss the tasks and information elements of the proposed process model in more depth.

2.3.2 Systems Tasks (Projects)

System tasks represent a rather traditional system development process (a project) that transforms problem specific, local requirements into a self-contained, user-specific system (Figure 2.3). In our framework projects form the lower level of system development activities in the process model (see Figure 2.1).

Taking into account the nearly endless list of publications and possible solutions in the area of software development methodologies and life cycles Davis (3), we limit ourselves to the rather abstract and general point of view that requirements analysis, system design, and implementation are the essential ingredients for a development model on this level. In whatever variation these elements are specialized to finally provide a complete project model is not our major concern in this presentation, as long as the minimal process elements, as identified above, are present. We are currently including four major categories of models: the waterfall model Rzoyce (22), Boehm (31), incremental development Boehm (34), the spiral model Boehm (35), and structured prototyping efforts Gomaa (5).

The main input for a project are, as usual, the user requirements. The notion of local user requirements, as used in this paper, refers to the needs and problems a

specific user group in an application domain wants to solve. However, we see these user requirements only as a part of what constitutes the requirements and needs of the whole application domain.

The main output of a project is a system. This system is the result of system development in the project and represents the solution for a self-contained problem area in the application domain. A system is only a part of the integrated solution we want to develop and has to be integrated with other systems to form a mega-system for the application domain.

Based on this concept, additional information is necessary to guide the development process in each project to provide the means for a smooth integration of systems into a mega-system. To achieve this goal, the elements of the mega-system level, i.e. domain model, integration architecture, and infrastructure, are utilized (shown in Figure 2.3 as flows with names written in italics).

2.3.3 The Role of Domain Model, Integration Architecture, and Infrastructure

A comprehensive domain model is used as additional information source during requirements modeling. This helps to standardize the perception of the application domain and allows the project's requirements to be engineered to derive project specific requirements within and from the framework that the domain model provides Zemel (27).

In the GenSIF, the domain model is the output of a domain analysis step in the mega-system tasks activity (Figure 2.2). It describes all phenomena regarding a specific

application domain on a conceptual level from different points of view Zemel (27), as given by the roles established in the user population, and is organized according to various aspects of modeling, e.g., static, dynamic, logical, legal, etc. This type of domain modeling is based on the use of ontological principles for conceptual modeling Wimmer (25) and strives to structure the domain along a set of user-definable criteria. See Zemel (27) and Zemel (28) for a more detailed discussion of the domain modeling approach in GenSIF.

The conceptual integration architecture is also derived in the mega-system tasks activity. An integration architecture influences system development in each project by providing a general design philosophy for the application domain, as suggested in Lawson (7). This philosophy is described in a reference model, defining high level design concepts, architecture primitives, and guidelines for system development Rossak (18), Rossak (20). By standardizing selected elements of system design in the domain, an integration architecture makes sure that systems developed with the GenSIF process model comply to the framework of the mega-system architecture and can, thus, be easier and in a meaningful way be integrated into the mega-system structure. To a certain extend, an integration architecture can be seen as the result of a constructive domain analysis step, as described in different variations in most reuse environments Prieto-Diaz (16) and for most programming-in-the-large concepts, e.g. Tracz (24).

It should be pointed out that the use of an integration architecture as a reference model does not predefine in advance the methods and approaches used in each project. Projects still can decide to run a locally optimized effort, as long as the results - the

systems developed in the projects - finally comply to the rules of the reference model and, therefore, to the design philosophy of the application domain. Such an approach allows the designer/engineer to synthesize mega-systems from singular systems in a highly flexible way, e.g. utilizing a systems of systems architecture Rossak (19), Zemel (28). An example for a successful implementation of this concept are application machines as described by Lawson (8), Lawson (9).

The third element of integration in the domain, the technical infrastructure, is a standardized development and execution platform. The infrastructure provides the so called enabling technologies Rossak (19) which allow the project team to comply to the application domain's integration architecture in their development process.

Typical elements of such a development/execution infrastructure are communication tools, database components, and user-interface generators which support standardized and transparent system handling in a distributed environment. Examples for infrastructures, or at least a subset or framework, are the ANSA[2] communication package ANSA (1), the reuse oriented RAPID/NM environment by AT&T Beck (30), and the "toaster-model" for an integrated software engineering environment as published by NIST (12).

2.3.4 Mega-System Tasks

Integration architecture, domain model, and infrastructure are specified and maintained by the mega-system tasks activity (see Figure 2.4). Mega-systems tasks form most of

[2] ANSA is a trademark of Architecture Projects Management Ltd.

the proposed mega-system level in our model and are the main element of systems integration in GenSIF. Mega-system tasks complement and extend system tasks by providing the "links", the models and platforms discussed above, which allow one to control and coordinate the projects in the application domain. Derived from these needs profile and mega-system tasks are split into domain analysis, integration architecture design, and infrastructure design; one activity for each required result. Domain analysis is currently specified according to suggestions in Prieto-Diaz (16) and includes

definition of domain boundaries,

identification and classification of information,

representation of the acquired information in a domain model,

evaluation and validation of the model and update and refinement of the model.

Concepts like information modeling and enterprise modeling are included in what we call a domain model. A refined process description for domain analysis in GenSIF to accommodate the above suggested inclusion of view-points, roles, and aspects into the domain model is discussed in Zemel (27).

However, the main steps remain in principle the same. We utilize object-oriented concepts as a basic means of communication and notation in this area. Object-orientedness fits best our needs if we want to stay with existing methodologies, even though we would like to integrate more information regarding the dynamics/activities of the application domain and more knowledge support to model consistency rules and a shallow reasoning capability. One major difference to most other approaches to domain modeling is that we do not want to include a constructive

phase in our modeling process but keep this aspect of domain analysis for the separate integration architecture design task.

Integration architecture design is based on the domain model and derives a conceptual technical design on a domain wide basis. Using existing architectures, e.g. OSCA (14), Lawson (8), and Best (30), we have identified the major elements of integration architectures Rossak (20) as

standards and restrictions for mega-system building blocks,

a model for domain wide transparent communication,

a model for the handling of (distributed) data of domain wide importance, and

standards and restrictions for user-interfaces.

Infrastructure design has the task to support the conceptual architecture derived during the integration architecture design task with a platform of integrated tools. Reference models for integrated software engineering environments, e.g. NIST (12), are a already well developed basis for this activity and show a relatively high degree of similarity to the list of architecture components that have to be supported.

Integration architecture design and infrastructure design are activities which rely heavily on the adaptation and specialization of existing solutions rather than new inventions. Design in these environments should stay as close as possible to existing standards and solutions and adapt them to the needs of the domain to achieve a proper application of their concepts. The first choice is to reuse and not to reinvent.

Another stabilizing element we see in our framework is that infrastructure design is driven by a conceptual architecture which itself is driven by a domain model, and not

vice versa. We recognize the close relationship and the ongoing feedback loop that connects especially architecture design and infrastructure design.

Nevertheless, we suggest looking at integration as a concept driven approach rather than a technology driven patchwork.However, this does not mean that we think of mega-system tasks as a strict sequence of time-limited phases. Even though domain analysis, integration architecture design and infrastructure design imply a certain sequence of steps at start-up time, they can be seen as executing in parallel during the later phases of integrated system development. Mega-system tasks are active as long as systems are developed, maintained, or executed in the application domain. Feedback from projects and other activities (Figure 2.2) provide a steady stream of information regarding the current status of the application domain, active and planned (mega-)systems, design considerations, tools, etc. This feedback is merged with the other inputs of the mega-system tasks and is used to steadily improve and adapt the domain model, the integration architecture, and the infrastructure platform.

2.3.5 Synthesis and Meta-Management Tasks

Even though the systems which are output of projects in the application domain have been developed with systems integration in mind, a separate synthesis task is shown to complement the mega-system tasks. This synthesis activity derives the integrated system from the set of systems produced by the projects. It can be regarded as a kind of linking activity, adding new systems to existing mega-systems and other legacy systems in the domain. Mega-system synthesis also takes care of other mega-system related

activities, e.g., the analysis of timing behavior if a new system is linked to the existing systems in the application domain Rossak (21).

A meta-management task, shown as a control activity, manages and coordinates the other activities in the GenSIF process model. It can be seen as a scaled up version of traditional management tasks and includes scheduling, budgeting, quality assurance and configuration management. Meta-management is directly involved in mega-systems tasks and in mega-system synthesis, coordinating the activities and steps of these tasks in a centralized manner. Systems tasks, however, retain a much higher level of independence by keeping up their own project management (Figure 2.3). In this case, meta-management influences local project management only as much as is necessary to achieve the proper use of the integration framework.

2.4 Integration Architecture and Infrastructure

A conceptual integration architecture and the related infrastructure are the two main elements in GenSIF to coordinate system development. This section gives a more detailed discussion of these concepts and is based on the remarks already made in section 2.3.3. The domain model, even though of equal importance, is not further discussed. We refer to Prieto-Diaz (16) and Zemel (27) for further details.

2.4.1 The Conceptual Integration Architecture Model

The conceptual integration architecture model describes the guidelines and standards of the architecture, linking the model of the application domain with the implementation

oriented concepts of development environments and enabling technologies Rossak (19). These guidelines usually include

standards and guidelines for building blocks,

a general strategy for system decomposition,

standards for interfaces (internal and user),

a communication model, and

a model for handling data storage/access.

The building block is the elementary concept of integration architectures. Building blocks are the components which provide the different functionalities in an application domain. The sum of all installed building blocks, and the relationships between these building blocks, form the integrated system.

A building block acts like a "black-box", offering services via a predefined interface, but hiding implementation details. It can be realized as a subroutine, a main program, a system of programs, a data-capsule, an object, etc. (i.e. any possible executable unit in an environment). The implementation language to program it and the method to develop it are of no concern to the integration architecture, as long as no rule of the conceptual model is violated. Every project is allowed to pursue its own local optimization effort. The important point is that every building block emulates the same type of interface, regardless of its origin, and uses the same method of communication and data access, as defined in the guidelines of the integration architecture.

A project may deliver one building block or a set of building blocks. This depends on the decomposition strategy of the integration architecture. If only one

building block is developed to implement the project, we have a situation similar to traditional approaches, where the result of development in a project is one monolithic unit. To avoid this and to support interaction between units from systems developed in different projects, it is necessary to decompose a system into building blocks. The architecture only restricts the possible layout of the final product in its appearance on the system level.

To connect building blocks in order to form fully functional systems, a communication model must be defined. This model specifies the standardized way one building block sends and receives data or control to another building block. Once again, this communication standard is only of concern at the system level of the application domain, but does not interfere with the internal characteristics of building blocks. So far we see two main strategies on this level: The free communication of standardized messages between building blocks or a centralized approach based on a structure similar to bulletin boards and databases Best (30).

By providing a standardized way of communication, project boundaries become obsolete once a building block is installed. The block can communicate to every other block in the domain and, in return, offers its services on the "free market". The same idea applies to the storage and retrieval of data items, at least to that data which is of global interest.

By standardizing the way data is modeled, accessed and stored, an integration architecture can open the set of global data items to all building blocks. In a traditional environment these data items would have been guarded and hidden by a given

application system. Combining the concepts of decomposition rules, building blocks, and standardized communication, data handling can become just another service offered by a (set of) building blocks.

Besides these more technical issues, a conceptual model for data handling must also deal with the relationship of domain objects and data objects, with the integration of different data models, and with the problems of data formats, e.g. Hsu (6). While the technical issues of storage and retrieval primarily influence the design aspects in the application domain, it is the conceptual data model which preserves consistency and integrity throughout the different projects from a meta-level of reasoning.

The guidelines of the conceptual architecture model should be general enough to provide support for a wide variety of problems, but must also be specific enough to guide system development in the application domain. The problems we face here are very much like the problems we have with standards in general: To find the balance between being obsolete because of over-generalization and being to restrictive because of too much details.

2.4.2 The Technical Infrastructure

The technical infrastructure of an integration architecture is the basis of compliance for projects which follow the rules of the conceptual model. The infrastructure, as the name suggests, provides the necessary standardized services which are an essential ingredient of the architecture. By providing these common system components in a structure similar to an integrated development, execution, and maintenance environment, the

integration architecture makes sure that all projects in the application domain can rely on a standardized implementation platform. The development of these elements is part of the efforts to specify and realize the integration architecture, and not part of individual projects which want to use the architecture.

What ultimately is included in the infrastructure depends to a large extend on the type of architecture the conceptual model describes. Typical elements, useful for most types of architectures are

- a software bus (channel) for communication,
- data storage facilities with standardized interface,
- tools to build a user interface, and
- a system which supports reuse of existing software/hardware.

A software bus facilitates the communication between building blocks. It works like a bus or channel in a hardware architecture, providing an open standard to interconnect the building blocks which act like "boards" and are simply "plugged in". A software bus also gives the system engineer a chance to hide a set of distributed hardware/software platforms underneath the interface of the bus. By providing logical addresses, a building block can communicate with another block via the use of the bus without having to know where the other building block is installed. Changes in networking technology can be masked by this information hiding concept, a factor which helps to maintain upwards compatibility of successive versions of the integrated system. The concept of a software bus is found as an explicitly specified element in many existing architectures, e.g. in Beck (30), Schaefer (23), and is also offered as a

separate tool you can buy off-the-shelf ANSA (1).

For most applications in a business oriented environment, the permanent and structured storage of data is as important as the communication between building blocks. To facilitate an integrated approach, data storage must be provided according to the guidelines of the conceptual architecture model. If the architecture is oriented towards a centralized database system, e.g. Best (30), this component not only provides the handling of data, but also a main element of communication in the system.

In the case of a more distributed approach, the data base has to provide distributed storage facilities and an interface with the software bus. Interfacing the data base with the software bus allows it to implement distributed data storage on the level of the architecture, where storage elements can be added and deleted as necessary, providing specialized services to the rest of the system, like any other building block which provides specialized services. The challenge is to provide a standardized and generalized interface for the data storage elements and to handle the problems of data conversion between different products, platforms and concepts, e.g. by using the software bus to handle these problems.

Such a solution of relatively independent data storage building blocks which are accessible over the software bus would offer a distributed, but still integrated data environment. The distribution is hidden underneath the software bus. Every building block can send a message to ask for service without knowing anything about physical locations or specialized data formats. This solution offers even more flexibility than the distributed database systems which are on the market. Most of those systems hide the

communication from the user (from building blocks) by using an internal communication mechanism which is not accessible to other applications and very often incompatible to an established communication standard like the software bus. For the user and any building block, the database appears essentially like a centralized system and integration of different database products and interfaces is left to the individual project.

Software bus and data storage are the two most important elements in the infrastructure. There is a long list of other necessary elements, as given in Beck (30), Best (30), Eisner (4). However, we want to discuss only two more supporting elements of integration architectures which we see as essential and basic: user interface support and software reuse support.

To include tools for user interface handling in the infrastructure of an integration architecture facilitates unification and integration of the external appearance, as well as the functional structure of different applications. Standardized interface solutions for the PC market, e.g. for the Macintosh[3] line from Apple, have proven this concept. For system designers who work on a project, a predefined set of interface tools in the architecture of the application gives then a chance to separate their functional design from concerns about the interface. Issues regarding upwards compatibility and unification, as far as guidelines and standards are concerned, are part of the conceptual integration architecture. Where tools and implementation issues are a concern, they are part of the infrastructure.

---

[3] Macintosh is a trademark of Apple Computers, Inc.

Software reuse is a natural concept in integration architectures and is supported by providing the infrastructure with its freely reusable components. To go beyond that type of reuse and to support modularity and uniformity of building blocks based on the given decomposition-, interface-, communication-, and data handling strategy, tool support for the retrieval and classification/storage of reusable components must be provided Biggerstaff (32). If a reuse tool not only supports the handling of components, but also the specification and usage of an environment similar to the mega-programming concept Tracz (24), it can be used as part of the main framework to specify the integration architecture itself. An example of such a reuse environment which offers generic templates, rule guided instantiation, and component handling, is the Software Base Mittermeir (10).

The components of the infrastructure should be as general as possible to support a wide variety of conceptual architectures. They should be able to run on different hardware and software platforms, must provide a standardized interface, and must be easily updated and adaptable while still maintaining upwards compatibility. Most components of such an infrastructure are based on pieces of enabling technology (hardware and software), but are usually not available off-the-shelf, as an enabling technology would be. Thus, the infrastructure must be developed and maintained as part of a separate project on the meta-level above the application projects.

To summarize our discussion on integration architectures, we can say that the concept of such an architecture is to relate existing technologies and new concepts, to provide a platform for system development and execution in an application domain.

Every architecture describes guidelines and standards for system development in its conceptual model and provides development and run-time support in its infrastructure.

2.5 GenSIF - A Summary

GenSIF strives for coordination on a domain wide basis instead of relying on patching up existing, incompatible systems later on. Based on the concept of a domain wide and planned approach to systems integration, GenSIF proposes an integration and development framework with two levels of activity. The lower level consists of enhanced system development projects. The mega-system level coordinates these otherwise independent projects. It provides the domain related concepts which drive the integration effort and synthesizes mega-systems from single systems according to a domain dependent integration architecture. An infrastructure is used as a standardized platform for tool support.

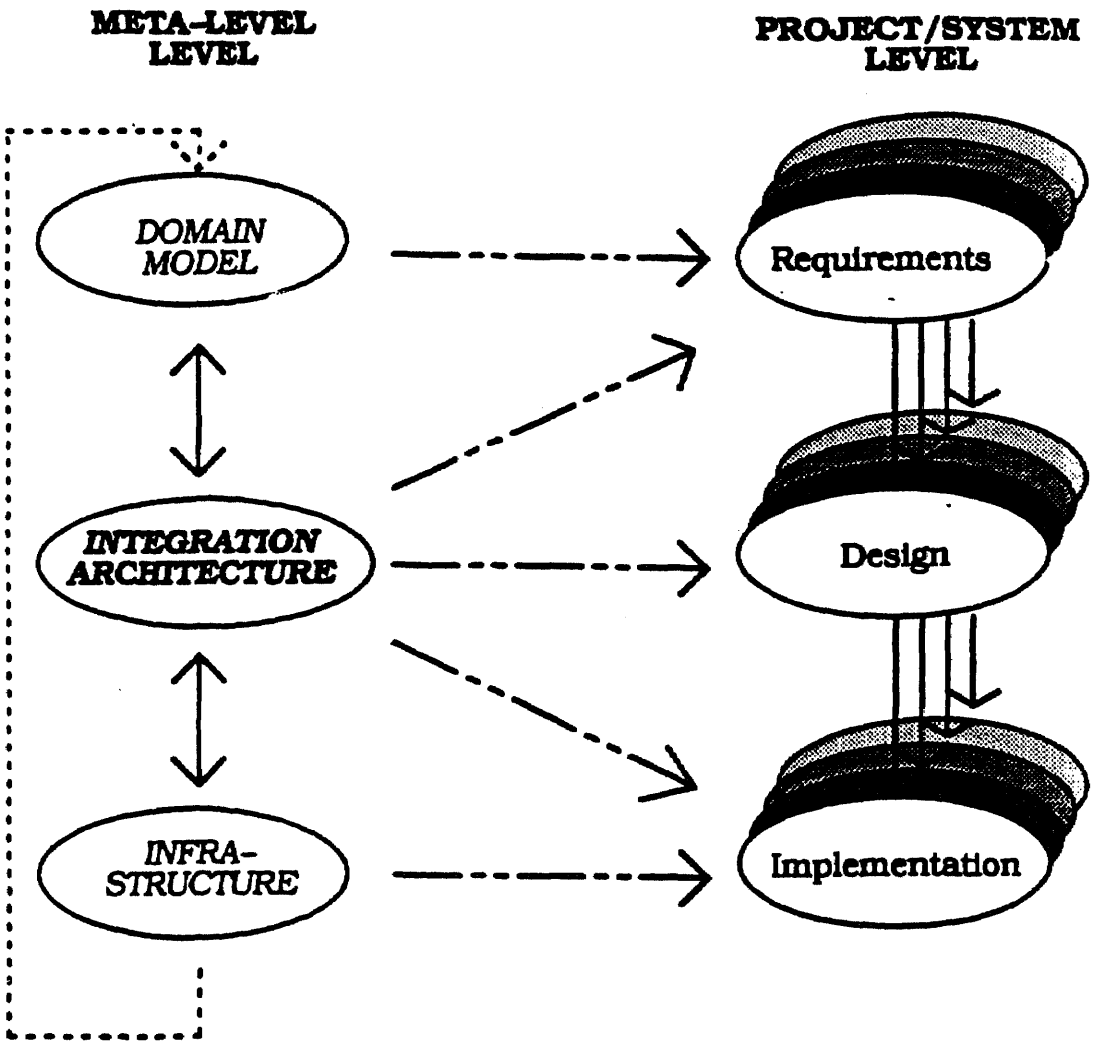

Figure 2.1 Two-Level Concepts of GenSIF

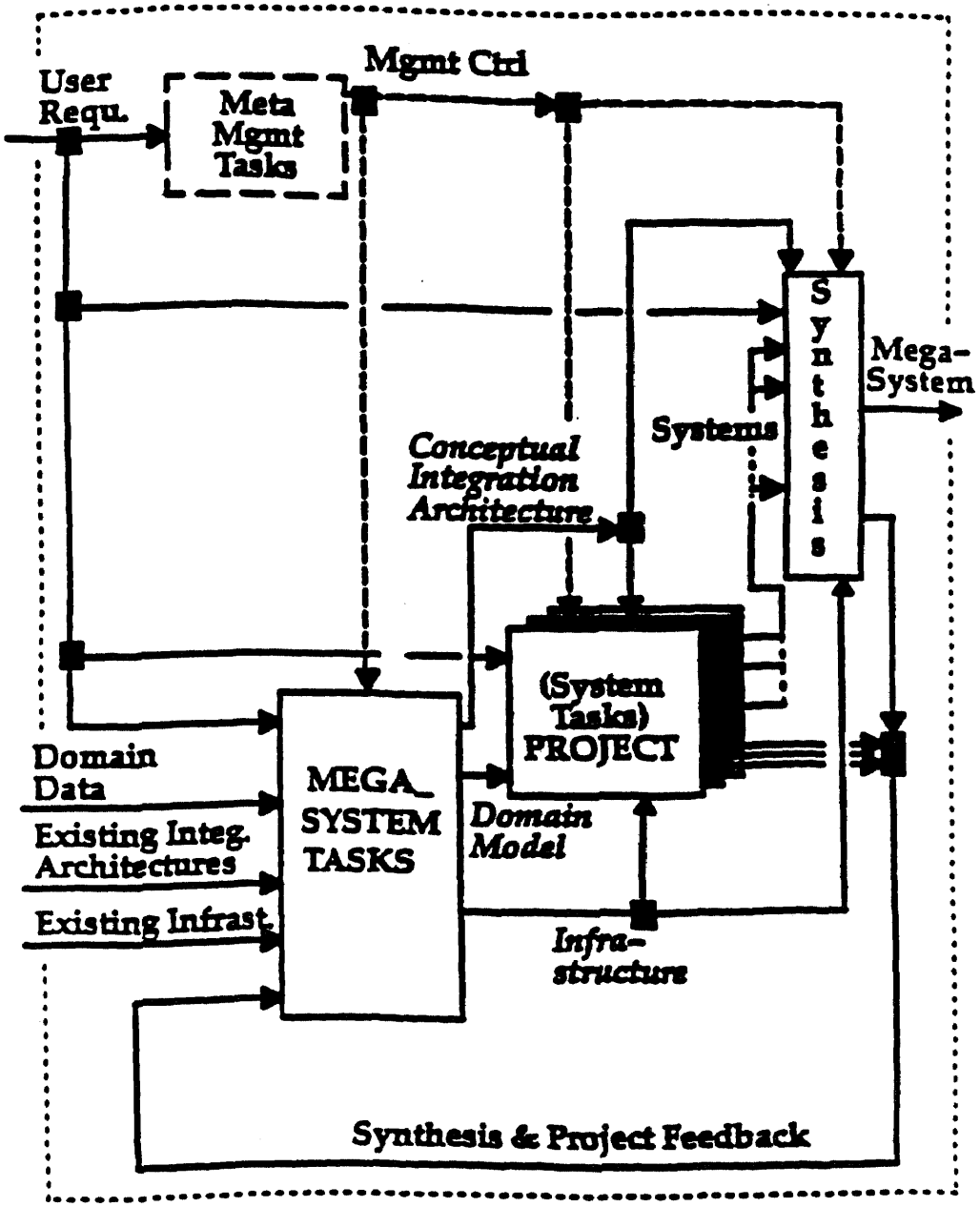

Figure 2.2 GenSIF - A Process Overview

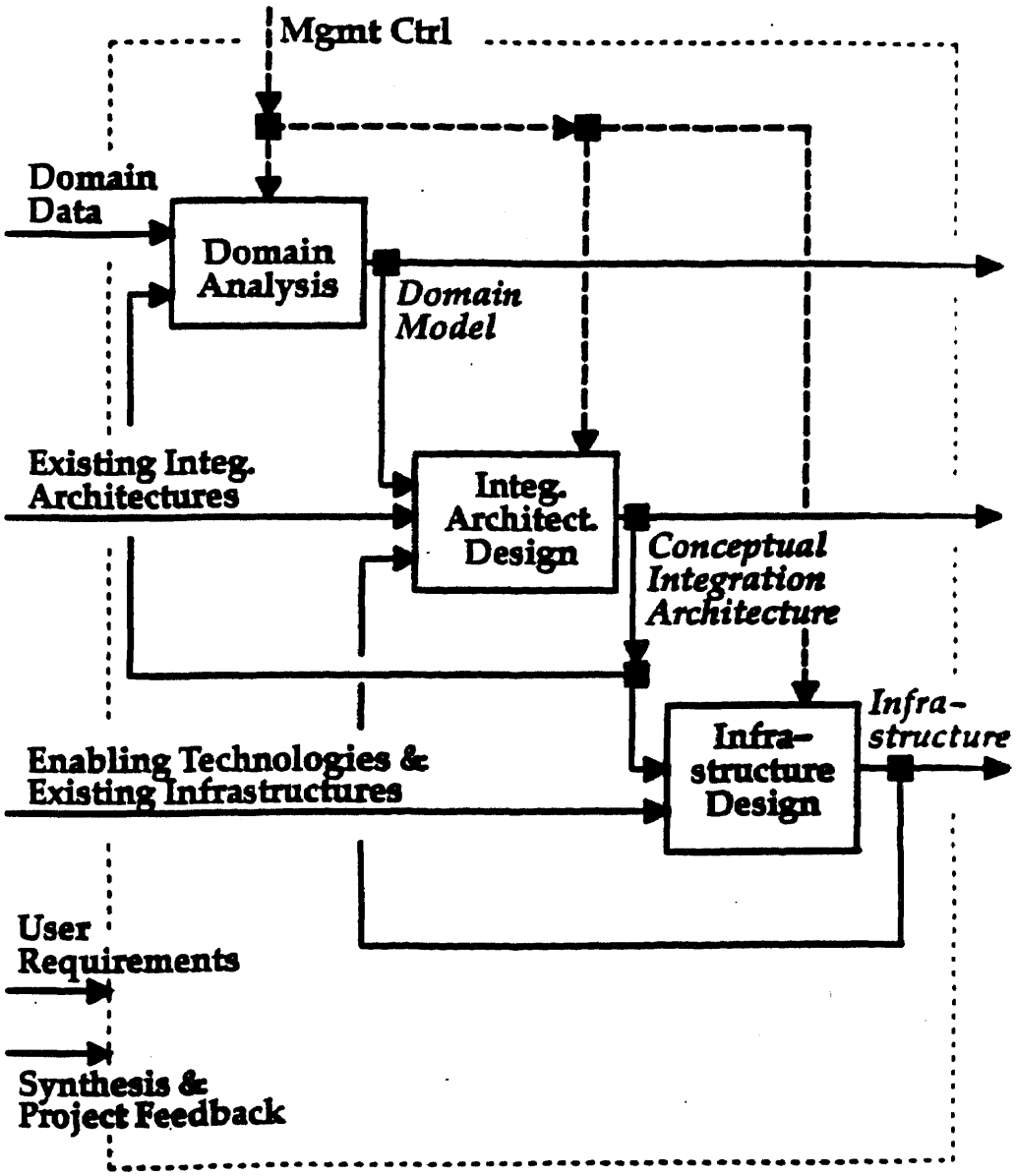

Figure 2.3 GenSIF - System Tasks/ Projects

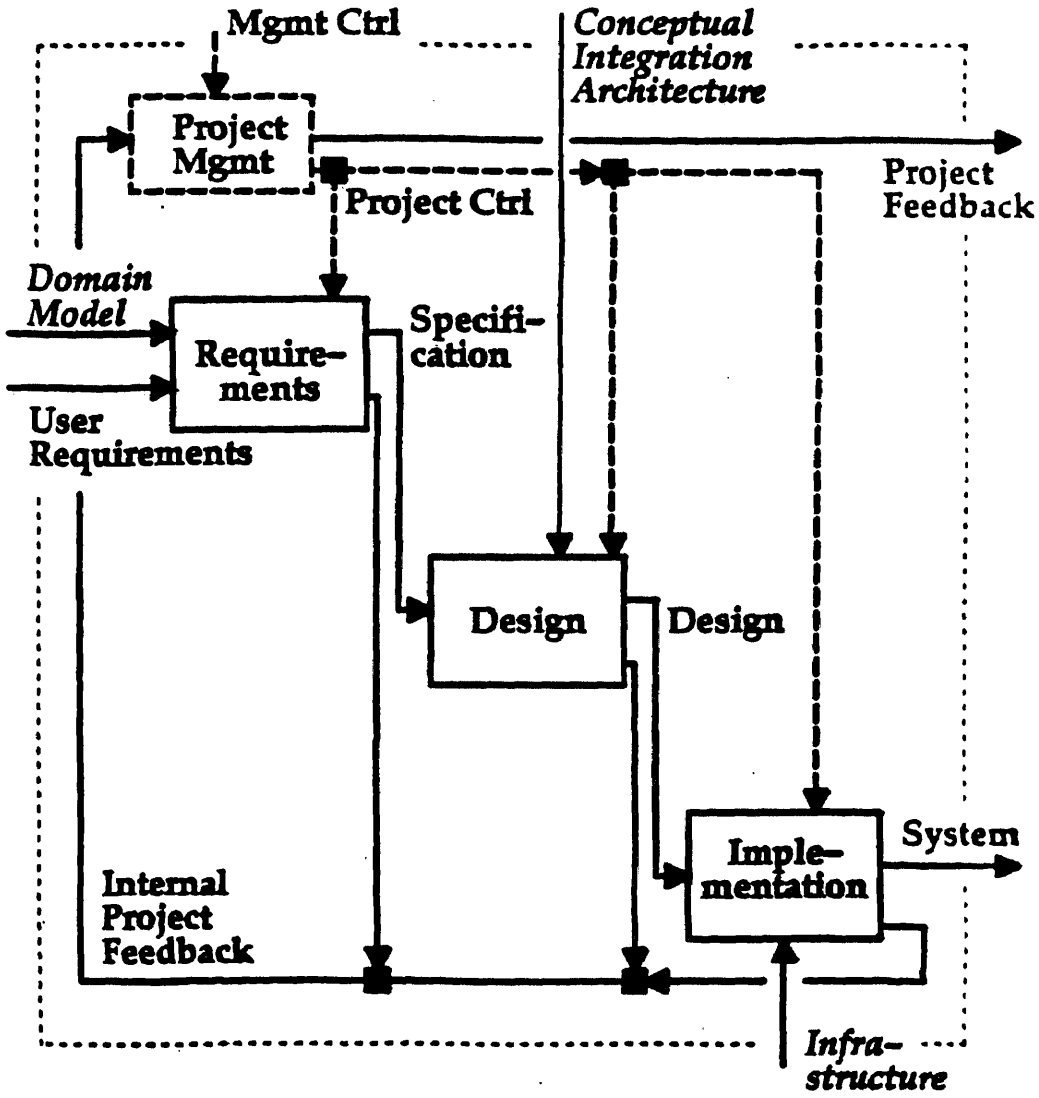

Figure 2.4 GenSIF - Mega-System Task

CHAPTER 3

BELLCORE'S OSCA ARCHITECTURE

3.1 Overview of the OSCA architecture

The OSCA architecture is Bellcore's view of an architecture designed to enable and enhance interoperability among applications. The OSCA architecture describes generic requirements to construct applications that meet the business needs of large-scale integrated, distributed operations. The architecture results in an open, loosely coupled collection of smaller, modular units, called building blocks (the interoperability units) that can be:

1. Configured to best meet the needs of a business;
2. Deployed across a variety of hardware/software platforms based on industry, national and international standards to allow a choice of suppliers;
3. Rapidly evolved, upgraded and improved to meet changing business needs and to take advantage of new technology; and
4. Easily expanded through the addition of new building blocks, either acquired or developed in house, which reuse and build upon existing functionality and corporate data to the maximum extent possible.

The OSCA architecture delineates generic requirements and principles for building blocks that 1) are interoperable and operable; 2) offer open access to all of their functionality to all authorized users and other building blocks; 3) support rapid changes and improvements in functionality by separating concerns between business operations, corporate data management and user interaction processes so that only the affected

building blocks need to be replaced, not the entire product; 4) manage redundant private and corporate data to provide a single consistent view of business information; 5) enable selection and seamless integration of various pieces of software that are developed in-house, reused, or acquired from multiple vendors; and 6) allow deployment of applications across a variety of hardware/software plat-forms by requiring adherence to appropriate industry and/or national/international standards.

Additionally, the OSCA architecture proposes generic requirements for the infrastructure of hardware/software plat-forms to support the interoperability of building blocks. The infrastructure is a platform of business independent functions to enable interoperability and operability; i.e., it is the backplane that allows building blocks to operate individually and to interoperate collectively.

Initial public announcement of the OSCA architecture was in December of 1988 and was updated and reissued in March 1992 OSCA (13). The OSCA architecture is not an architecture of a specific application, but a meta-architecture for applications. Fundamental to the OSCA architecture is the notion of separation of concerns. The OSCA architecture requires that business application functionality be separated (grouped) into "layers" (not to be confused with any OSI layers) or domains: a corporate data layer, a processing layer and a user layer. A layer is the union of all functionality defined as either corporate data functionality, processing functionality, or user functionality. The corporate data layer stewards corporate data and provides application data management functionality to support all create, retrieval, update and delete operations of corporate data in a semantically consistent manner. Semantic consistency

is taken in a broad sense to encompass business-meaningful integrity. The corporate data layer also supports redundancy management and ad-hoc retrieval. The user layer provides functionality to support human users and business application functions that directly interact with human users. The processing layer provides functionality to support business operations and management, such as telecommunications call processing control and non-interactive process control. The software that implements the functionality in these layers is partitioned into building blocks.

Building blocks, which are deployable sets of related application programs, data schemas and other related software under the control of a single transaction manager, are the units of application interoperability (interoperability units) where a high enough level of decoupling can be expected to support interoperability. Individual application programs often are not units of interoperability and require tighter interaction than expected for building blocks. Therefore, building block principles, such as release independence do not necessarily apply to individual programs within a building block.

Building blocks must adhere to specific interoperability principles, namely:

- release independence among building blocks;
- infrastructure and resource independence among building blocks;
- no accessibility assumptions among building blocks;
- execution under the control of a single transaction manager;
- location transparency; and
- the presence of a secure environment.

In addition, interfaces among building blocks must meet certain criteria. Such

interfaces are termed "contracts" in the OSCA architecture. The criteria that must be met are:

- the use of industry and international standards;
- restricted set of syntax encodings;
- isolation from building block internals;
- release independence;
- equality of invocation;
- well-defined interfaces;
- location independence;
- no contract accessibility assumptions;
- recognition of authorized humans and building blocks;
- minimum trust of the invoker;
- maintenance of the invoker's identity; and security audits.

Thus, the separation of concerns in the OSCA architecture requires first that business application functions and business independent functions be distinguished, the latter being assigned to the infrastructure. Then the architecture requires that business application functions be assigned to one of three layers and allocated to building blocks within the layer, such that no building block contains business application functions from more than one layer and that the interfaces offered by the functions of one building block to the functions of other building blocks are well-defined and well-formed (i.e. they are OSCA contracts), so that the functions of one layer are de-coupled from the functions of another layer.

It is not the intent of the architecture to separate data from behavior. The data and processing layers are application functionality layers. The data layer provides all applications to support corporate information. The architecture requires that the data layer includes the behavior of the data as expressed in semantic integrity constraints. The processing layer provides business applications that exercise or control the enterprise's process business rules and that are used by more than one building block. The user layer supports business applications that provide human access and control over the building blocks in the other layers.

3.2 Mapping the OSCA architecture to GenSIF

The OSCA architecture is a logical interoperability architecture. When mapping the OSCA architecture into GenSIF, We first list GenSIF guidelines for each activity of the Mega-system task and then check how these guidelines are covered in the OSCA architecture. We consider three main activities of the process model :

Domain analysis

Integration architecture

The technical infrastructure

3.2.1 Domain analysis

3.2.1.1 *Suggested GenSIF guidelines :*

To define the domain boundary.

To identify and classify the domain information.

To represent acquired information in the domain model.

To evaluate and validate the domain model.

To update and refine domain model.

GenSIF suggests to define the application boundary as a first step for domain modeling. This assigns fixed boundaries to the application domain. The second step is the identification and classification of information. This step is necessary in order to represent the information. The next step is to represent classified information in the domain model. This step is important for a system decomposition strategy. The following two steps are important for evaluation and refinement for existing domain model. The

sequence of above steps suggests a top down approach.

The OSCA architecture : The OSCA architecture does not directly offer any of the above mentioned guidelines. The OSCA architecture is not intended to offer/cover these guidelines. The OSCA architecture is the next step after domain analysis. However, the OSCA architecture refers to an enterprise and information model for domain analysis.

Enterprise model identifies and describes the functions performed by the enterprise, the data created and used by the functions and the interactions between the functions and the data. The OSCA architecture expects that an enterprise view be done to help to promote reuse of concepts, names, definitions, business rules, etc., as well as the reusability of concept, program code and data.

Information modeling is a description of information, data and behavior, independent of how it is stored or used; the corporate information model is the integration of specific application information models.

These two models are not a part of the OSCA architecture. The OSCA architecture expects domain analysis to occur. The OSCA architecture description mentions the need for the Information and Enterprise models of domain analysis to proceed the application of OSCA architecture. The OSCA architecture approach utilizes all these guidelines in engineering systems. The OSCA architecture and its decomposition strategy indicate that the OSCA architecture is designed to be used by a specific domain with a definite boundary. The OSCA architecture works from classified information. This reflects that identification and classification of information in business has been done. The OSCA architecture expects that in general domain information should be classified

as corporate data, rather than different private data.

The OSCA architecture is for the systems engineer, as it is a system architecture, and domain analysis must be done before. The OSCA architecture can be mapped as a middle step between domain analysis and system design. The OSCA architecture is a conceptual architecture and does not offer direct guidelines for domain analysis.

3.2.2 Integration architecture

3.2.2.1 ***Suggested GenSIF guideline :*** To provide standards and guidelines for system building blocks.

The OSCA architecture : The OSCA architecture specifies the principles and guidelines for building blocks. The OSCA architecture utilizes a building block approach to software product\ development in which a system is composed of many building blocks working together. The OSCA architecture suggests that building blocks in the system, possess well define, coherent, business aware functionality and interfaces. Building blocks utilizes hardware and software services from the infrastructure provided by operating systems and the communication networks. The OSCA architecture gives guidelines on how building blocks cooperate and the coupling and cohesion for building blocks. The OSCA architecture clearly lists building block principles.

Building block must adhere to the following principles:

- release independence[4];

[4] Release independence requires that the installation of the new building block or unit of sharable infrastructure or changes to a building block or piece of infrastructure shall be made in such a way that all users of previously-existing, supported functions continue to operate with no loss of function without concurrent installation of or changes to any building block or any piece of infrastructure.

- infrastructure and resource independence;
- no accessibility assumptions between building blocks;
- execution in only one recoverable domain;
- location independence;
- interaction among building blocks are by contracts; and
- secure environment.

3.2.2.2 ***Suggested GenSIF guideline :*** General guidelines for system decomposition.

The OSCA architecture : The OSCA architecture gives guidelines for system decomposition into so-called building blocks(Figure 3.2). The OSCA architecture suggests that business aware functionality be separated (grouped) into *layers* or *categories*. The OSCA architecture requires that system should be decomposed into following layers.

Corporate data layer

Business processing layer

User layer

Each of these layers then are sub-divided into building blocks. Thus the OSCA architecture gives explicit decomposition guidelines.

3.2.2.3 ***Suggested GenSIF guideline*** : Standards for interfaces.

The OSCA architecture : The OSCA architecture gives guidelines for interfaces among building blocks. Interaction among building blocks are done by messages and are defined

by ***contracts***, a specification of a well define set of business aware functionality and commitment by the building block to offer that set of functionality to all other building blocks, in a way which adheres to the ***contracts*** principle. The OSCA architecture gives guidelines and principles for ***contracts***.

Contracts must adhere to the following principles:

- Use of standards;
- Restricted set of syntax encodings;
- Isolation from building block internals;
- Release independence;
- Equality of invocation;
- Well defined interfaces;
- Location independence;
- No contract accessibility assumptions;
- Recognition of authorized humans and building blocks;
- Minimum trust of invoker;
- Maintain Identity of Invoker;
- Maintain Identity of Invoking human and building block; and
- Security audits.

The OSCA architecture provides guidelines for interaction with user. The OSCA architecture suggests to keep user interaction activities separate form all other internal interaction activities. There is a layout in the OSCA architecture about user interaction management.

3.2.2.4 ***Suggested GenSIF guideline :*** A communication model.

The OSCA architecture : The OSCA architecture suggests free and implementation-independent communication of messages in a distributed environment via a predefined infrastructure. The OSCA architecture gives rules and restriction for communications among building blocks.

3.2.2.5 ***Suggested GenSIF guideline :*** A data handling model.

The OSCA architecture suggests a way to handle data across the application domain. The OSCA architecture gives principles for data layer building blocks.

- Separation, a data layer building block must contain only the functionality enumerated in these principles and guarantees support of contracts over internal changes.
- Sufficiency, a data layer building block must provide sufficient and necessary access so that any piece of data that it stewards need only updatable and readable via that stewarding data layer building blocks.
- Openness, a data layer building block must allow ad-hoc retrieval[5] of all corporate data that it stewards for all authorized building block via contracts supporting an implementation-independent information model using an industry accepted standard query language.

---

[5] ad-hoc retrieval : a retrieval formulated using an arbitrary collection of properties of an entity in the information model to determine the set of entity instances of interest to retrieve an arbitrary set of properties of those instances, where properties include attributes of entities connected by an arbitrary chain of linkages in the information model to the entity of interest.

- Semantic integrity[6], a data layer building block must ensure the semantic integrity of the data that it stewards; must maintain well defined consistency requirements between any shared redundant data that it has and the corresponding stewarded data; and must ensure that the semantic integrity of the shared redundant copy is maintained.

- Managing redundancy, a data layer building block must provide means whereby updates to the corporate data that is stewards can be passed to building blocks having redundant copies of that data; and must not propagate updates received from the steward for any of its shared redundant data to other building blocks.Redundant data must be managed according to the following rules:

- If access requirements cannot be met by a single steward, then cooperatively stewarded data or shared redundant data should be used whenever applicable and practical and in preference to private redundant copies.

- The stewarded data is assumed to be the correct data.

- Only updates made to the stewarded data are valid updates.

- Shared redundant copies are obtained only and directly from the steward.

- A shared redundant copy does not propagate updates to other private redundant or shared redundant copies.

- Updates to a shared redundant copy are made only by the steward.

- A private redundant copy requiring automatic updates must be obtained from the steward; otherwise a private redundant copy may be downloaded from a shared redundant copy.

---

[6] Semantic integrity : the fact or quality of data being in a meaningful state, consistent with the semantics defined by the constraints associated with data and their relationships.

- The building block having a redundant copy is responsible for its copy.

Summary : All of the listed guidelines are covered/ offered in the OSCA architecture. This shows, The OSCA architecture exactly fits-in at the conceptual integration architecture level in GenSIF.

3.2.3 The Technical Infrastructure

3.2.3.1 ***Suggested GenSIF guideline :*** Communication tools.

The OSCA architecture : The OSCA architecture does not give any communication tool support straight away. The OSCA architecture discusses only the handling of protocol and messages on the application level. According to the OSCA architecture messages among building block can be sent and received via the so-called 'sharable infrastructure' using 'logical building block addressing'. The sharable infrastructure are the parts of the infrastructure which are sharable among any and all building blocks that may reside in a particular system.

There is a discussion in the OSCA architecture about principles of infrastructure and infrastructure interfaces. The OSCA architecture clearly notes that the details of the structure of so-called ***sharable infrastructure*** is not a part of the OSCA architecture because this is a part of system engineering and not system architecture. Although the modularity and the conceptual design of the architecture clearly shows that the OSCA architecture can easily utilize the channel based communication tools like ANSA or the software bus based communication tool. Other communication approaches such as ***remote procedure call structure*** or reuse- oriented software plateform can also be a suitable in

the OSCA architecture.

3.2.3.2 ***Suggested GenSIF guideline :*** Database Components.

The OSCA architecture : The OSCA architecture suggests a distributed data layer approach at the conceptual level and suggests to interact with database via data layer building blocks. However, there is no direct guideline such as programming languages, hardware components to handle data transaction among distributed databases, standardized interfaces for the data storage etc. This technical infrastructure portion of the GenSIF Mega-system task is not covered in the OSCA architecture.

3.2.3.3 ***Suggested GenSIF guideline :*** User-interface generator.

The OSCA architecture : The OSCA architecture suggests user interaction via user layer building blocks at the conceptual level, but does not give the direct elements required for generating user interfaces. There is a no guideline given for predefine sets of interface tools, programming languages, hardware component other technology required. This engineering model is a ongoing project.

Summary : The OSCA architecture neither offers ready-made infrastructure, nor provides guidelines and design for it.

3.2.4 Conclusion

The OSCA architecture fits at the conceptual integration architecture level in GenSIF. This is a meta-level. The OSCA architecture is not a user's document, it is a systems

architecture. The OSCA architecture does not cover domain analysis directly. However, it uses the outcome of domain analysis, in particular the Enterprise and Information Models. The OSCA architecture offers all the guidelines proposed in the GenSIF conceptual integration architecture. The OSCA architecture lays down principles and a framework to engineer an interoperable and operable infrastructure. However, future detailed engineering of suitable infrastructures is needed.

The GenSIF framework is a system integration process model, which covers systems development and design from two extreme ends of system engineering. One upper end is customer requirements, a most fuzzy and always changing environment. It involves a wide variety of customer needs. For this upper end, ***domain identification and knowledge*** on conceptual level is required. The lower end is the restricted existing enabling technology. This requires guidelines for derivation of suitable existing tools. GenSIF uses a proposed Mega system task to bring this two ends together. The OSCA architecture, as we see, is **a conceptual integration architecture** which fits exactly into GenSIF framework at the conceptual level.

The GenSIF framework is not limited to a conceptual integration architecture. GenSIF comprehensively covers the domain analysis, existing integration architectures, their principles and technical infrastructure to support global integration. The principles of the OSCA architecture matches with fundamentals of integration architecture in GenSIF. GenSIF suggests a layout for integration architecture. The OSCA architecture reflects in this layout.

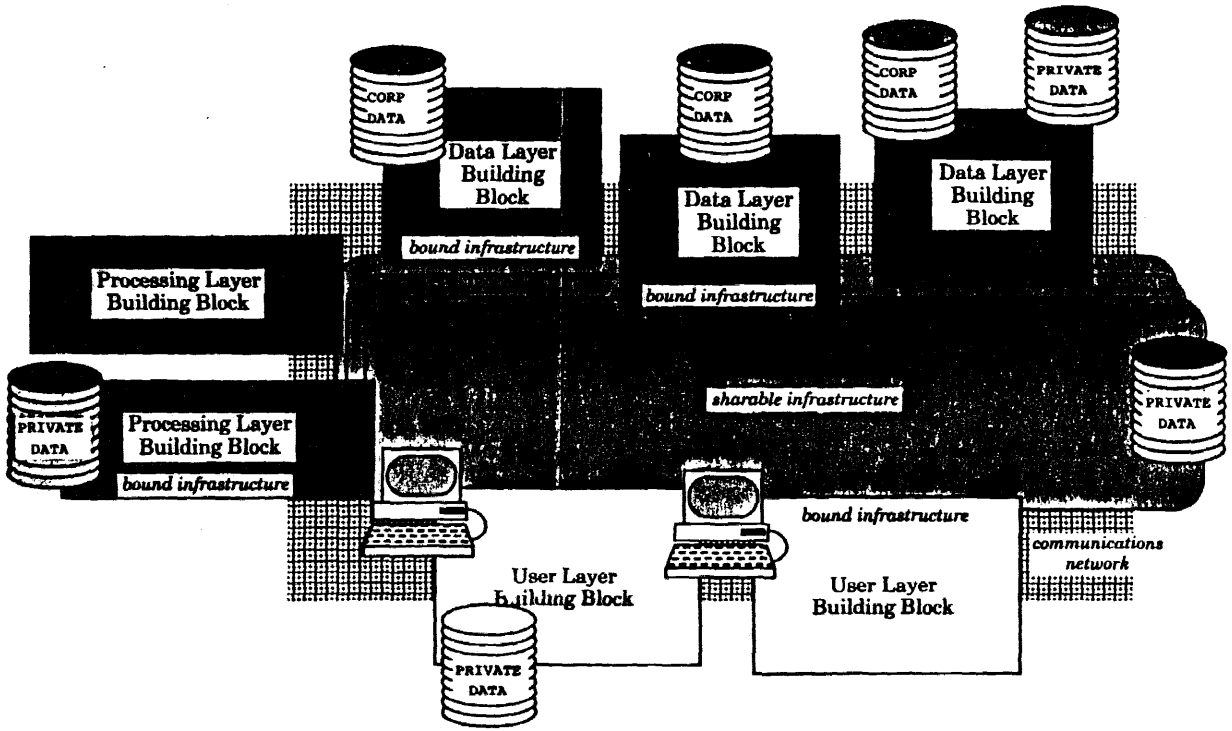

Figure 3.1 The OSCA Architecture Perspective

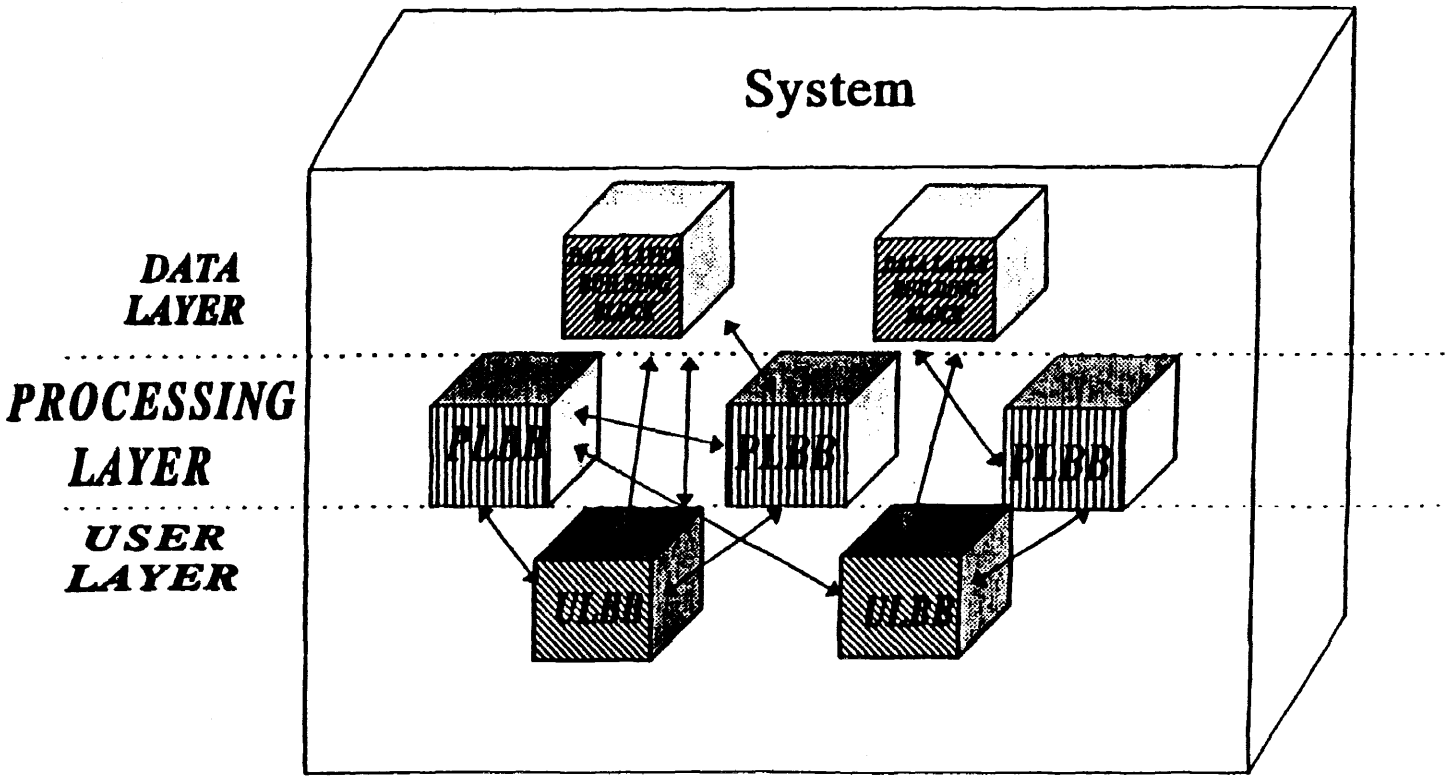

Figure 3.2 The OSCA Architecture systems decomposition startagy

CHAPTER 4

ANSA AND ANSAWARE

4.1 Overview ANSA and ANSAware

ANSA (Advanced Networked System Architecture) is an architecture for Open Distributed Processing. ANSA was sponsored by eight major IT (Information Technology) companies in UK to propose an architecture to build distributed applications and to promote it as a world-wide standard. ANSAware is an implementation of the ANSA architecture.

The objectives for the Distributed System are

- Generic to many fields of application
- Portable across a wide range of operating systems and programming languages.
- Operable in a heterogeneous, multi-vendor environments.
- Maximum opportunity for re-use of existing functionality.
- Provision for interworking between autonomously managed networks.

ANSA supports the design and construction of flexible a distributed applications. It is not constrained by network structure and size, or mixes of different hardware and operating systems and goes beyond distributed operating systems, databases and networking.

4.1.1 ANSAware - An Overview

ANSAware is a suite of software for building Open Distributed Processing systems,

providing a basic platform and software development support in form of program generators and system management applications. ANSAware is supplied as a suite of ANSI-C programs and instructions for installing them on UNIX, MSDOS and VMS.

ANSAware provides a uniform view of a multi-vendor world, allowing system builders to link together distributed components into network wide applications. It is designed to wrap around and link together existing software with minimal changes and overheads (Figure 4.1).

4.1.2 Providing foundations

ANSAware provides the basic functions and extension needed to convert an environment into an ANSA distributed computing system. To maximize portability, ANSI-C and POSIX 1003.1 standards have been followed. ANSAware facilitates interworking between applications running on remote or dissimilar machines. The basis of this portability is software which links into host operating systems and which provides a uniform, technology-independent platform above which application can be run. Distributed applications can be built on this platform and can be run alongside existing programs. In addition to communication, ANSAware provides facilities for parallel processing, synchronization and local software management.

4.1.3 Objects and Interfaces

ANSAware allows applications to be written in an object_based style, using the client/server model of interaction. An ANSAware object is an encapsulation of an

application and its data. An object provides services via interfaces. An object can provide and use more than one interface simultaneously. Several named operations may be provided in each interface which may be used either locally or remotely by client objects.

The object-based approach allows the physical separation of distributed program components to be managed effectively, and allows the containment of system failures. To reduce complexity, transparency mechanisms are available to hide the mechanics of distributing objects. These may be selectively specified by the application writer. Objects are regarded as logically separable. When they are co-located, ANSAware cuts out the networking overheads.

4.1.4 System Management

A suite of system management applications extends the functions of the basic platform. These management applications are themselves built using ANSAware so that users may configure, modify and optimize them according to their needs. These services provide and use interfaces like any other application.

4.1.4.1 Trading

Traders give access to information about available services. Trading matches offers and requests for particular services, using service names, interface types and service properties in combination as selection criteria.

4.1.4.2 Factories

Objects are created by factories in two stages. First a capsule is created, containing object templates. An capsule can then be instructed to make objects from the templates. These objects can then offer their services to the trader or to other objects.

4.1.4.3 Node management

Node managers provide a method for controlling the services available on a network. They use factories and traders to bootstrap and control both static and dynamic services.

4.1.5 Tools

4.1.5.1 IDL

IDL is the interface definition language which specifies the operations available in an interface. All interactions between ANSAware objects are via interface specifications written in IDL, preventing errors and misuse.

4.1.5.2 STUBC

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

4.1.5.3 PREPC

This is the preprocessor which extracts control commands from C programs and translates them into calls to the stub routines prepared by STUBC. Control commands

exist for declaring interfaces, performing trading functions and calling operations in local or remote interfaces.

4.2 ANSAware - the Conceptual Foundation

One of the primary goals for ANSAware is to simplify the design, construction, deployment, and maintenance of distributed applications on the tool-level. Two different styles of distributed application programming can be distinguished:

- imperative - the programmer must call lower-level libraries from the program source at runtime to explicitly drive transparency mechanisms
- declarative - the programmer states distribution requirements in the program source. Tools can then be applied to the source to yield components which meet the stated requirements.

The declarative approach facilitates simplified programming for the user and more comprehensive compile-time checking. Therefore it is the chosen style within ANSAware. A typical distributed application is constructed from several cooperating components. The correct cooperative behavior between pairs of components depends crucially upon a precise definition of the possible interactions between them. ANSAware provides a specification language (IDL) for defining the sets of legal interactions (interface specifications) between pairs of components and a tool (stub compiler) for generating the necessary code (called stubs) which facilitates the interactions when the components are distributed.

The interaction model supported by ANSAware forces all interactions between

components into a remote invocation paradigm.Two forms of remote invocation are supported 1) interrogation, in which the invoking client activity waits for the server to perform the operation and return any result and 2) announcement, in which the invoking client activity does not wait for the server to perform operation.

Once the interface specifications for a distributed application have been defined, components providing and using these interfaces must be coded. Due to the dissimilarity between remote invocation and local procedure calls, an embedded language (PREPC) has been defined for invoking operations and accessing other ANSAware facilities. A preprocessor is used to convert C language program source with embedded PREPC statements into compilable C source files.

The blocking nature of interrogations limits the performance of systems unless components with internal concurrency can be constructed. ANSAware supports concurrency through the provision of a threads package.

ANSAware provides event counters and sequencers as synchronization primitives from which critical regions can be constructed to protect concurrent access to a component's state by its threads.

One or more of the components making up a distributed application may be provided as an independent, general service in the distributed environment. To use such a component, it must be possible to bind to it from each application. To facilitate dynamic binding, ANSAware supports the passing of references to interfaces as arguments to and results from operation invocations. The trader provides a rendezvous mechanism for dynamic binding to well-known or published services.

Many distributed applications depends upon the dynamic instantiation of objects as the application proceeds. ANSAware provides factories for object creation. Some well-known services in the system are considered part of the infrastructure i.e. are part of the system bootstrap. Resource considerations may dictate that some of these well-known services be dynamically instantiated on demand. ANSAware provides a node manager which supports both needs on a per node basis

ANSAware provides a dynamically configurable tracing system for debugging programs which have included the tracing code in the binaries. There is also support for testing encoded assertions at runtime. The user is expected to use the operating system supplied debugging facilities for other forms of debugging.

In order to support the demanding communication requirements of components in a distributed application, the ANSAware infrastructure captures and processes all events which affect a components. Integrated use of other software packages (e.g window systems) in ANSAware components demands a facility for notifying them of events.

4.3 ANSAware - Structure and Concepts

The projections (levels of specifications) that make up ANSA are termed

enterprise,

information,

computation,

engineering and

technology projections

While all of the five viewpoints are relevant to the design of distributed systems, the computation and engineering models are the ones that bear most directly on the use and construction of distributed systems.

ANSAware is an implementation of the ANSA engineering model which is in turn an idealized design for a support environment for the ANSA computational model.

4.3.1 Computational Model

4.3.1.1 Services and objects

The basic element in ANSA is a service. A service is an information handling function - processing, storage or transfer. Components that use a service are called clients. Components that provide a service are called servers. The ANSA computational model permits an object to be both client and server of many services simultaneously. A component or object described purely in terms of the way it provides and uses services is referred to as a computational object.

When clients use the same service, they are said to share the service provided by the servers.A service is provided at an interface. Thus an interface is the unit of service provision. Figure 4.3 shows the relation between server, clients and interface.

Services are divided into application services which are specific to the task to be performed by the system (e.g. a booking service for airline reservations) and architectural services which are generic to a wide range of tasks, and as the name suggests, have been identified by the architecture.

4.3.1.2 Architectural services

Architecture services provide consistent means for such functions as naming and finding services, access control and management within a distributed system. ANSA recognizes that architectural services must exists in order to provide objects with access to the required functionality.

The prime example of such functionality is trading. This allows clients to find servers dynamically via a system-wide directory structure for recording and determining the availability of services. The architectural service which provides trading is implemented in ANSAware as an object called the Trader.

4.3.1.3 Transparencies

A transparency masks a particular aspect of the complexity of distribute programming. ANSA has identified a number of transparencies, which it treats as "selective" in that they may be individually turned on or off as the application demands.

Those transparencies which are implemented by ANSAware are :

- access - allows a uniform style of interaction between objects irrespective of their construction or their environment, or whether the objects are local or remote,
- location - allows interaction with an object without knowledge of its physical location.

Other transparencies to be included are replication, which allows multiple instances of an interface to be treated as one, hiding the details of the organization of the objects supporting this interface group.

4.3.1.4 Interface References

Objects communicate by passing interface references to each other.They are entities which refer to an instance of an interface. Possession of an interface reference by a client allows the invocation of service operations provided at that interface by a server. Interface references (and hence the ability to use services) may be freely passed from one object to another, with or without an intermediary trading service or trader. The process of making a service public by publishing an offer for it to a trader is illustrated in figure 4.4.

4.3.2 Engineering Model

The engineering model is designed to support the computational model over multiple technology models. It is constructed by the mapping of computational model objects into engineering model objects i.e services and the support environment are mapped into components of the engineering model.

4.3.2.1 Node, nucleus and capsules

The term node is typically used to refer to a single computer or workstation. However, it may be applied at different granularities depending upon context.

Examples of possible nodes include:

- a single computer (e.g. a personal desktop computer)
- a heavyweight operating system process or virtual machine.
- a network of computers managed by a distributed operating system.

The resources of a node are managed by an engineering object called the nucleus which assigns them to engineering objects called capsules. The service provided by the nucleus is to take the resources of the node and to build on them to provide a uniform, basic distributed computing environment independent of the underlying operating systems, computer and networks.

A capsule is the unit of autonomous operation within ANSAware. The programmer specifies and builds a number of communicating capsules, each of which represents a separate address space. A nucleus provides capsules with the following capabilities:

- encapsulation, i.e. a capsule is a protection domain and an atomic unit of failure
- provision for concurrent activities, and synchronization and ordering of those activities within the capsule
- communication with other capsules
- persistence of state between interactions (but not necessarily across failures)
- provision for creating further capsules.

4.3.2.2 Computational and engineering objects

A service implementation is a program made up of some number of programmer-defined computational objects.A computational object may have several interfaces, each offering the same or different sets of operations (i.e the same or different services). It is for the application designer to decide whether or not a program exposes internal interfaces

between the computational objects as services for use by other programs.

An activity is a thread of control which may cross from one computational object to another by invoking an operation on an interface at the target object. Computational objects are all potentially remote from one another. During compilation, operation invocation is translated into calls to the local nucleus. A compiled computational objects is called an engineering object.

An engineering object is the smallest unit in ANSAware which may be distributed, activated, passivated and migrated. It is the runtime representation of one or more computational objects. A single engineering object may be composed of many computational objects, bound together at compile time and interacting via local procedure calls.

4.3.2.3 Transparency services

Engineering objects interact with one another through the nucleus. Transparency services are added to a capsule as shown in the figure 4.5. A transparency service manages nucleus-provided resources in a capsule and communicates with its peers in other capsules to provide the services and one transparency service may depend upon another.

4.3.2.4 Protocols

The nucleus specification includes a service definition for the protocol required for communication between nuclei. In addition there is a recipe followed in ANSAware, for implementing the service in terms of three service layers. The top layer, the session

service, provides dialogue and synchronization structures that correspond to the nucleus functions for intercapsule communication. The middle layer provides an execution protocol which is responsible for providing the interaction semantics specified by the computational model. The bottom layer - the message passing service(MPS) - provides a transport services between nuclei. Either connection-orientated or connectionless protocols can be used to implement the message passing service.

4.3.2.5 Nucleus Structure

The resource provided by the nucleus are called tasks, threads, event counts, sequencers, sockets, plugs, channels, and interface references.

4.3.2.5.1 Tasks and threads

A thread is an independent execution path through a sequence of operations within a capsule. From the point of view of the programmer, it represents the unit of serial activity. A thread performs one logical activity within a capsule, but threads can share data structures and can synchronize with each other at significant points.

A task is a virtual processor which provides a thread with the resources it requires. It is a unit of execution for the purpose of allocation of the capsule's processing resources. The number of tasks within a capsule determines the degree of parallelism in the capsule's execution. Once bound to a task, a thread retains that task until the thread terminates. All capsules are multi-threaded and may optionally be multi-tasking.

Where the local operating system only supports one task per capsule,

multi-tasking capsules can be implemented via a coroutine package provided in ANSAware.

4.3.2.5.2 Event counts and sequencers

Event count and sequencers were chosen as the most suitable mechanism for providing efficient synchronization between threads. They are particularly suited for a multi-processor system since they minimize the extent of mutual exclusion necessary to obtain synchronization and ordering guarantees.

4.3.2.5.3 Sockets, plugs and channels

A socket is the unit of addressing for inter-capsule invocations. Sockets are associated with defined interfaces, and thus present a typed view of the services supported by the capsule. All communications are targeted at sockets.

A plug is the access point for the client of an interface. Inter-capsule operations are invoked upon plugs. Each plug is bound to a corresponding socket. The path from plug to socket is known as a channel. A socket data_structure represents the server end of an interface, whereas a plug is associated with the client end.

4.3.2.5.4 Interface references

Some means of identifying service interface instances is required in order to direct operation invocations to the service instance required. The ANSA engineering model defines an interface reference as the data structure for identifying interface instances.

Interface references are created by the binder service. In ANSAware the binder is implemented as a local service in each capsule.

Before any capsule can obtain an interface reference for any external service it wishes to use, it must obtain an interface reference for the trader. This bootstrap problem is solved by furnishing each capsule with a well-known interface reference, viz a reference to a trading service

4.3.2.6 Transforming computational to engineering objects

ANSAware provides two compilers to transform computational objects into engineering objects. The first is concerned with the generation of access transparency services (called stubs), while the second deals with the translation of interactions between client and service objects.

Interfaces to computational objects are specified using the Interface Definition Language (IDL). A compiler, called studc, is provided for automatically generating stub code which provide marshalling and unmarshalling functions for the data types specified by the IDL interface and for dispatching the appropriate service operation on receipt of a operation invocation. The PREPC language provides a means for embedding invocations of interface operations in C source files.

4.3.2.7 Structure of a network of ANSAware nodes

ANSA defines services which provide a network_wide management infrastructure for distributed objects; the following are implemented in version 4.0 of ANSAware:

- the trading service
- the factory service
- the node manager service

4.3.2.7.1 Trader

The trading service contains operations which allow engineering objects to register the service they provide (known as exporting) and to look for services which they intend to use (known as importing).

The ANSAware 4.0 implementation of a trading service (known as the Trader) also makes available other services which allow the system administrator to control the internal organization of the Trader. Services known to the ANSAware Trader are termed offers (i.e. offers by an engineering objects to provide a service).

4.3.2.7.2 Factory

ANSA recognizes the need for the dynamic creation of engineering objects to provide particular services. The Factory service is the means by which this is done. A factory service is provided for creating capsules. Each capsule provides a factory service to allow engineering objects to be instantiated within the capsule. Each engineering objects provides a factory service to allow interface instances to be created and destroyed.

4.3.2.7.3 Node manager service

The node manager provides a means for managing the services available on a particular

node. It allows for the static and dynamic creation of capsules and objects (providing services), and for their subsequent termination. The node manager uses the factory service to create new services, therefore a factory must exist on each node in conjunction with the node manager.

4.4 Mapping ANSAware to GenSIF

4.4.1 Domain Analysis

4.4.1.1 *Suggested GenSIF guidelines*

To define domain boundary.

To identify and classify the domain model.

To represent acquired information in domain model.

To evaluate and validate domain model.

To update and refine domain model.

GenSIF suggest the above guidelines in order to determine the integration architecture and in addressing the other issues of global integration such as semantic integration which involve an analysis of the application domain in order to define a common model of the environment the system is going to serve. It further influences the technological basis that is used to implement computer-based services in an application domain. GenSIF utilizes object-oriented concepts as a basic means of communication and notation in the area. ANSAware does not give any guidelines for domain analysis.

However, it allows applications to be written in an object-based style, using the client/server model of interaction. By using this approach, the ANSA computation model is able to take the scooping and encapsulation mechanism down to the level of simple data structures and data types, if required.

Summary : ANSAware does not offer any guidelines for domain analysis. However, ANSAware is based on object-oriented approach so that it may be straight forward to develop a system according to the given domain model.

4.4.2 Integration Architecture

4.4.2.1 ***Suggested GenSIF guideline :*** To provide standards and guidelines for system building block.

The basic element in ANSA is a service. A service is an information handling function - processing, storage or transfer. Clients use a service and servers provide a service. Here the client and server are computational objects. An ANSAware "object" is an encapsulation of an application and its data. Objects communicate by passing interface references to each other.

4.4.2.2 ***Suggested GenSIF guideline*** : A general strategy for system decomposition.

ANSAware does not have any guidelines for system decomposition besides that it uses object-oriented philosophy.

4.4.2.3 ***Suggested GenSIF guideline :*** Standards for interfaces.

In ANSAware an object provides services via interfaces. An object can provide and use more than one interface simultaneously. Objects communicate by passing interface references to each other. ANSAware gives guidelines and principles for interfaces specifications.

4.4.2.4 ***Suggested GenSIF guideline :*** A communication model.

Each ANSA system is running several applications, which are linked together with a trader and configuration manager. The trading service contains operations which allow

engineering objects to register the service they provide (known as exporting) and to look for services which they intend to use (known as importing). Engineering objects interact with one other through the nucleus. Below the nucleus there are components to provide execution protocols and message passing protocols. The nucleus components take the basic resources of the local infrastructure and build on them to provide a basic distributed computing environment common to each host. Thus providing a basic support platform for distributed computing.

4.4.2.5 ***Suggested GenSIF guideline :*** A data handling model.

In ANSAware an objects provides a set of operations by which it can be manipulated and these operations are accessible via interfaces.Some of these operations are

- Synchronous operations - a calling thread of control is transferred to the nominated operation with arguments and when the execution of the operation terminates the thread of control returns with the results.
- Atomic operations - they are all-or-nothing in effect and are indivisible in sense that the evaluation on one atomic operation cannot depend upon the partial evaluation of another.
- Asynchronous operations - they run in parallel with the thread that invoked them but there is no way for the invoker to randezvour with completion of the activity.

Summary : ANSAware does not give enough information about Integration Architecture but gives enough information about the mode of communication and the communication model. It does not include any strategy for system decomposition or data handling model.

4.4.3 The Technical Infrastructure

4.4.3.1 *Suggested GenSIF guideline :* Communication Tools.

The computation and engineering models are the ones that bear most directly on the use and construction of distributed systems. Figure 4.2 introduces the nucleus components. These take the basic resources of the local infrastructure and build on them to provide a basic distributed computing environment common to each host. These nucleus components are then able to work together along with the trader and configuration manager to provide a basic support platform for distributed computing.

ANSAware provides following tools for communication:

IDL - the interface definition language which specifies the operations available in an interface

STUBC - stub compiler which converts an interface specification into a set of stub routines and header files in C for inclusion in programs which will use that interface.

PREPC - a preprocessor which extracts control commands from C programs and translates them into calls to the stub routines provided by STUBC.

These tools insulate the programmer from the details of the ANSA run-time system. Programmer writes the client and server routine using embedded PREPC statements in C source code and by invoking the preprocessor on these files to yield C source files. These files are compiled and linked together to yield a capsule.

4.4.3.2 *Suggested GenSIF guideline :* Database Components.

ANSAware does not mention anything about database components except that it uses

communication tools to support basic distributed database processing.

4.4.3.3 ***Suggested GenSIF guideline :*** User-interface generator.

ANSAware suggests user can build his/her own user interface generator.It provides a means for using X11 Toolkit from within ANSAware applications, thus making it possible to create ANSAware applications with an X11 user interface.But other than that it does not give any direct elements required for generating user interfaces. There are no guidelines given for predefined sets of interface tools, programming languages, hardware component other technology required.

Summary : ANSAware covers a lot of information for communication but not enough information about database components and user interface.

4.4.4 Conclusion

The GenSIF framework is based on the idea to integrate the activities of the development process within a specified application domain by modeling the domain, deriving an integration architecture and by providing a set of standardized enabling technologies.

ANSAware is developed to support the design, implementation, operation and evolution of distributed information processing systems. It is the implementation of the ANSA architecture which provides a framework of specifications as an integrated set of structures, functions design recipes and implementation guidelines for construction of multi-vendor, heterogeneous, multi-domain Distributed Systems. ANSAware does not provide any information on Domain Analysis but it only specifies that it uses an

object-oriented approach to develop its applications. The communication model of the Integration Architecture is described but it does not give any guidelines for system decomposition or data handling as in GenSIF. ANSAware deals with the communication tools of the Technical Infrastructure but information on database components or user-interface generator is not given.Therefore ANSAware fits at the Technical Infrastructure level in GenSIF.

4.5 Mapping ANSAware to the OSCA Architecture

The OSCA architecture is designed to allow heterogeneous software products residing on a variety of computing systems, when they are designed within the OSCA architectural framework, to behave as a cooperating hole in a loosely coupled, distributed configuration so to achieve end-to-end automation, corporate data access and rapid deployment of advanced technologies.

ANSAware is an implementation of the ANSA architecture. The purpose of the ANSA is to support the design, implementation, operation and evaluation of distributed information processing systems where the different components that make up the systems, such as applications packages, operating systems, computers and networks, come from different vendors. The complexity that arises from this heterogeneity of hardware and software can only be managed if information technology vendors adopt a common approach to the design and interconnection of the components they offer.

4.5.1 Domain analysis

4.5.1.1 *Suggested GenSIF guidelines:*

To define the domain boundary.

To identify and classify the domain model.

To represent acquired information in the domain model.

To evaluate and validate the domain model.

To update and refine the domain model.

Both OSCA and ANSAware do not directly offer any guidelines for domain

analysis. However, OSCA refers to an enterprise and information model for the domain analysis. Though ANSAware does not deal with domain analysis. ANSA has enterprise model in its concepts.

Summary : Both the OSCA architecture and ANSAware do not offer any guidelines for domain analysis. However OSCA and ANSA refer to enterprise model and information model in domain analysis.

4.5.2 Integration Architecture

4.5.2.1 ***Suggested GenSIF guideline:*** To provide standards and guidelines for system building blocks.

The OSCA architecture suggest that building block in the system, possess well define, coherent, business aware functionality and interfaces. It also specifies the guidelines that the interfaces of a building block should meet to interact with other building blocks and the sharable infrastructure.

In ANSA a service is an information handling function - processing, storage or transfer. Clients use a service and servers provide a service. A service implementation is a program made up of some number of programmer-defined computational objects and objects communicate by passing interface references to each other.

Services are divided into application services which are specific to the task to be performed by the system (e.g. a booking service for airline reservations) and architectural services which identifies the architecture itself (e.g. trading, which allows clients to find servers via a system wide directory structure for determining the

availability of services).

4.5.2.2 ***Suggested GenSIF guideline:*** A general strategy for system decomposition.

The OSCA architecture suggest that business aware functionality be separated or grouped into layers or categories. They are:

Corporate data layer

Business processing layer

User layer

Each of these layer is sub-divided into building blocks.

ANSAware does not have any guidelines for system decomposition except that it uses object-oriented philosophy.

4.5.2.3 ***Suggested GenSIF guideline:*** Standards for interfaces.

In the OSCA architecture the interaction between building blocks is done by contracts, a specification of a well define set of business aware functionality and commitment by the building block to offer that set of functionality to all other building blocks, in a way which adheres to the principles of contracts.

In the ANSAware an object provides services via interfaces. An object can provide and use more than one interface simultaneously.Objects communicate by passing interface references to each other. Possession of an interface reference by a client allows the invocation of service operations provided at that interface by a server. Interface references (and hence the ability to use services) may be freely passed from one object

to another, with or without an intermediary trading service, or trader. Thus, ANSAware gives guidelines and principles for interaction using interfaces.

4.5.2.4 ***Suggested GenSIF guideline:*** A communication model.

The OSCA architecture suggests free and implementation-independent communication of messages in a distributed environment via a predefined infrastructure.

The infrastructure and infrastructure interfaces adhere to the following principles:

- Building block enablements
- Release independence
- Infrastructure and resource independence
- Location independence
- Isolation from infrastructure internals
- No accessibility assumption for infrastructure interfaces crossing system boundaries.
- No shareability of bound infrastructure
- Use of standards
- Equality of invocation and
- Security environment

Each ANSA system is running several applications linked together by trader and configuration manager. The trader service contains operations which allows engineering objects to register the service they provide or to look for services which they intend to use. These engineering objects interact with one another through the nucleus. The

components in each nucleus cooperate to provide an applications platform spanning the base systems offering basic access and location transparency. The classes of transparency are replication transparency, failure transparency (by combining replication and atomicity techniques) and migration transparency.

4.5.2.5 ***Suggested GenSIF guideline:*** A data handling model.

The OSCA architecture gives principles for data layer building block. It also specifies the step to cover redundant data and consistancy rules.

In ANSAware objects provide a set of operations by which they can be manipulated and these operations are accessible via interfaces.Some of these operations are

- Synchronous operations
- Atomic operations
- Asynchronous operations

Summary : When we map the OSCA architecture and ANSAware using guidelines given by GenSIF, it can be seen that the OSCA architecture exactly fits in the conceptual integration architecture level in GenSIF. ANSAware does not describe in detail guidelines for integration architecture; it describes the communication model and the mode of communication used. ANSAware does not include any strategy for system decomposition or data handling model.

4.5.3 The Technical Infrastructure

4.5.3.1 ***Suggested GenSIF guideline*** : Communication Tools

The OSCA architecture does not give any communication tool support directly. It only discusses the handling of protocols and messages on the application level. Messages are sent and received via "sharable infrastructure" using "logical building block addressing".

ANSAware provides tools like IDL, STUBC and PREPC for communication between application components. The trader is a distributed application component which acts as a directory and management facility for distributed application components. The trading service contains operations which allow engineering objects to register the service they provide (known as exporting) and to look for services which they intend to use (known as importing).

4.5.3.2 ***Suggested GenSIF guideline*** : Database Components.

The OSCA architecture suggests a distributed data layer approach at conceptual level and suggests to interact with database layer building blocks. However it does not give direct guidelines such as programming languages, hardware components to handle data transaction among distributed databases, standardized interfaces for the data storage etc.

ANSAware does not mention anything about database components except that it uses communication tools to support basic distributed database processing.

4.5.3.2 ***Suggested GenSIF guideline:*** User-interface generator.

The OSCA architecture suggest user interaction via user layer building block at conceptual level but does not give direct requirements for generating user interfaces.

ANSAware suggests user can build his/her own user interface generator.It provides a means for using X11 Toolkit from within ANSAware applications, thus making it possible to create ANSAware applications with an X11 user interface. But other than that does not give any direct elements required for generating user interfaces.

Summary : The modularity and the conceptual design of the OSCA architecture clearly show that it can utilize an channel based communication tool like ANSAware. However, both the OSCA architecture and ANSAware do not give enough information about database components used and only suggest that a user can build his/her own user interface generator.

4.5.4 Conclusion

The OSCA architecture promotes the interoperability and operability of software products which typically consist of large number of programs, transactions and data bases. ANSA is an architecture for Open Distributed Processing. The objectives for the Distributed System are to be generic to many fields of application, to be portable across a wide range of operating systems and programming languages, to be operable in a heterogeneous, multi-vendor environments (which is similar to the objects of OSCA). Since ANSAware is an implementation of the ANSA architecture, ANSAware could be used for the implementation of OSCA.

From this point of view, our conclusion is that the OSCA architecture describes an integration architecture while ANSAware defines a technical infrastructure.

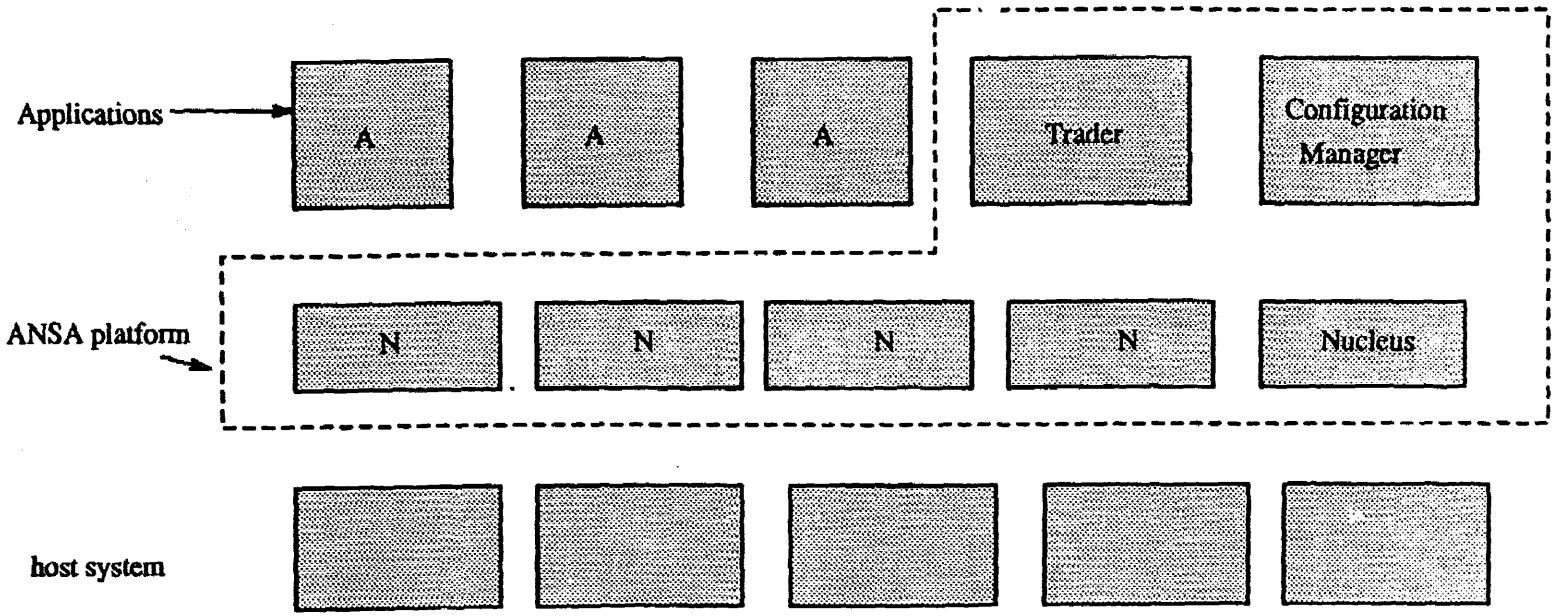

Figure 4.1 An ANSA System

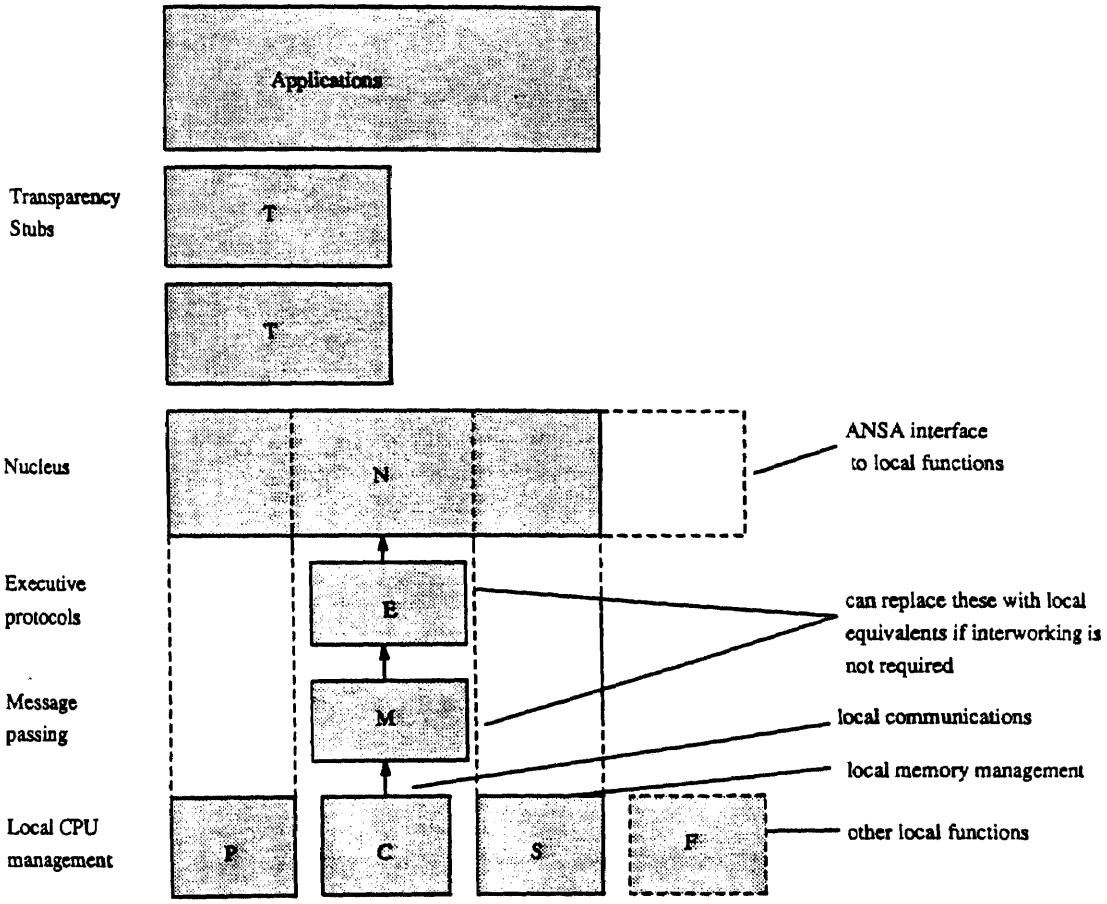

Figure 4.2 An ANSA Capsule

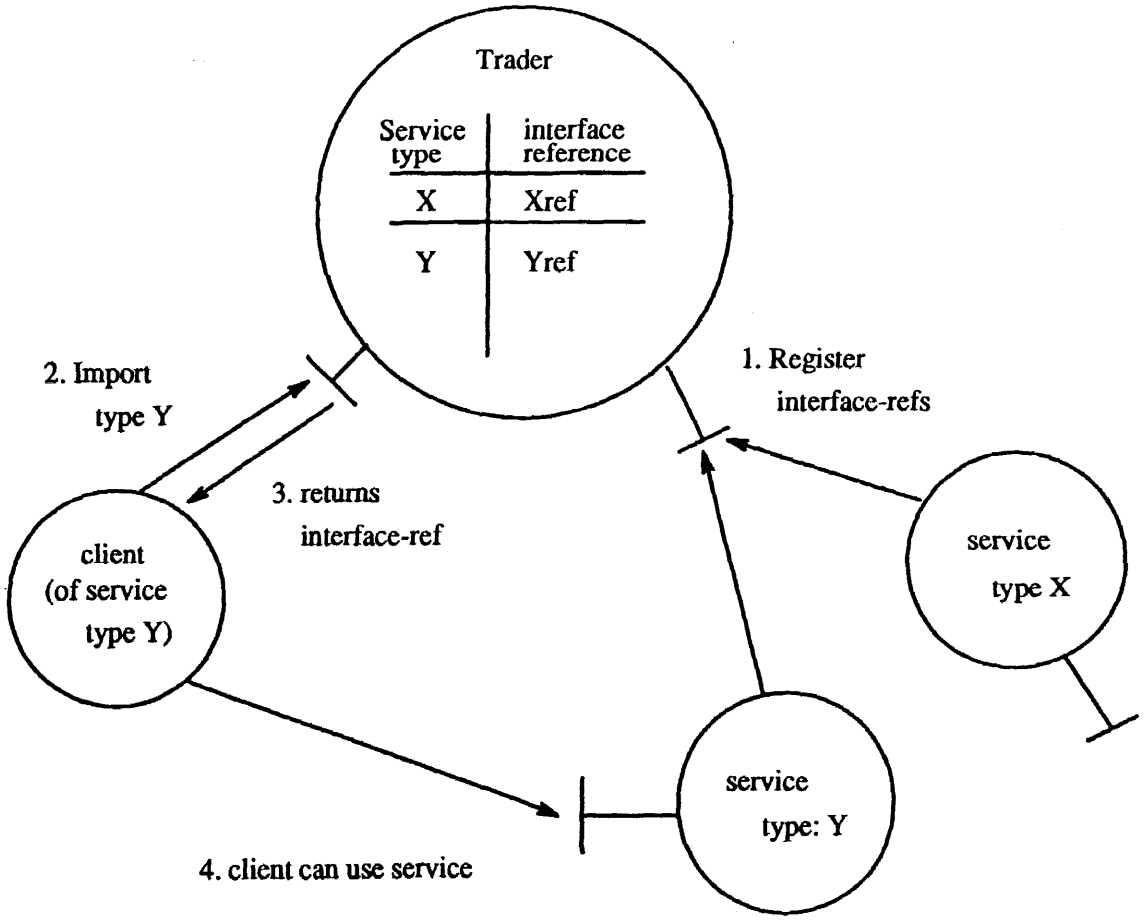

Figure 4.3 Trading Interfaces

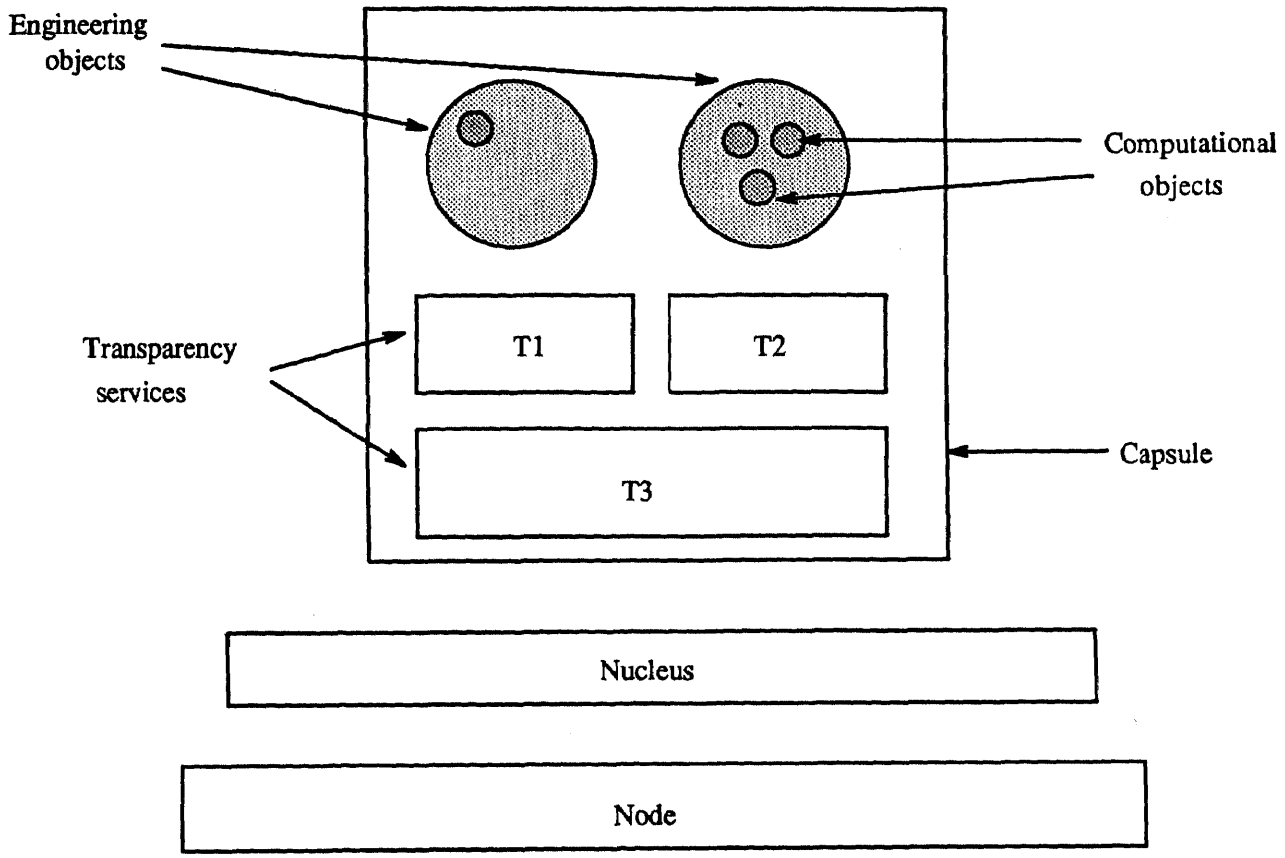

Figure 4.4 Capsule and Nucleus

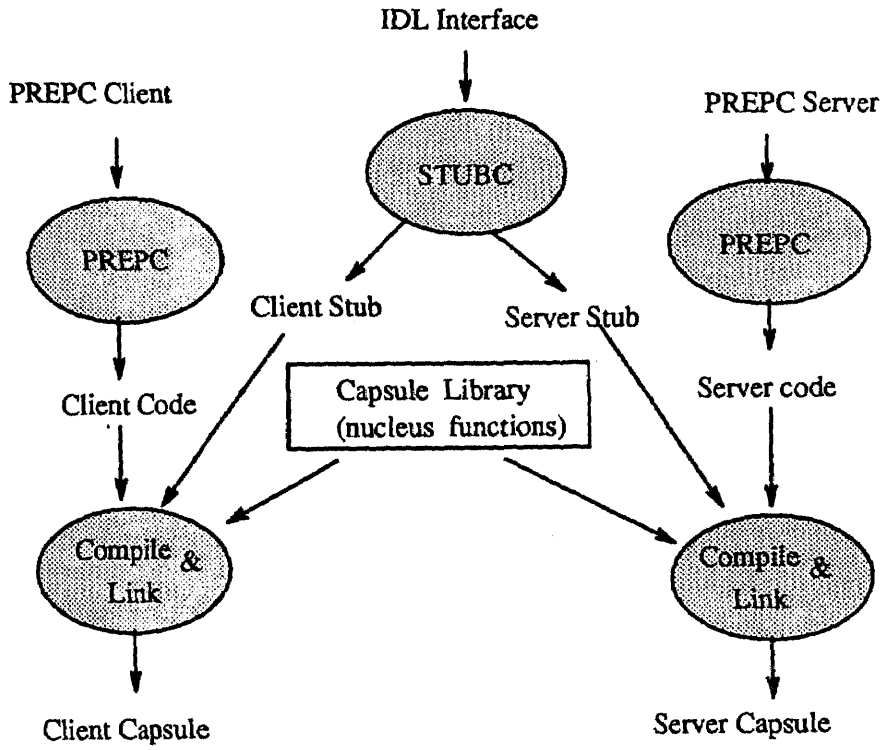

Figure 4.5 ANSAware Tools

CHAPTER 5

IBM'S SYSTEMS APPLICATION ARCHITECTURE (SAA)

5.1 Overview of SAA[7]

SAA, the system application architecture, is the IBM[8] framework for software development. SAA lays out strategies for improving the quality of computer systems by standardizing software internals and externals. The word "Standardizing" here means standards that IBM wants to implement for its own and vendor's products. SAA is not only for IBM itself, but for any organization using IBM computers. SAA covers software development, use and maintenance across IBM's major computer hardware and operating systems. It is intended to be universally applicable.

The SAA solution is aimed at helping IBM customers to obtain greater productivity from existing resources and create more productive business environments. The SAA solution is designed to provide growing commonality and integration across its environments in the areas of

- programming interfaces,
- user access,
- communications support and
- applications.

Figure 5.1 shows the the SAA structure. For the programming interfaces, user access and communications support, IBM has published specifications - definitions of

---

[7] SAA is a trademark of IBM Corporation.

[8] copyright International Business Machines Corporation.

those elements that will have significant commonality across environments. The SAA environments are MVS, VM, OS/400 and OS/2 Extended Edition. MVS includes the base system (TSO/E, APPC/MVS and batch) and the CICS(Customer Information and Control System) and IMS(Information Management System) subenvironments. Following are major advantages of the SAA framework:

- applications that can be moved easily between systems;
- applications that can span systems;
- applications whose user access is simpler and more uniform.

Software vendors who choose to build on IBM products will benefit and customers seeking broad solutions for their personal, departmental and enterprise-wide data processing needs will profit.

As quoted in IBM system journal(44)

SAA is a collection of selected software interfaces, conventions and protocols that provides the framework for development of consistent applications across the future offerings of the major IBM computing environments -- system/370, system/3X and Personal Computer. This is not a specific product, but rather a pervasive software architecture that underlines the commitment to provide, in an evolutionary way, cross-system consistency across a broad spectrum of hardware, architecture and operating systems environments.

A fundamental of SAA is to standardize the interface between programs and computers, between users and applications and among programs that communicate with each other. The SAA framework has four components. These components address the

user, programming and communications interfaces of systems.

- User interface (Common User Access- CUA)
- Developer tools and strategies (Common Programming Interfaces- CPI)
- Communications support for systems operating on diverse hardware(Common Communications Support- CCS)
- Common Application

A common application confirms to CUA interfaces and is written using appropriate CPI interfaces and CCS functions. Applications that follow the SAA standards and use SAA standard software interfaces are called "SAA Applications" and are the fourth component of the SAA strategy. OfficeVision is IBM's first common application.

5.1.1 User interface (Common User Access- CUA)

In software industry the level of common functioning has generally been missing; COBOL and UNIX[9] not withstanding. The software developed for IBM computers, mainframe as well as mini- and personal computers has had a very low level of receptivity to standards. SAA wants to define the standards for the user interface by defining how an application should present itself to a computer user and how that user should be able to interact with the application. The "Standards" here means to standardize user interactions, e.g. to close the window a user should click at a particular corner of a window. Right-now no standards exist which every developer has to follow:

[9] UNIX is a registered trademark of UNIX System Laboratories Inc.

Microsoft Windows and other products like OS/2 windows have different keystrokes for particular actions. A user interface is a set of tools and mechanisms that a user uses to interact with computer. This component of the SAA strategy is called Common User Access, CUA. CUA is described in two " interface design guides". The main aim of CUA is to improve user understanding of computer applications. If the user learns to use one application that has a CUA interface, the user can quickly learn to use any other application that has a CUA interface.

Common User Access is the specification of what the computer user should see and be able to do in interacting with application software. CUA defines the "user interface," how the person communicates with the computer and vice versa. CUA supports a GUI(Graphical User Interface). Its goal is to help people get their jobs done more rapidly and with fewer errors. The objective of CUA is to make computers more usable by providing an easily understandable environment. CUA provides standards for:

Presentation of the application to a user: what the users see

Interaction: how users interact with the components

Actions: how similar user actions are implemented

Communication sequence: how users and the computer communicate with each other.

CUA encourages to create a visual presentation of an application to the end-user. CUA recommends to construct consistent actions for user interaction with computer. CUA suggests to provide immediate actions with feedback and reversible actions. CUA standards intend to create actions having intuitive meaning. CUA encourages to design

principles that reduces a user's memory load. It is useful to provide windows with menu, so that a user does not need to memorize commands.

SAA provides a user-interface generator, such as PM[10], Presentation Manager for OS/2. At this stage some higher level programming tools, ranging from subroutines libraries to application generators are available from IBM and third parties. PM is a library of routines that implements a visual presentation of applications to end-users.

PM is used for Graphical model applications and DM, Dialog Manager, is used for Text model applications. Presentation Manager provides a full set of application program interfaces(API) for creating windowed user interface under OS/2. Presentation Manager and Dialog Manager routines performs such functions as displaying and sizing a window and displaying the contents of a list box or a panel.

5.1.2 Developer Tools and Approaches (Common Programming Interfaces- CPI)

SAA directly addresses the ability of application developers and maintainers to create and enhance complex systems of programs by defining a common programming interface and by promoting "object oriented " development. SAA- CPI give rules for creating modular programs or establishing object hierarchies. CPI is a step towards creation of common developer understanding of program structure and purpose.

The SAA CPI components offer a broad span of function that can meet most data processing needs. Each language in the interface is a valuable and efficient data processing aid in itself. These languages are provided by applicable IBM or IBM

---

[10] PM is a trademark of IBM.

AD/Cycle Business Partner products. In addition, each can call the interface services to create integrated applications that accomplish sophisticated ends. The components of the interface are made available in the three areas into which well-designed applications are structured:

User interaction (dialog display and management)

Functional logic

Data access.

The Common Programming Interface supports a variety of application styles, including the use of a programmable workstation linked with a host or server. It is not necessary that every application should have above mentioned structure, in order to take advantage of CPI components. The application's user interaction (and possibly other work) may take place on the workstation, while the application's intensive processing and data operations may take place on the server and/or host.

This style, often called cooperative processing, allows each part of the application to run where it is best suited and can contribute most to productivity. The front end takes advantage of the user-friendliness and display capabilities of the workstation. The back end of the application runs efficiently on the host or server, making use of its processing power and data facilities.

SAA common programming interface (CPI) is a collection of programming languages and application interfaces to system control and application enabling software. Other components, like the Presentation Interfaces and Communication Interfaces, represent new systems software that IBM is providing as a part of the SAA strategy.

Some other components like Data base Interface and the Query interface, are special purpose language constructs which can be used in conjunction with other programs. The CPI defines syntax and semantics for all these languages and interfaces. The Common Programming Interface (CPI) is an important base of the Systems Application Architecture framework. It is the languages and services that programmers employ. By using this interface, developers can create applications with minimal concern about the environments in which they will run. The components of the CPI fall into two general categories:

Language Interfaces	**Service Interfaces**
Application generator	Communications
COBOL	Database
FORTRAN	Dialog
PL/I	Presentation
Procedures Language	PrintManager
RPG	Query
Repository	
Resource Recovery.	

In the CPI, the SAA dialog and presentation interfaces allow a programmer to create applications that easily converse with human users at a programmable workstation. The SAA dialog interface is available for writing more simple forms of user interaction. The SAA presentation interface provides a full range of screen design and graphics possibilities.

The main processing and overall control parts of the application are created through the application generator or one of the traditional high-level languages (C, COBOL, FORTRAN, PL/I, or RPG).

For the data access SAA provides database interfaces. These are based on the standard SQL language. These interfaces let an application read and write information in relational format. The SAA query interface includes a higher-level database query capability and adds the ability to quickly produce results in formatted, easy-to-read reports. Access to data is provided in various ways. Access to files is through the familiar I/O statements in the languages.

The SAA printmanager interface provides a consistent and portable means of requesting print services throughout the enterprise. The SAA communications interface permits easy construction of cross-system applications. Through a simple call mechanism, various programs in an application can reside in various parts of enterprise and yet function as if they were one. The SAA resource recovery interface can be used with the communications interface or by itself to provide applications with a commitment coordination mechanism for change integrity.

The SAA Procedures Language interface is intended to create system procedures. These system procedures integrate and encompass application parts. It can also provide a macro processing facility for applications.

Because the Common Programming Interface components span the spectrum of typical application needs, they offer the assistance customers require-whether to do a programming job using a single component, or to build a more complex data processing

solution using multiple components. Moreover, they offer all the value of cross-system consistency and connectivity.

5.1.3 Communications Support for System Operating on Diverse Hardware (Common Communications Support- CCS)

Common communication support defines the protocols for implementing cooperative processing applications. CCS also deals with task of communications between and among computers and programs.

CCS defines protocols necessary for sending information

- from a user interface program on a workstations to a program on a mainframe, mini- computer, or other workstation;
- from one computer to another across a network;
- from processor to a printer;
- from a word processing program to one type of computer to a different word processing program on other.

It defines high level protocols for applications to talk to each other and low level protocols for the bit streams sent between two different hardware devices. CPI provides a consistent model of the communications process. CCS illustrates different levels of communication between device, networks, systems and applications and how they interact. Figure 5.1 CCS consists of IBM protocols and selected Open Systems Interconnection (OSI) standards that allow both IBM and non-IBM systems to be interconnected. Consistent implementation of CCS architectures will allow networks to

be built from systems with vastly differing capacities - from the smallest SAA system to the largest.

5.1.3.1 Elements of Common Communications Support

Common Communications Support (CCS) comprises elements of Systems Network Architecture (SNA[11]), as well as selected standards from the following:

International Telegraph and Telephone Consultative Committee (CCITT) recommendations

Institute of Electrical and Electronics Engineers (IEEE) standards

Parts of Open Systems Interconnection.

Elements of CCS are grouped into six broad categories:

Transmission Objects

Data streams

Application services

Session services

Network

Data link control.

These CCS categories are introduced in the following sections.

- **Transmission Objects**

Transmission objects contain the kinds of data that can be combined to create a finished product like a document or a database. The word objects, here is used to refer to

[11] SNA is a registered trademark of IBM.

transmission objects. The types of data that objects can contain include text, graphics, images and formatted data (such as database records). When a person creates a document, for example, it can contain any combination of these data types except database records. When that document is transferred between SAA systems, it must be in a format and structure that each system can interpret. Object content architectures (OCAs) define the structure and content of objects that can exist in finished products like documents and databases.

The object content architectures in Common Communications Support are:

Presentation Text Object Content Architecture (PTOCA)

Image Object Content Architecture (IOCA)

Graphics Object Content Architecture (GOCA)

Font Object Content Architecture (FOCA)

Formatted Data Object Content Architecture (FD:OCA).

Objects are transmitted in data streams and can be stored in libraries by applications and hardware components of the network.

- **Data Streams**

A data stream is a continuous ordered stream of data elements conforming to a given format. Application programs can generate data streams destined for a printer a workstation, or another application program. The data streams in Common Communications Support are:

Intelligent Printer Data Stream (IPDS)

3270 Data Stream (3270 DS)

Mixed Object Document Content Architecture (MO:DCA)

Character Data Representation Architecture (CDRA)

Revisable-Form-Text: Document Content Architecture (RFT:DCA).

- **Application Services**

Application services enhance the activity of the network by providing architectures that allow data distribution, document interchange and network management. The architecture's application services are:

For SNA:	For OSI:
Document Interchange Architecture (DIA)-SNA/Distribution Services (SNA/DS)	File Transfer, Access and Management (FTAM)
SNA/Management Services (SNA/MS)	X.400 Message Handling System
Distributed Data Management (DDM)	OSI Association Control Service Element (ACSE).
Distributed Relational Database Architecture (DRDA)	

- **Session Services**

Session services are required to establish communication between two application programs, to transfer data between the application programs and to terminate communication between the application programs. The architectures providing session services are:

For SNA:	For OSI:
SNA Logical Unit Type 6.2 (LU 6.2) architecture	OSI Presentation Layer-Kernel and ASN.1 OSI Session Layer-Versions 1 and 2 OSI Transport Layer-Classes 0, 2 and 4.

OSI session connection and an SNA LU 6.2 conversation have many similar functions, although the terminology differs. OSI session connection provides the means for associated service users in different networks to organize and synchronize their dialog and to manage their data exchange. An LU 6.2 conversation provides a logical interface through which transaction programs can access the SNA network and its resources.

- **Network**

Network services allow connectivity between systems. The architectures that provide network services are:

For SNA:	For OSI:
SNA Type 2.1 Node architecture for low-entry networking	Connectionless-Mode Network Services (CLNS) using Internet Connection-Oriented Network Services (CONS) using Subnetwork Interface to X.25 X.25 Packet Level Procedures for DTEs.

- **Data Link Control**

A link consists of transmission media and a data link control protocol. The transmission media may include any combination of telephone lines, microwave beams, fiber optics, satellite links, or coaxial cables. A data link control protocol specifies how to interpret

control data and how to transmit data across a link.

SAA systems may be interconnected using local area networks, telecommunication links, or packet-switched networks. The primary objective of a local area network is to provide high-speed data transfer among a group of nodes within a building or a group of buildings in a campus or office-complex environment. For SAA systems spread over a wide geographic area, an enterprise requires telecommunication links. In some cases, an enterprise needs to interconnect non-SAA systems to a network of SAA systems. The data-link-control architectures in CCS are:

For SNA:

Synchronous Data Link Control (SDLC) to interconnect SAA systems over a wide geographic area.

5.1.4 A Common Application Architecture

Common application architecture is an architecture for the various elements of SAA. Common application architecture relates SAA elements to each other to form common applications within the application itself.

SAA deals with applications which consist of only two types of software components, one that provides the functions from a user point of view and others which provide various services independent of the specific purpose. In this view of an application's internal architecture, SAA deals primarily with the "systems architecture," an integral component of the overall "application architecture."

An application architecture describes the structure of an application, how its

different pieces fit together. A general "application architecture" is an overall framework that describes the structure of a large number of systems, such as business systems or manufacturing systems. The architecture is a series of "layers" and levels, each identifying a class of system functioning. Figure 5.2 shows an architecture with four layers, some horizontal and some vertical. A horizontal layer indicates a layer which isolates two other layers and a vertical indicates a parallel layer, which spans two horizontal layers. The application function layer is what might be thought of as the substance of the application system. Each layer consists of programs that provide certain services to those at the next higher level and to parallel layers and that call upon services from the next lower level layers via defined interfaces.

Following such a layered architectural concept allows to create application systems that reuse common parts. Viewing each program in a system as a constituent part of a particular architectural layer serves to isolate programs in one layer from changes in the others. A significant result of this view is to simplify design and maintenance of applications by encouraging the reuse of entire layers of software.

The application enabling layer provides database access services to the application function layer while calling upon the operating system layer to perform its functions. Programs in the application function layer should not have to call upon services in the operating system layer directly to perform data access -- that should all be provided through the application enabling layer. If the application enabling layer is general enough, it can be used across a wide range of application systems. The existence of many generalized database management systems is an example that this concept is followed by

many software developers. The layered application architecture is somewhat idealized concept, but this is a normal translation of a theoretical construct into actual practice.

The application function layer of a system can itself be divided into levels. For example, in Figure 5.3 the application function layer has been divided into four, with "application specific programming" now representing the substance of the particular application. The lower layers represent services of increasing generality used by the application specific programs. The stair-stepped shape of the boxes indicates that the application specific programs may need access to each of the lower levels. IBM's OfficeVision products are architected according to this exact model.

SAA defines key components of a comprehensive application architecture, i.e., the interfaces to enabling and system control software. Referring to Figure 5.4, the SAA components provide an architecture of systems layers and lower application layers. The structure of OfficeVision for OS/2 hints at the introduction of higher level interfaces.

Common User Access defines a strategy for that part programming interface dealing with user access. The Common Programming Interface defines the interfaces between the application layers and the systems layers. Common Communications Support and various parts of the CPI also deliver the functions of the systems layers of the architecture to the application layers.

5.2 Mapping SAA to GenSIF

SAA is a systems application development strategy. We map SAA into GenSIF. We first list GenSIF guideline concept for each activity of Mega-system task and then check how much this guideline is covered in SAA.

5.2.1 Domain analysis

5.2.1.1 *Suggested GenSIF guidelines :*

To define the domain boundary.

To identify and classify the domain information.

To represent acquired information in the domain model.

To evaluate and validate the domain model.

To update and refine the domain model.

SAA : SAA does not gives guidelines, how to define the application domain boundary. However, AD/Cycle, the SAA application development environment speaks about Enterprise Modeling. Enterprise Modeling provides tools to develop an enterprise computing environment model. These tools can be used to create entity-relationship models of the business and prototyping rules that govern them. This is a indirect approach of SAA to cover domain analysis.

SAA also refers to the Application Development Workbench (ADW)/ Planning Workstation: a PC-based business modeling tool from KnowledgeWare, Inc., which assists in creating software designs, including classification and identification of information.

Summary : SAA does not cover domain analysis as a part of the architecture. SAA indirectly suggests to use offerings from IBM other vendors to cover different activities of domain analysis.

5.2.2 Integration architecture

5.2.2.1 ***Suggested GenSIF guideline :*** To provide standards and guidelines for system building blocks.

SAA does not refer to the word "building blocks", however it acknowledges that a system is decompos into objects or modules. SAA gives specific guidelines for CPI, CUA and CCS as a part of the architecture. However, these guidelines are not at a conceptual and abstract level. These guidelines are more at tool-level.

CUA is the system element that deals with Common user access. SAA provides guidelines for CUA. SAA directly addresses the definition of common programming interface. To create and enhance complex systems of programs and to promoting "object oriented" development, SAA-CPI gives necessary guidelines. SAA defines key components of a comprehensive application architecture, i.e., the interfaces to enabling and system control software. The CPI defines syntax and semantics for programming languages and interfaces. By using the software products that deliver the CPI, that is, the language compilers and interface routines, SAA wants to create applications with common form and structure. SAA CPI give rules for creating modular programs or establishing object hierarchies. The CPI is a step to create a common developer understanding of program structure and purpose.

5.2.2.2 ***Suggested GenSIF guideline :*** General guidelines for system decomposition.

SAA : SAA explicitly gives a system decomposition strategy. The SAA solution is based on IBM's approach for structuring product and software management. Figure 5.5 shows the decomposition of systems.

- **Application enablers** represent code, such as compilers and database programs, needed by application programs.
- **Communications** represents code that allows a specific system to communicate with its attached devices, or with other systems in the network-VTAM in ESA/370, ESA/390 and VM and the communications portions of OS/2. Extended Edition and OS/400.
- **System control programs** represent the operating system and its extensions for a given hardware environment.

However, system decomposition as defined in SAA is mainly a decomposition of the application of environment and not a decomposition of the application itself.

5.2.2.3 ***Suggested GenSIF guideline*** : Standards for interfaces(internal and user)

SAA : SAA provides guidelines and standards for both internal and external interfaces. The external interfaces are covered under SAA-CUA. SAA wants to define the standards of user interface by defining, how an application should present itself to a computer user and how that user should be able to interact with the application.

SAA offers design specifications and guidelines for internal interfaces. These are covered in CPI specifications. SAA-CPI guidelines are not at conceptual and abstract level. These are more at tool level. It seems that IBM designed these guidelines to utilize

existing tools to produce internal interfaces.

5.2.2.4 ***Suggested GenSIF guideline :*** A communication model.

SAA : SAA offers communications model. The SAA-CPI provides a consistent model of the communications process, illustrating different levels of communication between device, networks, systems and applications and how they interact. The communications Interfaces of CPI defines a specific type of inter-program communication in which programs hold a synchronized "conversation" with each other.

5.2.2.5 ***Suggested GenSIF guideline :*** A data handling model.

SAA : Data handling is not represented as a conceptual model in SAA. However, SAA-CPI covers guidelines for data handling. Again, data handling from the point of view of technical processes and not on the level of applications.

5.2.3 The Technical Infrastructure

5.2.3.1 ***Suggested GenSIF guideline :*** Communication tools.

SAA : SAA gives Communication tools as a part of CCS elements. CCS comprises elements of SNA as well as selected other standards like: IEEE, CCITT and OSI. CCS provides protocols that allow standardize communication among devices, application programs, systems and network. CCS consists of IBM protocols and selected OSI standards, that allows both IBM and non IBM systems to be interconnected. SAA offers applications services as a component of CCS. These services provides communication

tools for enhancing network activity for communication, data distribution, document inter-exchange and network management.

CCS offers session services, which establish communication between application programs, with network gives connectivity between systems. VM/CMS gives a set of subroutine calls which can be used by application programmer to develop cooperative programs. These subroutines implement LU6.2 " program-to- program" communication defined in CCS.

5.2.3.2 ***Suggested GenSIF guideline :*** Database Components.

SAA : SAA explicitly gives database components at tool level. At this stage SAA gives database components for only SNA and selected OSI standards. SAA common communication support provides database components with Data streams. The Remote Relational Access Support facility in SQL/DS creates the same type of program-to-program communication to fulfill the SQL request.

5.2.3.3 ***Suggested GenSIF guideline :*** User-interface generator.

SAA : SAA provides user-interface generator, such as PM for OS/2. At this stage some of higher level programming tools, ranging from subroutines libraries to application generators are available from IBM and third parties. PM, presentation manager and DM, Dialog Manager are essential libraries of routines that implements the presentation and elements of SAA common programming interfaces.

PM, Presentation Manager is used for Graphical model applications and DM is

used for Text model applications. Presentation Manager provides a full set of application program interfaces(API) for creating windowed user interface under OS/2. Presentation Manager and Dialog Manager routines performs such functions as displaying and sizing a window and displaying the contents of list box or panel.

Summary : SAA covers most infrastructure components, however it seems that these components are not complete to cover all IBM hardware platforms for global integration. For example PM and DM can generate user interfaces only on OS/2 environments. At current stage there is no SAA tool available to create user interface on the system/3270. This indicates that SAA relies heavily on third party vendors and their offerings. Similarly for database components and communication tools there are only limited ready-made tools SAA offers at this time.

5.2.4 Conclusion

SAA fits currently at the technical infrastructure level in GenSIF. Integration architecture concepts offered by SAA are more tool oriented. SAA mainly focuses on two points.

- common user and developer environment
- the strategic importance of treating computing as an enterprise-wide resource.

Implementation of common applications will develop a common understanding among computer users of how applications work and a common understanding among developers of how applications have been developed. Having a common strategy for software development and use will save time and effort for new developments.

The SAA framework has some holes. These holes are the missing strategies for

certain areas. The word "missing" means unannounced or incomplete enabling software. According to SAA publications many of these missing software will be filled in as SAA evolves over the next several years.

Much of the domain analysis portion is not covered directly in SAA. SAA relies on other vendors and tools in this area. The development strategies of SAA match with some fundamentals of integration architectures and most elements of a technical infrastructure in GenSIF. Thus, SAA is a technical infrastructure for current existing IBM hardware platforms which offers very limited elements of application structuring and integration. SAA is also linked to efforts in process modeling and development environments described in AD/Cycle.

5.3 Mapping SAA to the OSCA architecture

The OSCA architecture is an interoperability architecture. SAA is a system application architecture. We map SAA with OSCA. We use the GenSIF guideline concepts for each activity of the Mega-system task to compare these two architectures.

5.3.1 Domain analysis

5.3.1.1 *Suggested GenSIF guidelines :*

To define the domain boundary.

To identify and classify the domain information.

To represent acquired information in the domain model.

To evaluate and validate the domain model.

To update and refine the domain model.

None of these architecture, OSCA or SAA, directly offer any of the above mentioned guidelines. The OSCA architecture is not intended to offer/cover these guidelines. SAA suggests to use other application developments offered by IBM or its vendors to cover domain analysis. Both these architectures do not offer domain modeling as a part of the architecture itself.

The OSCA architecture and SAA expect domain analysis to occur before any future developments. The OSCA description mentions the need for the Information and Enterprise models for domain analysis to proceed the application of the OSCA architecture. The OSCA approach utilizes all these guidelines in engineering systems. SAA mentions the need of ADE (application development environment). The

decomposition strategies of both architectures demonstrate that domain analysis is needed.

OSCA and SAA can be mapped as a stage after domain analysis in GenSIF. For the activity of domain analysis itself, both architectures offer little at this point in time.

5.3.2 Integration architecture

5.3.2.1 ***Suggested GenSIF guideline :*** To provide standards and guidelines for system building blocks.

Both these architectures give guidelines for system building blocks. SAA does not use the word "system building blocks", however it specifies implementation guidelines for "modules" and "objects" of the system. The OSCA architecture specifies the principles and guidelines for building blocks very comprehensively and at the application level. SAA is more specific and not at an abstract level but at the application level. SAA utilizes CPI to develop modular systems and gives specifications for CPI, CUA and CCS.

5.3.2.2 ***Suggested GenSIF guideline :*** General guidelines for system decomposition.

The OSCA architectures gives guidelines for system decomposition on the application level. SAA suggests to break application systems into modules to handle the complexity of the system. However, system decomposition guidelines of these two architectures are different:

Table 5.7 System decomposition components for OSCA and SAA.

OSCA	**SAA**
Corporate data layer	Common Communications Support

Business processing layer	Common Programming Interface
User layer	Common User Access

The SAA decomposition strategy is more tool oriented. The OSCA strategy is more application oriented.

5.3.2.3 ***Suggested GenSIF guideline*** : Standards for interfaces(internal and external). The OSCA architecture gives guidelines for interfaces among building blocks. SAA also gives guidelines for interfaces among different object of a system. For internal interfaces both architectures give guidelines. A difference is for external interfaces: SAA gives specifications for user interfaces in CCS. OSCA does not offer explicit guidelines for user interfaces.

The OSCA architecture suggests to keep user interaction activities separate from all other internal interaction activities. There is a layout in the OSCA architecture about user interaction management. However, OSCA does not offer direct guidelines and design specifications for user interfaces. For internal interfaces both architectures have different perspective. The OSCA architecture provides conceptual and abstract level guidelines, SAA gives technical level guidelines.

5.3.2.4 ***Suggested GenSIF guideline :*** A communication model.

The OSCA architecture gives the principles for contracts, infrastructure and infrastructure interfaces. The communications interfaces of SAA-CPI provide a consistent model of the communications process, giving different levels of communication between device,

networks, systems and applications and how they interact. The difference between SAA and the OSCA architecture is the level: the OSCA architecture lists communication model at abstract level without giving any specific tool support. It represents high level design. SAA covers communication model by using existing technology. The communication model offered in SAA is at lower level, tool level.

5.3.2.5 ***Suggested GenSIF guideline :*** A data handling model.

The OSCA architecture gives principles for data layer building block. SAA does not portrait data handling as model, however, under SAA-CPI guidelines for data handling are covered. The OSCA architecture offers a conceptual model. It does not give a tool level details. SAA gives data handling Model only as tool support.

Summary : The OSCA architecture exactly fits-in at the conceptual integration architecture level in GenSIF. SAA maintains a different level. OSCA offers an integration architecture more at conceptual level, SAA offers tool oriented architecture with impacts on the application area.

5.3.3 The Technical Infrastructure

5.3.3.1 ***Suggested GenSIF guideline :*** Communication tools.

The OSCA architecture doesn't give communication tool support straight away. The OSCA architecture discusses only the handling of protocol and messages on the application level. SAA gives Communication tools as a part of CCS elements. CCS comprises elements of SNA as well as selected other standards like: IEEE, CCITT and

OSI.

The OSCA architecture delivers all necessary foundation for communication tool, such as design and guidelines for so-called 'sharable infrastructure' etc. However, the OSCA architecture clearly notes that the details of the structure of so-called ***sharable infrastructure*** is not a part of the OSCA because this is a part of system engineering and not a system architecture.

In contrast, SAA CCS provides protocols that allow standardize communication among devices, application programs, systems and network. CCS consists of IBM protocols and selected OSI standards, that allows both IBM and non IBM systems to be interconnected. SAA offers applications services as a component of CCS. These services provides communication tools for enhancing network activity for communication, data distribution, document inter-exchange and network management.

5.3.3.2 ***Suggested GenSIF guideline :*** Database Components.

The OSCA architecture doesn't give any database Components such as programming languages, hardware components to handle data transaction among distributed databases, standardized interfaces for the data storage etc. directly. This technical infrastructure portion of GenSIF Mega-system task is not covered in the OSCA architecture. In contrast, SAA explicitly gives database components at tool level. At this stage SAA gives database components for SNA and selected OSI standards.

The OSCA architecture suggests a distributed data layer approach at conceptual level and suggests to interact with databases via data layer building blocks. This technical

infrastructure element of GenSIF is covered in SAA and is not covered in the OSCA architecture.

5.3.3.3 ***Suggested GenSIF guideline :*** User-interface generator.

The OSCA architecture does not give direct elements required for generating user interfaces. There is a no predefined set of interface tools such as programming languages, hardware component and other technology.

SAA at this stage offers some of higher level programming tools, ranging from subroutines libraries to application generators from IBM and third parties for generating user interfaces.

Summary : The OSCA architecture does not offer a ready-made infrastructure, but provides guidelines and requirements for it. SAA offers most of a ready-made infrastructure.

5.3.4 Conclusion

OSCA and SAA are systems architectures, not user documents. OSCA is on the conceptual and abstract level. SAA is on the technical level. In both architectures domain analysis is missing. However, both architectures rely to a certain degree on the result of domain analysis.

The OSCA architecture and SAA offer elements proposed in the GenSIF conceptual integration architecture. SAA gives only a tool oriented architectural guidelines. OSCA is a top-down approach and first addresses concepts of building blocks

and contracts. After defining exact guidelines for building blocks and contracts, the OSCA architecture specifies the system in the form of building blocks. SAA is a bottom-up approach of IBM. SAA does not address building blocks or objects at conceptual level. SAA does not define constraints and guidelines for building blocks.

IBM wants to develop a systems integration architecture through SAA for its existing hardware platforms. However, with SAA, IBM has still not been able to provide a **conceptual** application architecture which can be used for top-down developments. SAA, at this point in time, is a very good technical infrastructure but not a conceptual integration architecture. SAA strive to go bottom-up. SAA needs a conceptual application architecture such as OSCA for new system design and development. However, how OSCA can be mounted on SAA; how much are these two developments compitable to each other are questions and need a seperate study.

Our observation indicate that, SAA can utilize building blocks principles of the OSCA architecture. SAA wants to use object oriented developments for system design. The building blocks principles of the OSCA architecture provides necessary base for such an object oriented developments. The employment of OSCA architecture for application systems produces a interoperable modular and flexible system. This can be implemented to heterogeneous hardware platform by using the SAA infrastructure services. The OSCA architecture does not offer a technical infrastructure. SAA, in contrast to OSCA, offers a technical infrastructure by providing necessary tool support.

The OSCA architecture and SAA both are still evolving. The OSCA architecture needs a technical infrastructure to reach the hardware-platforms. SAA on the other hand

would need a comprehensive conceptual application architecture. The OSCA architecture, is **a conceptual integration architecture** which fits exactly into the GenSIF framework's conceptual level. SAA fits not **really as an integration architecture**, but rather as a **technical infrastructure** level in GenSIF.

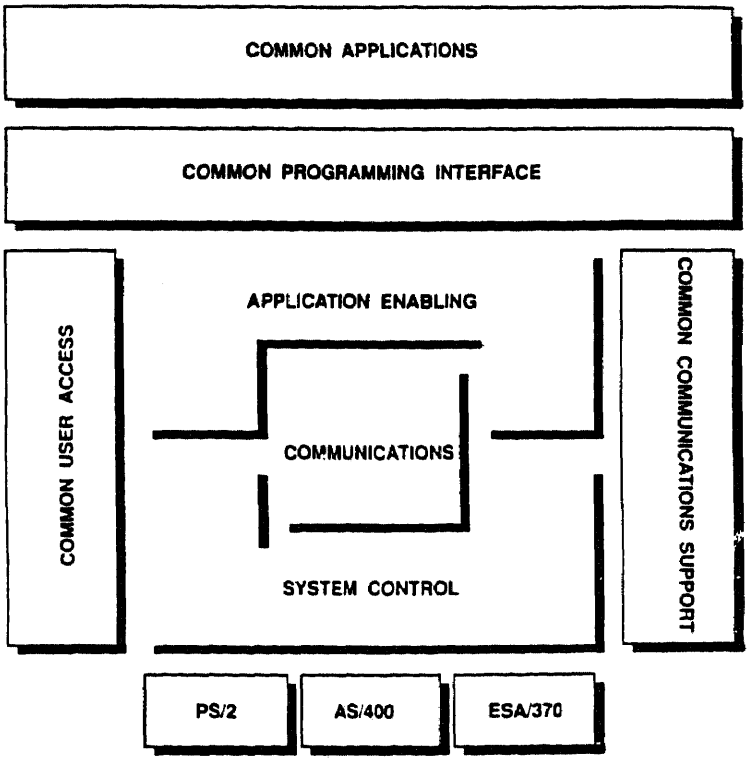

Figure 5.1 SAA - Structure of System Application Architecture

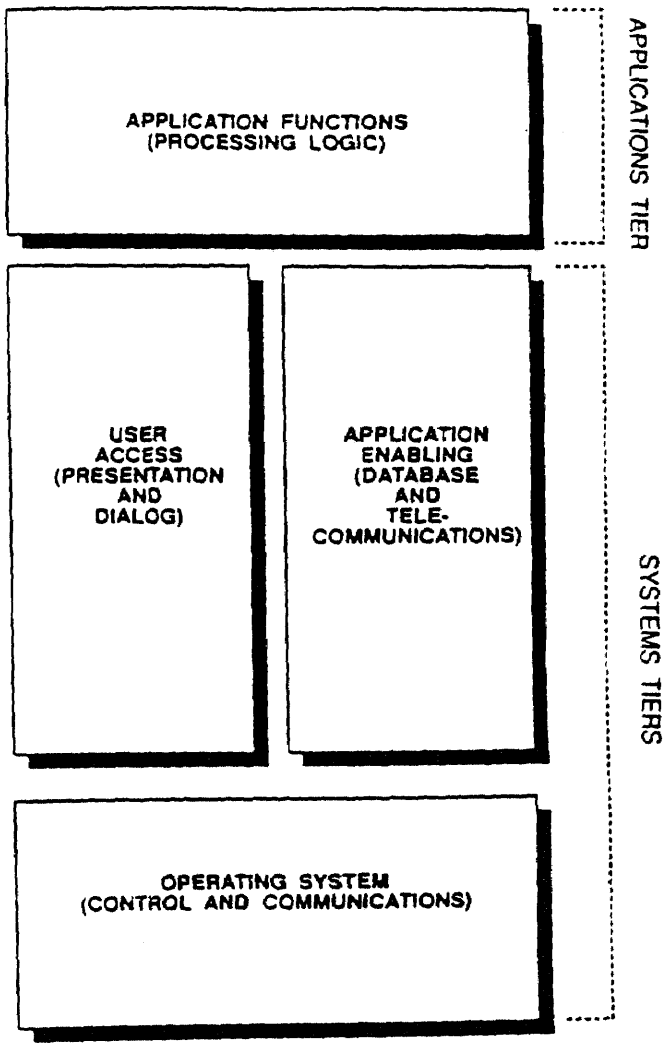

Figure 5.2 SAA - Application Architecture Layers

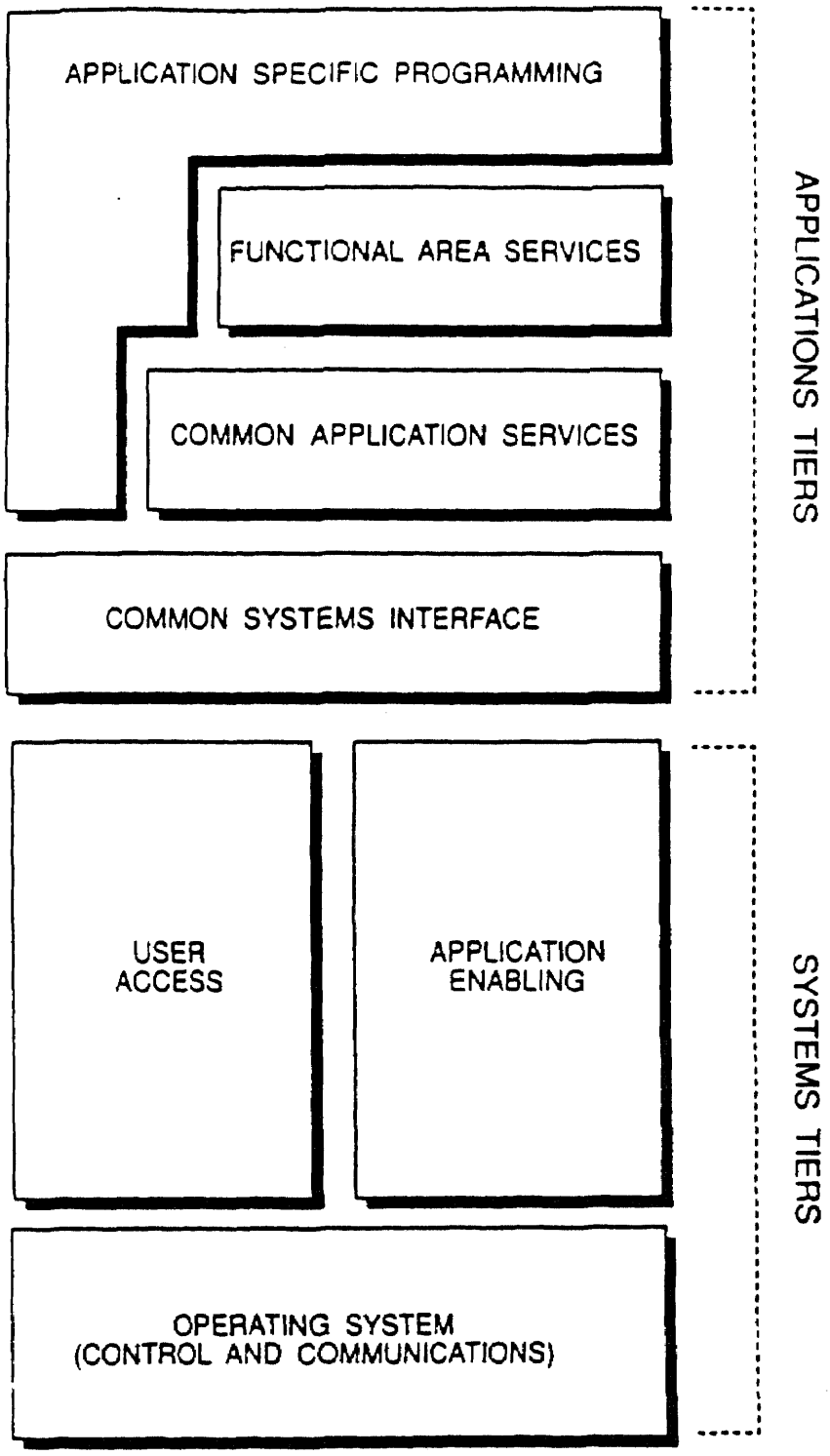

Figure 5.3 SAA - Application Architecture Layers with further Breakdown.

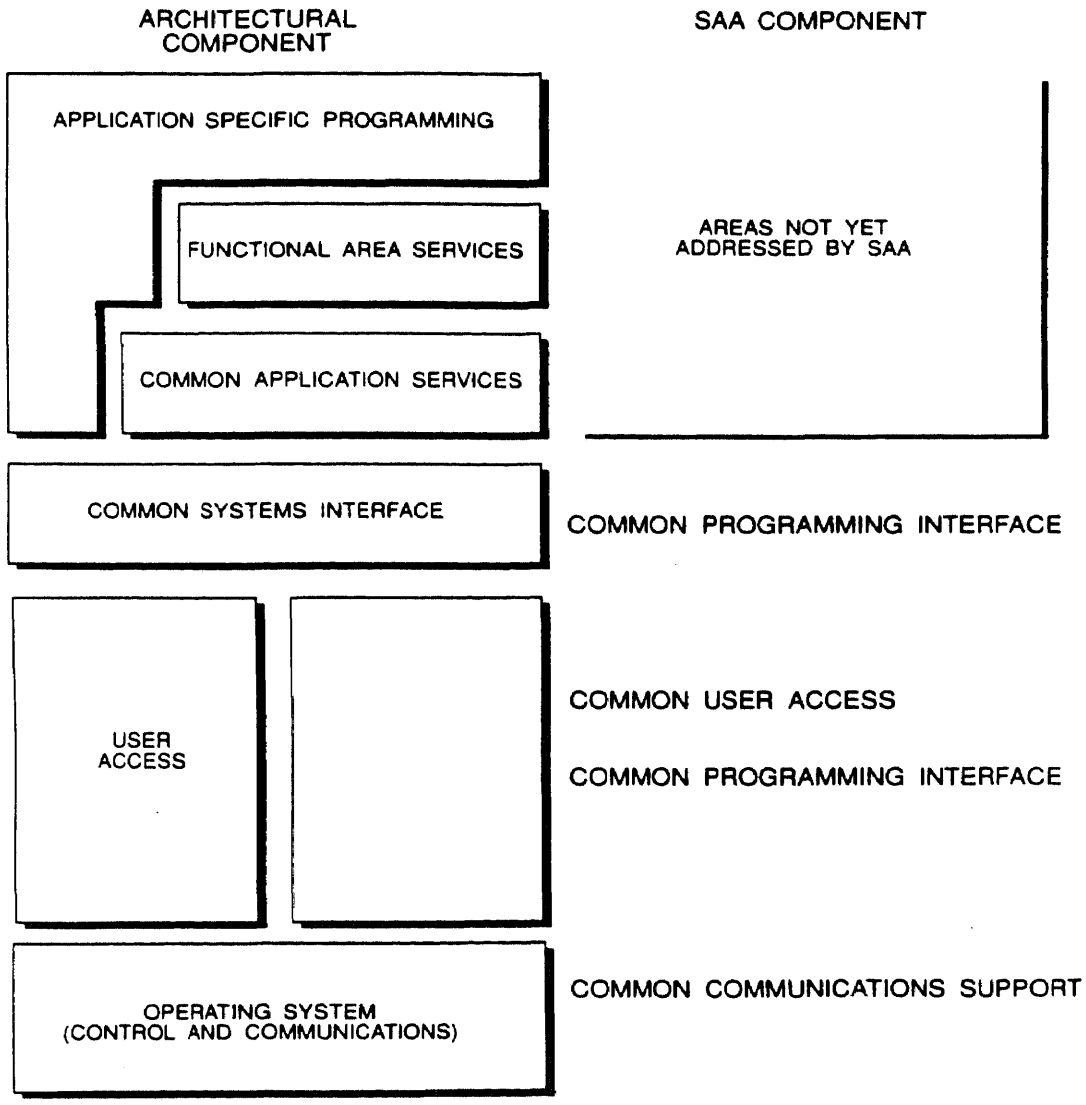

Figure 5.4 SAA -System Application Architecture components with Application architecture layers Breakdown.

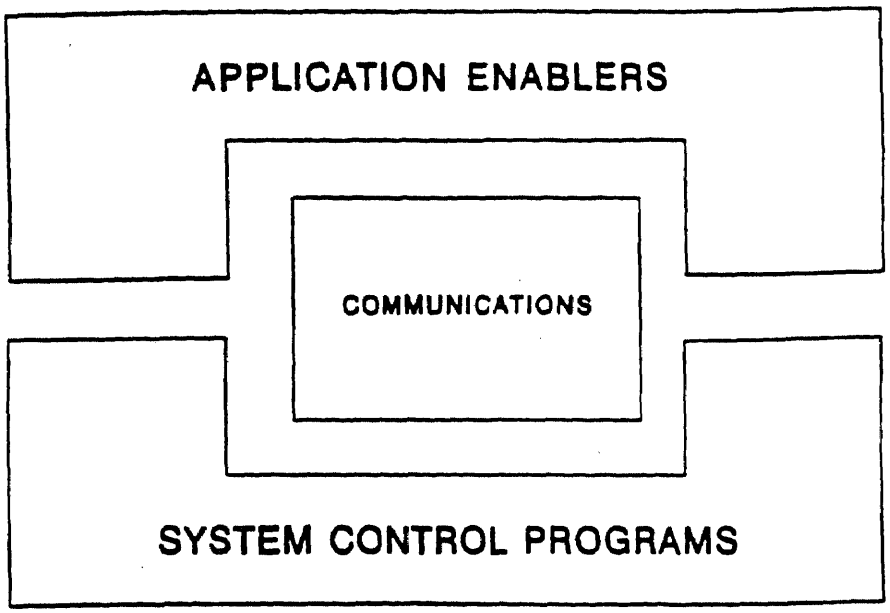

Figure 5.5 SAA software Foundation

CHAPTER 6

BULL'S DISTRIBUTED COMPUTING MODEL(DCM)

6.1 Overview of DCM[12]

As Bull describes : *This Model provides an open and logically distributed systems architecture that will serve as the overall architecture of Bull product offer in the 1990's.* DCM is represented as a set of specifications (interfaces, objects, protocols, rules) to be used in a distributed environment. These components describe a family of services with a common function. DCM is represented pictorially as a three-dimensional model (Figure 6.1). It consists of four horizontal and two vertical components. The horizontal components are:

1 the Applications,

2 the Application Services,

3 the Distribution Services and

4 Communication and System Services.

The vertical components are:

1 the Integrated Systems Management and Security and

2 the Application Development Framework.

We believe that with the representation in figure 6.1, Bull wants to point out that all four horizontal components work in cooperation with each other. For example, Distribution services provides necessary services to applications services. The horizontal components give access to services and bridge the gap between hardware platform and

[12] DCM is a of Groupe Bull.

the end-user.(Figure 6.1).

Thus, DCM provides two different perspectives: The Application developer perspective and the System administrator perspective. However, the ultimate goal is to give more flexibility and quality to end-user applications. Figure 6.1 shows these three different aspects of the model.

The Model offers services for each of these components. These services are easily integrated into the customer environment providing a coherent set of architectural building blocks. They give maximum flexibility while ensuring integrated control. The model offers a high degree of modularity. Using this modularity, the customer, or the systems designer/admistrator can modify and construct applications system as a whole taking into account business needs and ensuring integrated control at the same time.

DCM is an Open Model. Bull claims that DCM is based upon a comprehensive set of **de jure** and **de facto** standards. These means systems from Bull can work effectively with products from other supplier or vendors who follow these standards."Open" here means it is not restricted to Bull's own standards. This provides the basis for interoperability and systems integration.

Bull suggests that the Model possess a key feature, **transparency**. Transparency means that end-users have transparent access to applications on multiple platforms and applications can locate services distributed throughout the network. Each component provides uniquely specified functions and offers its own level of transparency.

6.1.1 Applications Component

The Applications Component is the top level of the DCM(Figure 6.1). The Applications Component represents the applications available to the end user to help meet end user's business goals. These applications are logically different from the underlying services. This component is the most significant to the end user. It contains the applications that can be divided in two general classifications(Figure 6.2):

- the generic and;
- the industry specific.

Generic applications are those, such as office automation applications, which are used generally by all end-users. IMAGE Works[13] is an example of a generic application. This application is mainly to manage compound documents containing the text, image and graphics. It provides a digitalized document filing,retrieval and printing environment. This application has been designed to meet the requirements of specific industries and individual customer. This application deals primarily with the manipulation of office documents. Applications call upon Application Services such as Transaction Processing Service, Mail Service and Print Service. Application services themselves use the Distribution Services to solve network problems. Communication and Systems Services are used to provide lower level network links to hosts. The Integrated Services Digital Networks(ISDN) is an example of such a link.

Industry Specific applications are those which are especially suited to the requirements of a certain industry. The Production Management Application for CIM

---

[13] IMAGE Works is a trademark of Groupe Bull.

Architecture is an example of a specific industry solution proposed. An application running on a UNIX[14] system processes customer orders coming from a sales order processing application running on a Bull DPS 7[15] mainframe or another vendor's platform. The interoperability between the applications is supported by Application Services (File Transfer and/or Transaction Processing Service), Distribution and Communication Services offered by the Distributed Computing Model. During the production process managers will have correct and timely information on the desktop. This will support decision making on problems such as materials, work-in-process, resources management, etc. This enterprise wide information availability is provided by the Application Services that will give access to heterogeneous databases using standard languages.

Either type of application may be developed by the customer. The Distributed Computing Model simplifies the implementation of application solutions to business problems, taking advantage of distributed systems, while integrating existing applications and data as necessary.

6.1.2 Application Services Component

The Application Services Component is the second layer from the top in the pictorial representation of the Model(Figure 6.1). The Application Services Component consists of elements which "enable" applications by processing, displaying or sharing information.

---

[14] UNIX is a registered trademark of UNIX System Laboratories Inc.

[15] DPS 7 is a trademark of Groupe Bull.

These services are grouped logically into three classes according to their primary area.(Figure 6.2):

- End User Services
- Exchange Services
- Live Data Services

End-User Services: The End-User Services satisfy a user or application's need to view, represent, print or store information. The available End-User Services are: Presentation, Print, Scanning.

Exchange Services: These services ensure the exchange of information that is necessary for interactive cooperation between users, groups of users or applications. The available Exchange Services are: Directory, Mail, Electronic Data Interchange.

Live Data Services : With word "Live Data" Bull wants to emphasis on the vital and permanent information of the enterprise. These Services are the services which operate on live data of the enterprise. The available Live Data Services are: Document Filing and Retrieval, Database Access, Transaction Processing, Workflow.

A major aspect of DCM architecture is that the traditional services, such as presentation or print may be performed on a remote server. These services are no longer necessarily performed on the local system. Users need not be concerned with where the services are actually performed. There is no need to worry about any of the underlying components which support access to these functions, no matter where these services such as Print, Transaction Processing or Database may physically be located. In DCM, any user within a distributed system can access all of the services offered.

The currently defined generic Application Services of DCM are:

End-User Services	**Exchange Services**	**Live Data Services**
Presentation	Directory	Documents Filling & Retrieval
Print	Mail	Database Access
Scanning	Electronic Data Interchange Service File transfer	Transaction Processing

6.1.2.1.1 Presentation Service

Presentation Service is a set of functions necessary for an application to interact with end users. The Presentation Service presents a window and icon based interaction scheme on display for all applications which utilize it. Any end user applications requiring easy to operate, "point and click" kinds of interfaces will use this service.

The Presentation Service is able to interface with existing applications as they run on existing machine such as any mainframe(IBM) and using a consistent Graphical User Interface, offers a new attractive presentation on other environments like DOS Windows 3.0 workstations without any modification of the mainframe application. These new interfaces are generated using sophisticated, object-oriented, interactive tools for rapid prototyping and application deployment.

Bull in partnership with other industry leaders is developing a Distributed Dialog Manager. This function will deliver a set of high level semantics to the application

developer. These application interfaces will provide for transparent heterogeneous distribution of application logic and presentation in any network environment supported by DCM. The application interfaces that are created for the Presentation Service are forward compatible with the Distributed Dialog Manager. Application developers utilize API's(Application Programming Interface) specific to the platform that will provide the presentation. The following table relates the interfaces to the platforms:

PLATFORM	**PROGRAMMERS I/F (API)**	**SUPPLIER**
DOS workstation	Windows 3.0 (*)	Microsoft
UNIX based-	X Window (*)	MIT
Workstation &		
X-Terminal	Motif 1.1 (*)	OSF
OS/2 Workstation	Presentation Manager	Microsoft
All the above	Dist. Dialog Manager	New industry standards

(*) and subsequent versions

6.1.2.1.2 Print Service

The Print Service provides the end user or an application with the ability to send information to a printer. The printer could be local or remote. The Print Service handles the print request and respond on completion of the action. Each printer is able to be shared by several users. The printer is managed through a management interface. The Print Service complies with the ECMA(European Computer Manufacture's Association) Distributed Print proposals and ISO(International Standardization Organization) model

for print and is based on the MIT(Massachusetts Institute of Technology) Palladium technology. This service supports three privileged formats; Standard Device Protocol (SDP)[16] Mathilde, Postscript and Intelligent Printer Data Stream (IPDS); as well as any transparent print formats. Following advantages are offered by this service: All applications have transparent access to all printers in the enterprise. Users have transparent access to all available printers from a single view, without individual logins. This service provides print capability for any application (office information, transaction processing), running on any workstation or server.

6.1.2.1.3 Scanning Service

Scanning Service allows an application to capture information from documents, including paper, microfilm and other media. The Scanning Service accepts input from devices connected to remote servers or stations. The Scanning Service supports compression according to various CCITT[17] algorithms (GIII, GIV) and Optical Character Recognition (OCR). The recognition can be applied to specific zones defined by the user. It separates zones of texts from graphic zones. The Scanning Service also provides automatic indexing. This service includes support for all the common standard formats (ISO A0 to A4, ANSI A to E) and resolutions from 200 to 400 dpi. The following advantages are offered by this service: Compactness - Documents in their native form are compressed and recognized when they contain text, to become processable by computer applications.

---

[16] SDP is a trademark of Groupe Bull.

[17] Commite' Consultatif International Te'le'graphique et Te'le'phonique

Efficiency - When OCR is applied to the image, automatic indexing can be performed, increasing the efficiency of the retrieval of documents by direct search. Integration - The resulting image can be merged with other electronic documents produced by OA (Office Automation) tools, to constitute electronic folders which can be processed by higher level applications.

6.1.2.2.1 Directory Service

The Directory Service is intended to give access to user-oriented information about the user's information system environment. This service may be used in several different ways:

Coupled to a mail facility - It provides addresses, location, phone number, fax number of a person or a group of persons. It can be used as a private directory in an Office Automation environment.

Coupled to a software package - The Directory Service provides location of equipment (e.g. room number) within buildings.

Coupled to a user application - It can locate information processing objects like computers, printers, files or other applications.

The user can access this service through a standardized interface called XDS (X/OPEN[18] Directory Service). Based on the X.500 directory, it allows creation of objects, listing of objects, modification of entries and object search based on specified criteria.

---

[18] Consortium of computer vendors developing a standard definition of the UNIX operating system.

Following advantages are offered by this service: Openness- The Directory Service offers full X.500 functionality. Therefore the interoperability with OSI systems is guaranteed. Portability - The Directory Service is accessed through an X/OPEN interface that guarantees portability across a wide range of equipment and easy adaptation to various environments. Modularity - The user can decide to open different directories by type of objects manipulated or by type of functions.

6.1.2.2.2 Mail Service

The Mail Service allows the exchange of information between correspondents (end user or applications). Users and applications may create, read, send, receive, forward and otherwise manipulate information in the form of electronic mail messages. It is possible to operate on these messages in their original format. The Mail Service is based on X.400[19] recommendations for Message Handling Systems. The main functions of the Mail Service are:

- message submission,
- format conversion,
- message delivery.

Applications can access the Mail Service through standard interfaces defined by X/OPEN (MT-API, MA-API).

Following advantages are offered by this service: Openness - The Mail Service is based on international standards. This gives access to a wide range of message

[19] ISO designation for definition of electronic mail protocol.

handling systems in a heterogeneous environment. Flexibility- The Mail Service can be adapted to an individual, a group of people or an entire enterprise. compatible- Coupled with Fax and Telex services, the Mail Service can reach any subscriber's equipment world-wide.

6.1.2.2.3 Electronic Data Interchange Service

Electronic Data Interchange (EDI) is the electronic transfer of business information between applications in a format conforming to a standard. The information represents standard business documents, such as invoices, purchase orders, price quotations, shipping notices etc,. EDI involves the transmission of data in one of several standard formats. The components of the EDI Service are:

- Translation and generation of formats
- Trading Partner Management (profiles and connectivity)
- Document tracking
- Data Mapping
- Scripting facilities

The supported standards for EDI formats are:

- EDIFACT for European and International Networks
- ANSI X.12 for Canada, U.S, Australia and parts of the Pacific.
- TRADACOMMS for the UK
- Any interprofessional standards can be integrated into EDI format facility.

Following advantages are offered by this service: Efficiency - Using the EDI

Service, the user will save the time required to get and store information within the enterprise. It will avoid possible errors of coding-decoding. Competitive tool - The EDI Service improves the customer service. It increases the productivity and improves the management control.

6.1.2.2.4 File Transfer Service

The File Transfer Service provides the ability to move files between heterogeneous systems. It uses the OSI File Transfer Access and Management (FTAM) standard. This standard is widely implemented on the platforms of different vendors. This standard has the advantage that the files on the different systems need not have the same format. The File Transfer Access and Management (FTAM) translates the local and specific file system formats to an internationally agreed interchange format.

The service is available to applications through the FTAM API. Facilities are provided which allow an end-user to initiate and control the movement of files, as well as providing filestore management functions such as remote creation or deletion of files. The File Transfer Service works in conjunction with the Distributed File Service of DCM(But it differs from the Distributed File Service). The File Transfer Service provides an international standard access facility to foreign systems outside of the distributed environment.

Following advantages are offered by this service: Openness - The File Transfer service is based on OSI protocols. This can interoperate with all other systems implementing the OSI FTAM standards. Capacity - The File Transfer Service allows bulk

file transfer. 3rd Party Transfer - The File Transfer Service allows authorized users to initiate file transfers between remote locations.

6.1.2.2.5 Document Interchange Service

The Document Interchange Service provides generalized conversion capabilities, based upon a preferred standard format. It ensures freedom from documents formats for exchange between heterogeneous systems. DCM specifies the ISO Office Document Architecture and Interchange Format (ODA/ODIF) as the preferred unique format for:

- public document storage,
- long term document storage,
- document exchange across networks.

The following ODA profiles as defined by standardization bodies (EWOS, ISO, NIST, CCITT) are taken into account:

- Level 2 (Q 112, FOD 26) provides for the interchange of multimedia documents between advanced word-processing systems within an integrated office environment.
- Level 3 (Q 113, FOD 36) a level 2 enhancement.

Following advantages are offered by this service: Efficiency - Maximize information flow regardless of document format.

6.1.2.3.1 Document Filing and Retrieval Service

The Document Filing and Retrieval(DFR) Service is a foundation of applications such as Document Management of Office Information Systems. It includes two classes:

- Folder Management

- Content Management

Both classes are able to deal with large documents of various types integrating text, graphics, images. It comply to various document standards such as ODA, SGML(ISO, Standards Generalized Mark-up Language) and CGM (ISO, Computer Graphics Metafile) or de-facto standards such RTF(Microsoft, Rich Text Format), HPGL(Hewlett Packard Graphics language), PCX (Microsoft, Paint Brush File Formats) and Postscript(Adobe, Page Description language as defined by Adobe).

The Following advantage is offered by this service: Ease of use - The DRF Service provides easy location of documents.

6.1.2.3.2 Database Service

The Database Service provides applications access to private, departmental, or enterprise data. DCM specifies a SQL interface for applications accessing databases. This standard interface is mandatory for applications requiring distributed and transparent access to heterogeneous databases. Using this interface, applications can access multiple databases. It is independent from the hardware they reside upon and independent of the vendor supplied Database Management System (DBMS). These databases may be:

- Distributed among various systems (GCOS, POSIX, IBM/DB2, DEC)

- Of various structures and organization (e.g., CODASYL, Relational, etc.)

- From various vendors (e.g. Oracle, Ingres, Bull, IBM)

Following advantages are offered by this service: Competitiveness - The Database

Service provides the ability to build new applications. These applications utilize existing enterprise and non-enterprise data with new data. Thus it provides the ability to implement competitive information systems. Increased Data Consistency - By using the Database Service, applications architects can reduce replication of data. Thus it minimize problems of synchronization and consistency. Modularity - The standard preferred interface provides independence of applications from the organization of the underlying data and reduce the interdependence of applications. Openness - This standards - based interface also opens the door to a larger variety of applications developed by ISV's and reduces the technical complexity of application systems.

6.1.2.3.3 Transaction Processing Service

The Transaction Processing Service provides the specific functions necessary to implement applications that change the state of the enterprise in real-time. On-line transaction processing (OLTP) applications require Transaction Processing services that insure that transactions are completed completely or not at all. Completion of transactions must have certain properties. The Transaction Processing Service insures that transaction applications are: Atomic, Consistent, Isolated and Durable (ACID):

- Atomic, meaning that all pieces of the transaction are processed or none at all.

- Consistent, in that if some part of he transaction is not completed, all parts of the system effected by the transaction are left in their original state.

- Isolated, such that while the transaction is in process the shared resources of the system are not accessible by any other transaction.

- Durable, so that in case of failure the transaction is not lost.

The Transaction Processing Service provides resource management to insure coordination of commitment and specialized communications capability to support on-line transaction processing applications that are distributed.

Following advantage is offered by this service: Performance and Reliability - The Transaction Processing Services gives high performance and reliability.

6.1.3 Distribution Services

Distribution Services is the third horizontal component in DCM (Figure 6.1). Distribution Services provide the transparent distribution of processing functions across the network. They allow access to a wide variety of services and applications. The user does not need to be concerned about the location of the services or particularities of the platforms which support them. DCM is based on the client server model[20]. This model standardizes a way of structuring processing into two complementary but different parts

the consumer of a service **(client)** and;

the service provider **(server)**.

They may be distributed over a network. Within the DCM environment Distribution Services are the key enabling technologies of client-server computing. It provides for the transparent distribution of Application Services.

The services offered in the Distribution Services Component are (Figure 6.2):

---

[20] A Client- Server Model is an Asymmetric Computing Model, using two separate and logical entities, working as "front-end"(client) and "back-end" (server) components in a cooperative way, with related general task. The Client requests information or action, the Server replies the request.

Distributed File

Naming

Remote Procedure Call (RPC)

Timing

These services are globally available services and need not be installed on every system in the distributed environment. It is a core set of services globally available and accessible from any workstation or application, regardless of location and network. The client server model offers flexible and transparent extension of service. Addition of a server, or cooperating external servers, do not modify customer or application visibility of the service. Access to OSI(Open Systems Interconnection) services or public services may be achieved easily at the server level without impacting client applications.

6.1.3.1 Distributed File Service:

The Distributed File Service is intended to allow a high degree of data-sharing capability throughout the network. The Distributed File Service is essentially integrated with the user's own system and so accessing global files seems as simple as accessing local files. In addition, the Distributed File Service provides file location transparency, high availability and a uniform name space. The distributed file system used in DCM is the Distributed File System (DFS) from OSF (Open Software Foundation) DCE. It offers:

- a crash recovery mechanism,
- a high performance physical file system,
- a protocol exporter, making it possible for an NFS(SUN's Network File System)

client to access DFS servers,

DFS follows the client server model. It supports any number of clients and serve, organized in an administrative and operational domain (cell).

Following advantages are offered by this service: High Performance - The Distributed File Service incorporates replication mechanisms which allow for several copies of commonly accessed files to be available in the network, thus reducing access time. Interoperability - The Distributed File Service is network independent. Security - The Distributed File Service includes the same user authentication and access authorization as the Distributed Computing Security Services.

6.1.3.2 Naming Service :

The purpose of the Naming Service is to map user-oriented names or objects of interest in a distributed computing environment into computer-oriented entries in a distributed database. The objects to be named include such things as countries, organizations, persons, groups, organizational roles, computers, printers, files, processes and Application Services. The clients of the Naming Service span a wide range, from other services comprising the distributed environment, such as management programs, to applications. through the use of the RPC Service, the Naming Service works in LAN as well as in WAN (Wide Area Network) environments. The components of the Naming Service are based on OSF DCE technology and comprise:

- "local" naming service
- "global" X.500 service

- XDS and XOM API conforming to X/OPEN specifications

Following advantages are offered by this service: High Performance - The Naming Service provides high performance through use of replication, the creation and maintenance of multiple copies of critical data, by allowing names to replicated near the people who use them. Modularity - the Distributed Computing Model is in part due to the scalability of the Naming Service. The Naming Service offers the ability to easily add domains, local or remote, thereby accommodating large networks as easily as small ones.

6.1.3.3 Remote Procedure Call (RPC) Service

The Remote Procedure Call (RPC) Service is the heart of a client-server model. It lets programs running locally call procedures implemented on remote systems. The RPC is the mechanism which allows the distribution of access to Application Services. It is used to "export" the API of the Application Service onto other platforms. In this context, RPC is entirely transparent to the programmer using the Application Service. From the point of view of the programmer coding access to a service is implemented via an API, there is no difference between a "remote" access to a distant server and a local procedure call. The actual location of the servers is determined automatically through the Naming Service.

Independent Software Vendors and customers with specific requirements may also use the RPC Service to develop their own applications.The RPC Service is based on OSFs DCE RPC. It includes two major components:

- A remote procedure call facility developed specifically to provide simplicity,

performance, portability and network independence. The RPC Service contains an automatic data conversion that masks the differences between data representations (e.g byte ordering, floating point) on different machines;

- A compiler that converts high-level interface descriptions of the remote procedures into portable C-language source code.

Following advantages are offered by this service: Evolutionary tools - Current applications can be easily modified for network computing. Network independent - The RPC Service uses the XTI(X/OPEN Transport Interface) transport interface from X/Open that hides the protocols used for transmitting data. Internationalization support - The RPC Service can support any type of character sets (e.g Latin, Chinese, Arabic, Japanese)

6.1.3.4 Time Service

Time Service regulates the system clocks throughout the network, so that they closely match each other and provide an accurate time for all distributed processing. Many applications need a single time reference to schedule activity and determine event sequencing and duration. Different components within a distributed environment may obtain the time from clocks on different systems. The Time Service is a software-based service that synchronizes each computer to a widely recognized time standard. It provides synchronization for systems in both local area networks and wide area networks.

Following advantages are offered by this service: Fault-Tolerance - The Time service identifies server with faulty clocks. Managements - The Time service offers a user interface for controlling and monitoring the software.

6.1.4 Communication and System Services

The Communication and System Services are the services which provide the system processing power and perform the transport of data(Figure 6.1).

DCM offers two types of services(Figure 6.2).

6.1.4.1 Communication Services

In distributed computing a major requirement is operation in heterogeneous environments. DCM wants to offer a wide range of interconnection protocols for this such as: OSI, TCP(Transport Control Protocol), SNA. The complete set of communication elements offered on the different Bull platforms is given in the "Interoperability Environment" document. The Communication Services provide for the transparent transport of data from one position to another. Communication may be either connection-oriented or connectionless. There are three major types of communications in DCM:

32-XTI (X/OPEN Transport Interface)

CPI-C (X/OPEN Program-to-program communication)

PC Communication and Terminal Emulation

All these three communications have different benefits. The XTI Service provides a transport interface covering both ISO and TCP transport protocols in connection oriented and connectionless mode. The basic CPI-C interface allows a program to dialogue synchronously with another program in peer-to-peer mode. The functions provided are: Open a dialogue, Read, Send, Close a dialogue. PCs are connected to

servers or Mainframes through various means. For different case DCM offers different communications. For example, PCs to IBM systems use an interface called HLLAPI and emulate a 3270 terminal. PCs to UNIX servers or OS/2 servers use Microsoft LanManager[21] interfaces called Mailslot and NamedPipe. These interfaces give access to all the work-group applications.

6.1.4.2 System Services

System Services provide the basic functions required by applications and other services to interact with the system platform. Systems Services include functions required to allocate memory, create and manage processes and to manage real-time events. The different applications architectures supported DCM have various sets of Systems Services, tailored for the class of applications best supported by the platform. Systems Services are a core part of the applications architecture, which is defined as the set of interfaces available to applications programs. Bull's offer includes open applications architectures :DOS, OS/2 and UNIX based systems that conform to industry de facto and de jure architectures such as X/OPEN. In addition, Bull supports applications architectures in the GCOS[22] environments which provide enhanced functionality required for certain classes of applications.

6.1.5 Integrated System Management and Security

---

[21] LanManager, MailSlot are the trademarks of Microsoft Corporation.

[22] GCOS is a trademark of Groupe Bull.

The Integrated System Management (ISM) and Security Component provides the services to maintain a secure and consistent environment for the distribution of all functions throughout the network. This component of the Model provides distinct architectural frameworks for Integrated System Management (ISM) and for Security. These two functions, ISM and Security, fit together as they manage the same resources but in different ways.

The horizontal components of DCM are the resources within the model. These components are manipulated as objects. The links between these resources ensures the link to the object oriented representation. Within DCM, key functions are implemented as common services. It includes access to the Information Bases. Specialized applications perform management security tasks. It uses advantage of the appropriate common services via programmatic interfaces. ISM and Security applications also use services such as Directory, Presentation and Remote Procedure Call. The security administrator or system manager sees his domain of the enterprise information system through the window of his graphical station with the applications needed to do his job. This component is sub-divided into two parts.

6.1.5.1 Integrated System Management

A major objective of the Integrated System Management (ISM) is to identify a comprehensive framework of management of systems ranging from stand-alone systems to systems in a distributed environment. This framework is supported by appropriate tools, to meet extendibility, maintainability and adaptability in the four following areas:-

Network management

System management

Spool management

Storage management

Integrated System Management is implemented by applying standards based technologies in three major areas:

6.1.5.1.1 Management Applications

These individual applications are used to perform management tasks. Management applications free the user from the detailed steps necessary to perform these tasks. It also ensure the integrity of management data. User registry is a file system configuration. User registry and print spooler management are examples of management applications. These applications manage objects through the use of clear and concise interfaces to common management services. Object-oriented techniques are introduced to facilitate management application development. These applications are provided by Bull, Independent Software Vendor, or the customers themselves.

6.1.5.1.2 Common Management Services

Common Management Services support the coherent management framework by making programming interfaces available which are used by management applications. They include:

-Sieves, for the efficient reduction of information to the relevant facts concerning a

network or system event.

- A script facility, which allows structured automation to be applied to a series of management tasks or operations,

- An inference engine, which enables customers to reap practical benefits from the application of Artificial Intelligence. It facilitates the creation of Knowledge Based Systems to manage or assist in the management of enterprise systems.

- Event services which provide a coherent point of coordination for enterprise related events. The Common Management Services simplify the development of portable management applications. Management Information Base (MIB)Services allow management applications to manipulate management information. These services are presented through a programmatic interface and use relevant standards such as OSI CMIS/CMIP and Internet SNMP.

6.1.5.1.3 Managed Objects

A managed object is a representation of resources within the computing environment. An example of a system resource is a file, which is represented and managed by its associated management object. Other examples of sources include devices, Print and Mail Services, users and end-user application software.

For the end user or administrator, the Integrated System Management improves the reliability and availability of systems and networks, increases the portability of user skills between different platforms.

The DCM security services adhere to US DoD standards. It includes three parts.

Secured Objects

Common Security Services

Security Applications

Secured objects represents a subset of the objects managed by ISM and include :massages, databases, communication paths etc. Common Security Services support the security framework by presenting programming interfaces used by security applications.

6.1.6 Application Development Framework

Application Development Framework includes a variety of tools and services required by the developer for a complete applications development environment. Application Development Framework provides the application developer with a set of standard programming languages and development tools. Access to all components of the distributed environment is provided through open standard Application Programming Interfaces(APIs). Moreover, it provides an environment for integrated Computer Aided Software Engineering(CASE) and an Integrated Project Support Environment (IPSE). The Application Development framework, seems to be still under research. details about it.

6.2 Mapping DCM to GenSIF

DCM is a model, which offers a distributed computing strategy. This model provides an open and logically distributed systems architecture that will serve as a overall architecture of the bull products. We map DCM into GenSIF. We first list GenSIF guideline concept for each activity of Mega-system task and then check how much this guideline is covered in DCM.

6.2.1 Domain analysis

6.2.1.1 *Suggested GenSIF guideline :*

To define the domain boundary.

To identify and classify the domain information.

To represent acquired information in the domain model.

To evaluate and validate the domain model.

To update and refine the domain model.

DCM : DCM does not offer any guidelines for domain analysis. The top horizontal component of the model is the component, which deals with applications available to end-user. DCM addresses generic and industry specific application but no guidelines for domain analysis and domain modeling are given.

Summary : DCM does not cover domain analysis as a part of the architecture, but DCM strategy seems to support the concept that domain analysis and domain modeling should be performed before any specific development.

6.2.2 Integration architecture

6.2.2.1 ***Suggested GenSIF guideline :*** To provide standards and guidelines for system building blocks.

DCM: The Model is intended to provide a flexible framework for the architectural building blocks of enterprise computing. However, the model does not list guidelines and standards for conceptual architecture within the model. DCM recommends to employ de jure and de facto standards, so that systems, following DCM can work effectively with products from other suppliers. Moreover Independent Software Vendors can add products within this environment using these standards.

The Model offers direct services and not the guidelines or standards for system building blocks.

6.1.2.2 ***Suggested GenSIF guideline :*** General guidelines for system decomposition.

DCM: DCM expects that application system should be decomposed, in order to take the advantage of the services DCM offers. However, DCM does not give specific guidelines or rules for applications system decomposition within the model. The top most horizontal component, applications, needs to utilize the other lower horizontal components. This implies that, system decomposition is inevitable for DCM.

6.3.2.3 ***Suggested GenSIF guideline :*** Standards for interfaces(internal and external).

DCM: DCM provides standards for internal interfaces. However, it is not mandatory to apply only these standards for internal interface.DCM encourages to use UVTI(Bull's

specification) for internal interfaces . DCM is based on the client server model. This model standardizes a way of structuring processing into two complementary but different parts the consumer of a service (client) and the service provider (server).

For external interfaces DCM insist to use X/OPEN and OSF standards. At this point DCM does not comply to use its own standards. We believe that DCM wants to be an Open Model and therefore it does not restrict developer for its own standards. DCM encourage to use following standards in priority for interface.

1. de jure standards,
2. consortium products,
3. de facto standards,
4. Bull specific interfaces.

6.1.2.4 ***Suggested GenSIF guideline :*** A communication model.

DCM: DCM does not offer a conceptual communication model on the application level as a part of DCM. DCM refers to necessary communication tools.

6.1.2.5 ***Suggested GenSIF guideline :*** A data handling model.

DCM: DCM does not offer a conceptual data handling model as a part of DCM. DCM offers Electronic Data Interchange (EDI) services that allow to build standardized data exchange and provides tool-level interfaces to different database products. Within the Bull Distributed Computing Model, effective and standardized exchange of messages, data and documents is provided by these services.

6.2.3 The Technical Infrastructure

6.2.3.1 ***Suggested GenSIF guideline :*** Communication tools.

DCM offers a wide range of communication tools on different levels. With communication services, DCM offers a wide range of interconnection protocols such as OSI, TCP, SNA. Distribution and application services build their communication capabilities on this basis.

6.2.3.2 ***Suggested GenSIF guideline :*** Database Components.

DCM offers Database components under various services. Application services covers two database services.

Exchange services

Live data services

Exchange services ensure the exchange of information and data that is necessary for interactive cooperation between users and applications. Exchange services offers Electronic Data Interchange (EDI). EDI is the electronic transfer of business information between applications in a structured format conforming to a standard. EDI involves the transmission of data in one of several standard formats. In most instances, data from installed applications are translated to the standard prior to transmission, otherwise a third-party service will carry out on-network translation. It is usually necessary for the data to be translated again into formats recognized by a trading partner's computer applications.

Live data services offer the Database Service. The Database Service provides

applications access to private, departmental, enterprise, or intra-enterprise data. These databases may be:

- Distributed among various systems (GCOS, POSIX, IBM/DB2, DEC)
- Of various structures and organization (e.g., CODASYL, Relational, etc.)
- From various vendors (e.g. Oracle, Ingres, Bull, IBM)

6.2.3.3 ***Suggested GenSIF guideline :*** User-interface generator.

DCM offers End-User Services such as

Presentation

Print

Scanning

The Presentation Services are sets of functions necessary for application to interact with end users. The Presentation Service presents a window and icon based interaction scheme on display for all applications which utilize it. The Presentation Service, using a consistent Graphical User Interface, offers a presentation on DOS Windows 3.0 workstations without any modification of the mainframe application. These interfaces are generated using sophisticated, object-oriented, interactive tools for rapid prototyping and application deployment. This function delivers a complete set of high level semantics to the application developer. These application interfaces are intended to provide for transparent heterogeneous distribution of application logic and presentation in any network environment supported by the Distributed Computing Model. The application interfaces that are created for the Presentation Service are forward compatible

with the Distributed Dialog Manager to insure no loss in the appearance and behavior of their applications. Application developers utilize API's specific to the platform that will provide the presentation.

6.2.4 Conclusion

No guidelines, standards or rules for domain modeling are listed in DCM description. It seems at this point to us that DCM has to add Domain analysis as a part of the Model itself. DCM relies on an Application development component which speaks about types of systems.

DCM has very little to offer for integration architectures. It does not provide conceptual architectural guidelines. DCM wants to be an "Open Model" and tries to include existing standards (national/ International) for integration on the tool-level only. DCM relies more on the services that it offers. We consider it at the technical infrastructure level in GenSIF.

6.3 Mapping DCM to the OSCA architecture

We use the GenSIF guideline concepts for each activity of the Mega-system task to compare these two architectures.

6.3.1 Domain analysis

6.3.1.1 *Suggested GenSIF guidelines :*

To define the domain boundary.

To identify and classify the domain information.

To represent acquired information in the domain model.

To evaluate and validate the domain model.

To update and refine the domain model.

None of these architecture, OSCA or DCM, directly offers any of the above mentioned guidelines. However, the decomposition strategies of both architectures demonstrate that domain analysis is needed. OSCA and DCM can be mapped as a stage after domain analysis in GenSIF.

6.3.2 Integration architecture

6.3.2.1 *Suggested GenSIF guideline :* To provide standards and guidelines for system building blocks.

DCM does not stress to use any particular rules or guidelines for building blocks. DCM uses a component based approach throughout the model but fails to give explicit importance to it on the application level. OSCA, on the other hand, provides a conceptual

and abstract level of guidelines and rules for system building blocks within the architecture. These guidelines and rules can be utilized for any further design and development.

6.3.2.2 ***Suggested GenSIF guideline :*** General guidelines for system decomposition.

The OSCA architecture explicitly gives a general guideline for system decomposition. DCM dose not provide necessary guidelines for system decomposition. The OSCA architecture wants that systems should be decomposed into three major areas such as user access, processing, database. DCM speaks about two types of applications systems, generic and industry specific, But does not emphasize to decompose an application in any specific way.

6.3.2.3 ***Suggested GenSIF guideline*** : Standards for interfaces(internal and external).

DCM does not directly list rules and guidelines for internal interfaces on the conceptual level. OSCA on the other hand, gives explicit guidelines for internal interfaces. For external interfaces OSCA is at a developing stage and does not offer in current publications OSCA (14) anything about it. DCM includes end-user services and presentation support without speaking about concepts which would go beyond the tool-level.

6.3.2.4 ***Suggested GenSIF guideline :*** A communication model.

DCM does not offer an application communication model as a part of the architecture.

The OSCA architecture offers a communication model.

6.3.2.5 ***Suggested GenSIF guideline :*** A data handling model.

A conceptual level data handling model is not covered in DCM. OSCA gives conceptual data layer building block principles for data handling.

6.3.3 The Technical Infrastructure

6.3.3.1 ***Suggested GenSIF guideline :*** Communication tools.

DCM offers a most of communication tools as dicussed above. The OSCA architecture does not offer communication tools at this point.

6.3.3.2 ***Suggested GenSIF guideline :*** Database Components.

DCM offers Database components under various services. The OSCA architecture does not offer database components such as programming languages and DB-systems.

6.3.3.3 ***Suggested GenSIF guideline :*** User-interface generator.

DCM provides user-interface generators in its End-User Services. OSCA does not support this notion.

6.3.4 Conclusion

The Distributed Computing Model is modular and open. Bull wants to provides products as components which integrate into the enterprise information systems. We believe that

this model is at a developing stage and not complete. Many services are not ready for use. DCM is a strategy which offers services to integrate existing and new systems. The Distributed Computing Model is not, as its name points out, a conceptual model. It is more a set of services that is supposed to perform integration of distributed computing.

The OSCA architecture, on the other hand, is a conceptual architectural model. The OSCA architecture principles are more conceptual and comprehensive. We believe that these architecture principles can be utilize for distributed computing. For example, OSCA lists building blocks principles in detail. Using these principles one can design a new system which can be easily integrated in a distributed environment using services that DCM offers(Figure 6.3). OSCA strategy is a top-down approach. First ensuing conceptual principles, a modular and flexible system can be developed. After that, using suitable tools one can integrate this system over different platforms and environments.

The OSCA architecture offers most elements proposed in GenSIF conceptual integration architecture. DCM does not cover a **conceptual integration architecture**. DCM gives only tool level architectural guidelines.

DCM is a bottom-up approach of Bull. DCM does not describe building blocks at conceptual level. DCM does not define constraints and guidelines for building blocks. Bull's strategy is to integrate systems through services. DCM does not involve an explicit conceptual design logic to develop generic distributed systems. DCM is a good technical infrastructure in terms of the services it offers. We believe that DCM can employ a conceptual architecture such as OSCA for its application level. Employment the of OSCA architecture produces a modular and flexible system. This system fits as generic or

industry specific application at top component in DCM. The rest of DCM components would fit for further services. However, the compatibility and how the OSCA architecture can associate with DCM services needs an additional study with internal details of both OSCA and DCM.

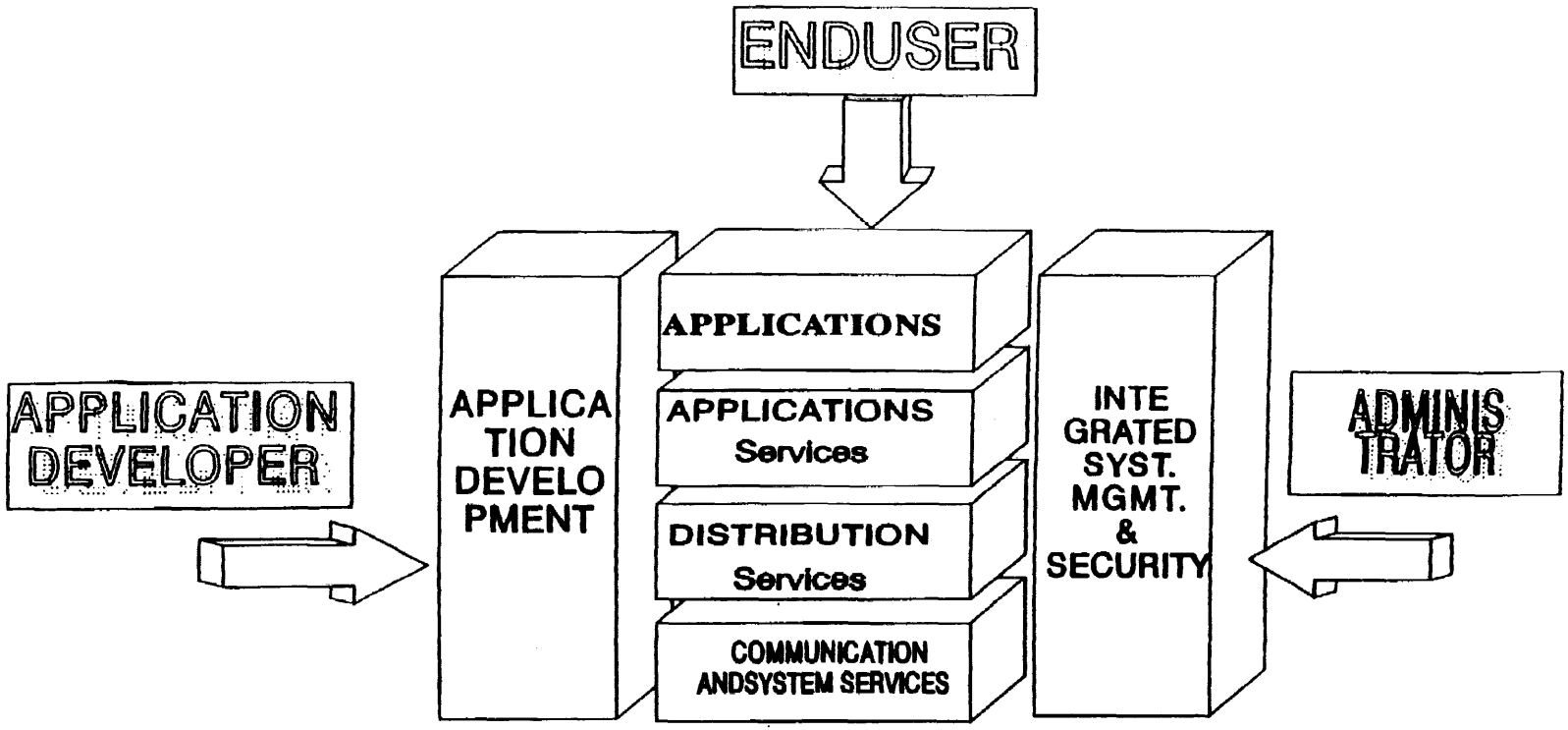

Figure 6.1
Distributed Computing Model(DCM)

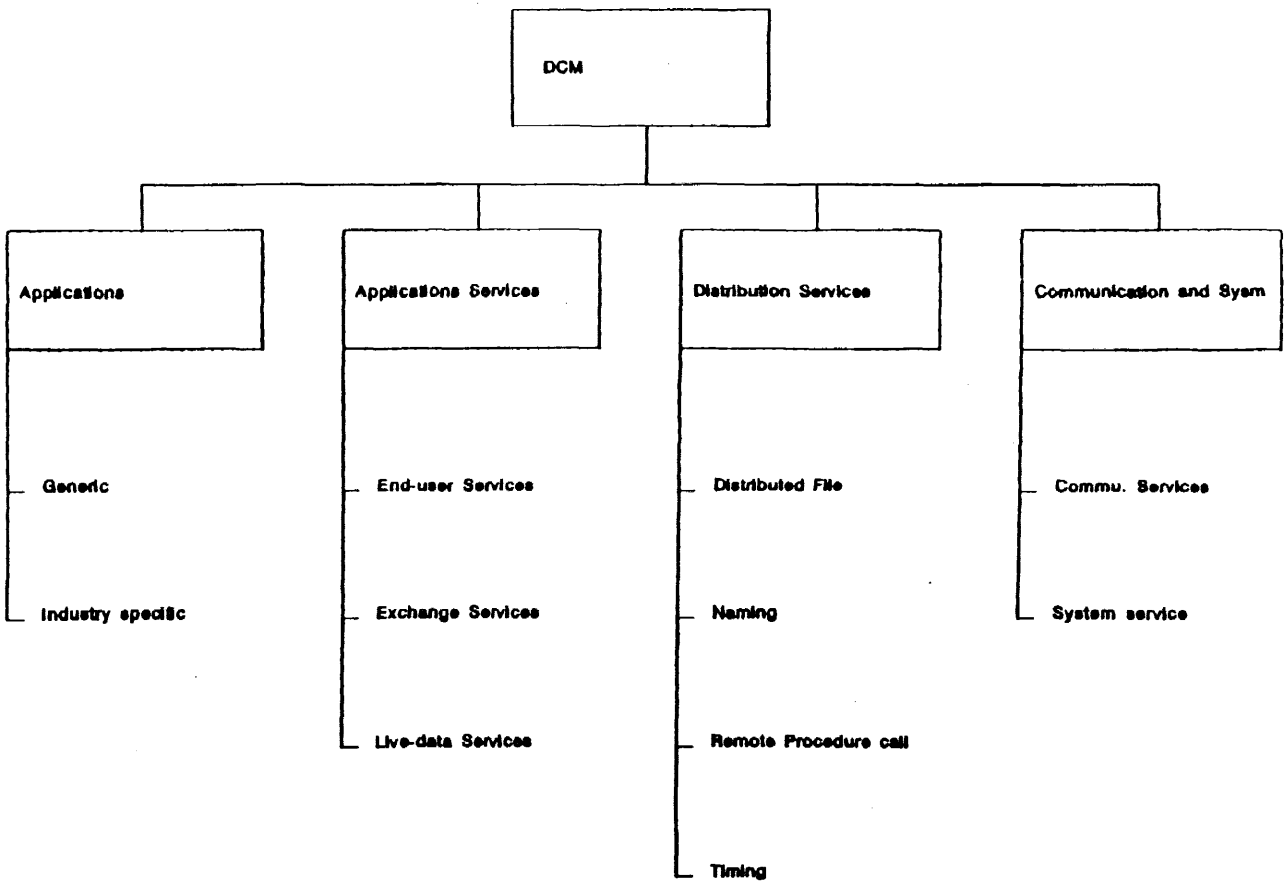

Figure 6.2 DCM - Horizontal components

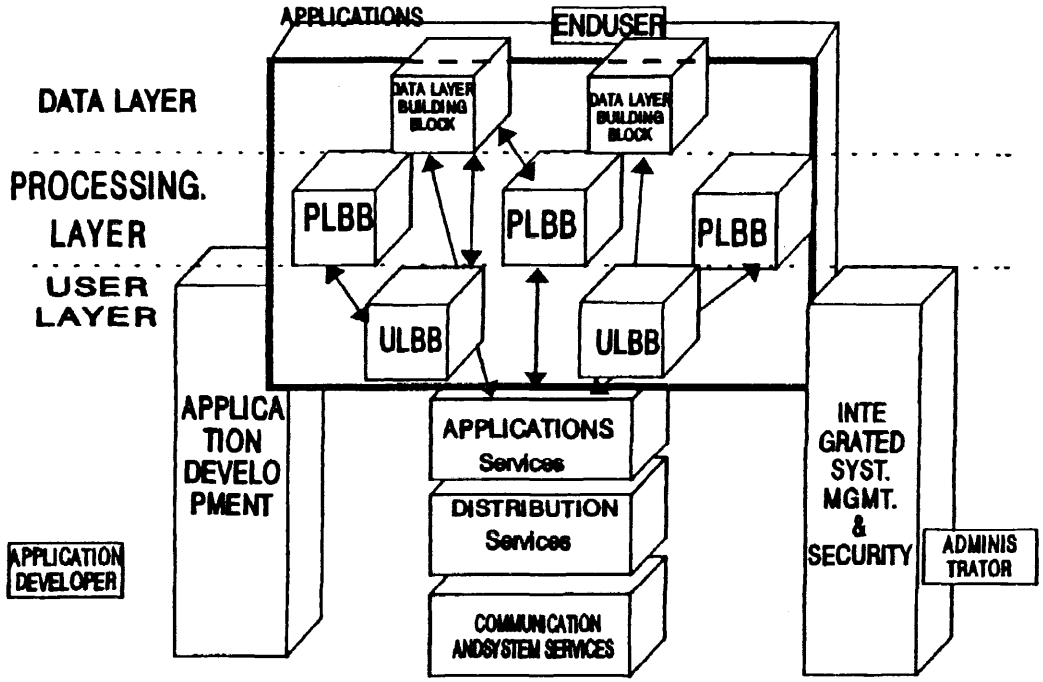

Figure 6.3
OSCA and Distributed Computing Model

CHAPTER 7

CONCLUSION

The GenSIF concepts propose an alternative of the existing post-facto, bottom-up approach to develop an integrated system. GenSIF provides a comprehensive and complete framework for integrated system development. The GenSIF development is a top-down approach. This top-down approach assures integrity from the very beginning on, imposing meta-level control in the form of integration architectures and by pre-defining domain boundaries. This provides the necessary base for the derivation of a suitable infrastructure. Domain analysis, integration architecture design, and technical infrastructure design are the three main steps of the GenSIF development.

Domain analysis deals with concepts and semantics of an application domain. At present, domain analysis is not seen in any of the discussed architectures (OSCA, ANSAware, SAA, DCM). We believe that for a top-down approach, domain analysis is inevitable. Domain analysis provides not only a basis for semantic integration, but it is also main input to decide on the design of the integration architecture.

A conceptual integration architecture is the core of GenSIF. Our investigations and comparisons suggest that Bellcore's OSCA architecture is the most comprehensive conceptual integration architecture among all discussed architectures. The OSCA architecture relies on an abstract Model of "separation of concerns". The basic rules, guidelines and constraints of the OSCA architecture match with the GenSIF concepts. The OSCA architecture exactly fits GenSIF as one possible conceptual integration architecture.

ANSAware gives a communication model at a conceptual level. ANSAware is developed to support the design, implementation, operation and evolution of distributed information processing systems. It is an implementation of the ANSA architecture which provides a framework of specifications, structures, functions, design recipes and implementation guidelines for the construction of multi-vendor, heterogeneous, multi-domain distributed systems. ANSAware itself does not provide guidelines for an integration architecture. It only specifies that it uses an object-oriented approach to develop its applications. The communication model of the Integration Architecture is described in ANSA but it does not give any guidelines for system decomposition or data handling as in GenSIF.

IBM's SAA provides rudimentary rules, guidelines and constraints for an integration architecture. SAA suggests a different approach then OSCA or GenSIF. It uses a bottom-up approach providing necessary guidelines, rules and standards only for services on the application level, but not for the applications themselves. Our observation suggest that IBM is designing an application architecture, keeping in mind only the technical services it offers.

Bull's DCM is also a set of services rather then a complete conceptual Model. From our point of view, DCM is a structured bottom-up approach. It does not specify only any specific Bull standards but wants to be an "Open Model" and welcomes other national and international standards e.g. OSI and CCITT. We believe that in keeping DCM an "Open Model", Bull possibly avoided to define its own standards. This resulted in a simultaneously positive and negative situation. On the positive side DCM can claim

to be an "Open Model". On the negative side DCM does not define specific guidelines for an integration architecture.

The technical infrastructure is a third component of GenSIF. In order to perform integration at the technical level, the derivation of a technical infrastructure is inevitable. The OSCA architecture does not offer a design or a derivation method for a technical infrastructure. The OSCA architecture suggests that a technical infrastructure is part of system engineering and not of a conceptual integration architecture.

ANSAware offers the means of communications for distributed systems and provides the necessary base for a communications model. However, ANSAware does not offer all services proposed in GenSIF, for example, data handling tools and external interface generators.

SAA is a combination of software and hardware design strategies of a giant "hardware company" (IBM). SAA offers most of the technical infrastructure elements. Most of these offerings are ready-made tools. SAA does not offer a process model for integration as proposed in GenSIF. We consider SAA to be a good example for a technical infrastructure.

DCM offers also a technical infrastructure by providing necessary tool support. However, DCM services are more user requirements oriented. This helps DCM in adapting application architectures like OSCA; even though to comment that, "DCM services fit the OSCA architecture" would be pre-mature at this stage.

Every organization has a healthy degree of self-interest in mind in formulating its strategy. We conclude that hardware manufactures come up with a bottom-up approach,

ready-made tools and services, to survive the diversity that they have created over a long period of time. However, there are developments, GenSIF and others, which work without self-interest to deliver a framework for integration of a seemingly chaotic computing world and there is no doubt that we still have a long way to go to fully implement that strategy.

Throughout all the chapters I attempted to provide as accurate and complete a picture of particular architecture as I found in the publications that organizations have provided to me. The opinions and speculations are, of course, my own, but they are borne from the viewpoint of someone working with a wide variety of information and different publications provided by the different organizations.

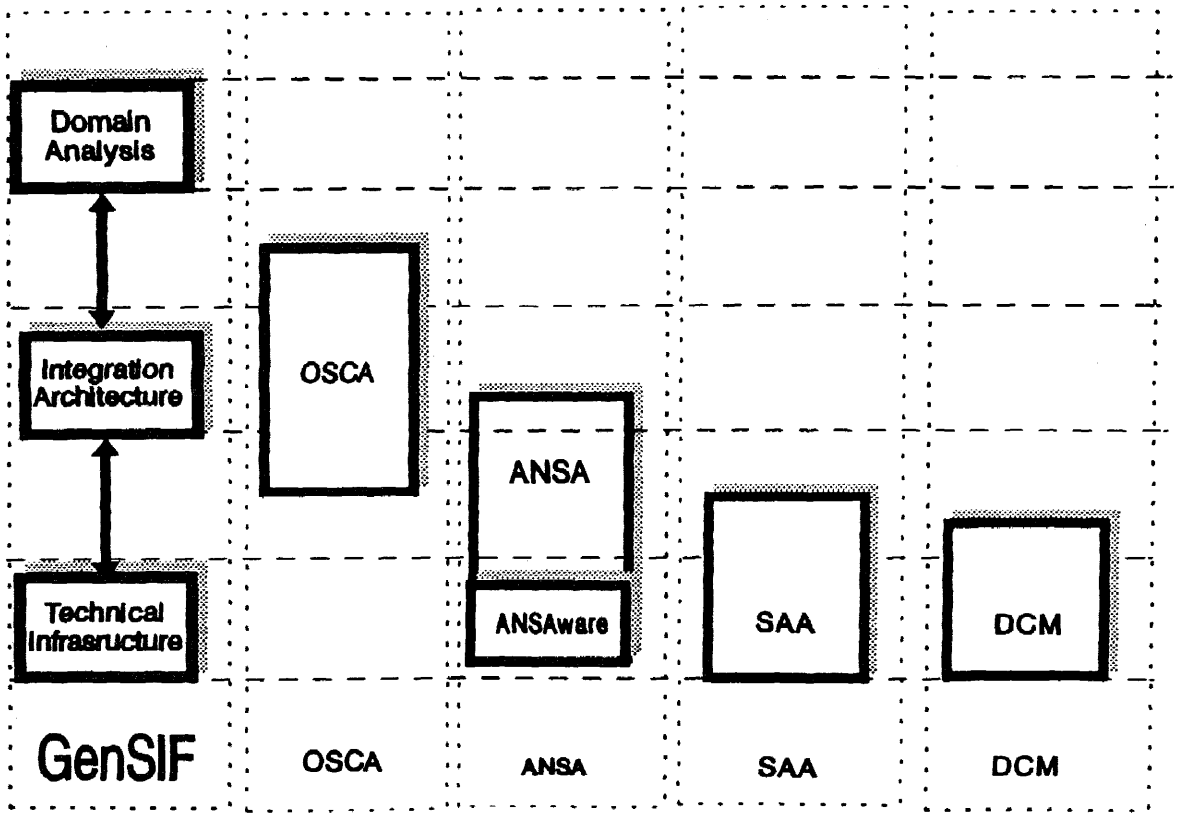

Figure 7.1 Mapping of Architectures

REFERENCES

1. ANSA: An Engineer's Introduction to the Architecture, Architecture Projects Management Limited, UK, Release TR.03.02, November 1989.

2. Boehm, B. W. "Software Engineering." IEEE Transactions on Computers, Vol. 25, No. 12, December 1976, pp. 1226-1241.

3. Davis A. M., E. H. Bersoff, and E. R. Comer. "A Strategy for Comparing Alternative Software Development Life Cycle Models." IEEE Transactions on Software Engineering, Vol. 14, No. 10, October 1988, pp. 1453-1461.

4. Eisner, H., J. Marciniak, and R. McMillan. "Computer-Aided System of Systems (S2) Engineering."Proc. of the 1991 IEEE/SMC International Conference on Systems, Man, and Cybernetics, Charlottesville, VA, IEEE Computer Society Press, October 1991, pp. 531-537.

5. Gomaa, H. "The Impact of Prototyping on Software System Engineering." in R.H. Thayer, M. Dorfman (eds.), Systems and Software Requirements Engineering, IEEE Computer Society Press Tutorial, 1990.

6. Hsu, H., M. Bouziane, W. Cheung, J. Nogues, L. Rattner, and L. Yee. "A Metadata System for Information Modeling and Integration." Proceedings of the First International Conference on Systems Integration, Morristown, NJ, IEEE Computer Society Press, April 1990, pp. 616-624.

7. Lawson, H.W. "Philosophies for Engineering Computer-Based Systems." IEEE Computer, Vol. 23, No. 12, December 1990, pp. 1859-1874.

8. Lawson, H.W. "Application Machines - An Approach to Complexity Reduction." presented at Computer Based Systems Engineering Workshop 1992, University of Maryland, USA, March 1992.

9. Lawson, H.W. "Application Machines: An Approach to Realizing Understandable Systems." Keynote Address, EUROMICRO '92, Paris, France, September 1992, to appear.

10. Mittermeir, R.M., and M. Oppitz. "Software Bases for the Flexible Composition of Application Systems", IEEE Trans. on Software Engineering, Vol. SE-13, Nos. 4, April 1987, pp. 440-460.

11. Mittermeir, R.M. "POWDER - A Recursive Methodology for Prototyping of Wicked Development Efforts with Reuse." Institut fuer Informatik, Universitaet Klagenfurt, Austria, Report for International Software Systems Inc., Austin TX,

USA, April 1991.

12. NIST, A Framework for a Software Engineering Environment, NIST Special Publication 500-201, Technical Report ECMA TR/55, 2nd Edition, NIST, 1991.

13. OSCA, The Bellcore OSCA Architecture, Bellcore - Bell Communications Research, Technical Advisory, TA-STS-000915, Issue 3, March 1992.

14. OSCA, The Bellcore OSCA Architecture, Bellcore - Bell Communications Research, Technical Reference, TR-STS-000915, Issue 1, October 1992.

15. Power, L. R. "Post-Facto Integration Technology: New Discipline for an Old Practice." Proceedings of the First International Conference on Systems Integration, Morristown, NJ, IEEE Computer Society Press, April 1990, pp. 4-13.

16. Prieto-Diaz, R., and G. Arango. Domain Analysis and Software Systems Modeling, IEEE Computer Society Press, Los Alamitos CA, 1991.

17. Ross, D. T. "Structured Analysis (SA): A Language for Communicationg Ideas." IEEE Transactions on Software Engineering, Vol.3, No.1, January 1977, pp. 16-33.

18. Rossak, W., and S. Prasad. "Integration Architectures - A Framework for System Integration Decisions." Proc. of the IEEE Internat. Conference on Systems, Man, and Cybernetics, Charlottesville VA, USA, October 1991, pp. 545-550.

19. Rossak, W., and P.A. Ng. "Some Thoughts on Systems Integration - A Conceptual Framework." Journal of Systems Integration, Vol.1, No. 1, Kluwer, 1991, pp. 97-114.

20. Rossak, W., and P.A. Ng. "Systems Development with Integration Architectures." Proc. of the IEEE Second International Conference on Systems Integration, Morristown NJ, USA, June 1992, pp. 96-103.

21. Rossak, W., A. Stoyenko, T. Zemel, and P.A. Ng. "On Real-Time Aspects of Systems Integration." Internal Report, CIS-92-11, New Jersey Institute of Technology, 1992.

22. Royce, W. W. "Managing the Development of Large Software Systems: Concepts and Techniques." in Proc. of WESCON, August 1970.

23. Schaefer, W., and H. Weber. "European Software Factory Plan - The ESF Profile." in P.A. Ng and R.T. Yeh (eds.), Modern Software Engineering, Van Nostrand Reinhold, New York, 1989, pp.613-638.

24. Tracz, W. "A Conceptual Model for Megaprogramming." ACM SIGSOFT Software Engineering Notes, July 1991, pp. 36-45.

25. Wimmer, K., and N. Wimmer. "Conceptual Modelling Based on Ontological Principles." Siemens AG, Corporate Research and Development, Munich, Germany, 1992.

26. Zemel, T., and W. Rossak. "Mega-Systems - The Issue of Advanced Systems Development." Proc. of the IEEE Second International Conference on Systems Integration, Morristown NJ, USA, June 1992, pp. 548-555.

27. Zemel, T., W. Rossak, and H. Thimm, "Domain Analysis as a Major Component of Integrated Systems Development." Proc. of SERF '92, 1992 Software Engineering Research Forum, Indialantic FL, USA, November 1992.

28. Zemel, T. "A Mega-System Development Framework." Ph.D. Thesis, Department of Computer and Information Science, NJIT, in work.29.

30. Beck, R. P., S. R. Desai, R. P. Radigan, and D. Q. Vroom, "Software Reuse: A Competitive Advantage." Report, AT&T Bell Laboratories, Columbus Ohio, 1991.

31. Boehm, B. W. "A Spiral Model of Software Development and Enhancement." IEEE Computer, Vol. 21, No. 5, May 1988, pp. 61-72.

32. Biggerstaff, T. J., and A. J. Perlis. Software Reusability, Vol. I and II, ACM Press, Addison Wesley, New York, 1989.

33. Best, L. J. Application Architecture - Modern Large Scale Information Processing, Wiley, New York, 1990.

34. Boehm, B. W. Software Engineering Economics, Prentice Hall, 1981.

35. Boehm, B. W. "A Spiral Model of Software Development and Enhancement." IEEE Computer, Vol. 21, No. 5, May 1988, pp. 61-72.

36. Mills, J. A., and L. Ruston. THE OSCA ARCHITECTURE: Enabling independent product software maintenence; EUROMICRO '90, Workshop on Real Time; June 1990.

37. IBM, Office Information Architectures: Concepts, IBM GC23-0765

38. IBM, Introducing the OfficeVision Family: An SAA Application, IBM GH2 1 -0448.

39. IBM, Open Systems Interconnection within Systems Application Architecture, IBM G511-1137.
40. IBM, Systems Applications Architecture: AD/Cycle Concepts, IBM GC26-4531.

41. IBM, SAA Applications: A Value Guide, IBM G320-9804. IBM, SAA Library.

42. IBM, Systems Applications Architecture: Writing Applications, A Design Guide. IBM SC26-4362.

43. IBM, Systems Application Architecture Common Communications Support: Summary, IBM GC31-6810.

44. IBM, Systems Application Architecture Common Programming Interface Applications Generator, Reference, IBM SC26-4355.

45. IBM, Systems Application Architecture Common User Access Advanced Interface Design Guide, IBM SC26-4582.

46. IBM, Systems Application Architecture Common User Access Basic Interface Design Guide, IBM SC26-4583.

47. IBM, Systems Network Architecture: Concepts and Products, IBM GC30-3072.

48. Grochow, J. M. "SAA: A guide to Implementing IBM's Syatem Application Architecture." Yourdon Press, Englewood Cliffs, NJ 07632.

49. DCM, Bull's Distributed Computing Model: Specifications Overview, 0:00.

50. DCM, Bull's Distributed Computing Model: Interoperability Environment, 0:10.

51. Mullender, Sape. "The ANSA Project and standards." Distributed Systems.

52. An Overview Of ANSAware 4.0, Document RM 099.00, March 1992.

53. Application Programming in ANSAware, release 4.0, Document RM 102.00, March 1992.