2

THE DESIGN OF AN ADVANCED AUTONOMOUS SPACECRAFT COMPUTER

Technical Report No. ESC-82-005

2

THE DESIGN OF AN ADVANCED AUTONOMOUS SPACECRAFT COMPUTER

Technical Report No. ESC-82-005

1

T. Gomi
M. Inwood
I. McMaster

Eidetic Systems Corporation

March 15, 1983

Government of Canada Gouvernement du Canada

Department of Communications

DOC CONTRACTOR REPORT DOC-CR-SP -83-033

DEPARTMENT OF COMMUNICATIONS - OTTAWA - CANADA

SPACE PROGRAM

TITLE: The Design of an Advanced Autonomous Spacecraft Computer [illegible]

AUTHOR(S): T. Gomi (Applied AI Systems Inc., Kanata, Ont.)
M. Inwood
I. McMaster

ISSUED BY CONTRACTOR AS REPORT NO: ESC-82-005

PREPARED BY: Eidetic Systems Corp.
P.O. Box 13340
Kanata, Ontario

DEPARTMENT OF SUPPLY AND SERVICES CONTRACT NO: 15ST.36001-2-0561
SERIAL No. 0ST82-00056

DOC SCIENTIFIC AUTHORITY: R.A. Millar

CLASSIFICATION: Unclassified

DATE: March 1983

DD 4058080
DL 4087618

QL
C655
G64
1983

C O N T E N T S

	Page
Acronyms	i
Acknowledgements	ii
Summary	iii
1. Introduction	1-1
2. Hierarchy of System Autonomy	2-1
2.1 Introduction	
2.2 Definition of Hierarchy	
2.2.1 Layer 1: Physical	
2.2.2 Layer 2: Deterministic	
2.2.3 Layer 3: Module Integrity	
2.2.4 Layer 4: Module Relationship Integrity	
2.2.5 Layer 5: Intelligence_1	
2.3 Onboard Autonomy Management	
2.3.1 Introduction	
2.3.2 Function of the OAM	
2.3.3 Structure of the OAM	
2.4 Fault-Tolerance Management	
2.4.1 Functions	
2.4.2 Examples	
2.4.3 Examples	
3. Structural Hierarchy	3-1
3.1 Introduction	
3.2 Definition of Structural Hierarchy	
3.3 Structural Hierarchy	
3.4 External Linkage	
3.5 Subsystem Interconnection	
3.5.1 Introduction	
3.5.2 Mode of Communication	
3.5.3 Traffic Volume	
3.5.4 Homogeneity and Dynamism	
3.5.5 Application	
3.5.6 Type of Network	

Note: The underscore (_) is commonly used in Computer Science to form an identifier from a collection of names.

3. Structural Hierarchy (continued) 3-1

3.6 Module Interconnection
3.7 Element Interconnection
3.7.1 Introduction
3.7.2 Serial Backplane
3.7.3 Flexibility

4. System Test 4-1

4.1 Test Concepts
4.2 Methodology
4.3 Test Examples
4.4 Resources
4.4.1 Personnel
4.4.2 Computer Equipment
4.4.3 Simulators
4.4.4 Debugging Aids
4.4.5 Test Data Generators

5. Conclusion 5-1

6. References 6-1

Appendix A-1

ACRONYMS

AASC	Advanced Autonomous Spacecraft Computer (AASC)
Ada	DoD defined Ada programming language
AI	Artificial Intelligence
AOCS	Attitude and Orbit Control System
APSE	Ada Programming Support Environment
ASM	Autonomous Spacecraft Maintenance
CCN	Cross Country Network
CKB	Compound Knowledge Base (AASC/OAM/CKB)
CSMA/CD	Carrier-sense Multiple-Access with Collision-Detection
EGN	Extra Global Network
FA	Factory Automation
FTM	Fault-Tolerance Manager
FTC	Fault-Tolerant Computing
GN	Global Network
IEEE	Institute of Electrical and Electronic Engineers
IIU	Subsystem I/O Interface Unit (AASC/IIU)
ISO	International Standardization Organization
KB	Knowledge Base
KE	Knowledge Engineering
LA	Laboratory Automation
LAN	Local Area Network
LHN	Long Haul Network
MBPS	Mega Bits Per Second
MIPS	Mega Instructions Per Second
MN	Metropolitan Network
MTFF	Mean Time to First Failure
NASA	National Aeronautics and Space Administration
OA	Office Automation
OAM	On-Board Autonomy Management (AASC/OAM)
OCS	Onboard Consultation System (AASC/OAM/OCS)
OSI	Open Systems Interconnection
PCU	Subsystem Processor Complex Unit (AASC/PCU)
SSN	Solar System Network
ULAN	Ultra Local Network
VLAN	Very Local Network
VLSI	Very Large Scale Integration

ACKNOWLEDGEMENTS

This report is one of a series resulting from studies performed for the Federal Department of Communications, Communications Research Centre, Shirley Bay, Ottawa, Ontario, Canada under DSS Contract OST82-00056.

The authors would like to acknowledge the assistance of Ian Cunningham for a contribution on "TRANSPORT Services", contained in an Appendix to this report, and for consultation on network issues; Dr. John Tsotsos, Department of Computer Science, University of Toronto; Dr. Nancy Leveson, University of California, Irvine; Kurt Joerger, Director of Knowledge Systems, Tecknowledge Inc., California; Michael Deering, AI Research Laboratory, Fairchild Camera and Instrument Corporation, California; Robert Hindon, Bolt, Beranek and Newman, Cambridge, Massachussetts; and Lee Holcomb, of NASA. The authors also acknowledge the support of R.A. Millar, of the Communications Research Centre.

SUMMARY

The design of the Advanced Autonomous Spacecraft Computer (AASC) is divided into Autonomy Design and Structural Design. The system test to be performed on the implemented AASC is also described. Both Autonomy and Structure are decomposed using top-down layered models (for which meta-rules are given), consistent with the Open System Interconnection (OSI) reference model that serves as a basis for the communications protocol used throughout the AASC. System test methodologies are chosen to be consistent with the top-down design and proposed implementation.

The layers of autonomy in the AASC range from the Physical (lowest) level, Deterministic Control, Module Integrity, Module Relationship Integrity, to Intelligence_1 and beyond. Established state-of-the-art techniques are specified for implementing the first four layers, termed Fault Tolerance Management (FTM) containing methodologies developed mostly in the field of software fault-tolerance, and advanced knowledge engineering techniques are specified at the highest level, called On-board Autonomy Management (OAM).

The OAM is decomposed into functional units which allow frame-based and other representations of knowledge, generalized inference over a compound knowledge base, the generation of explanations for ground control, and limited knowledge acquisition. The FTM is decomposed into more traditional software and hardware units that perform detection of, isolation of, analysis of, and recovery from failures at layers from the physical components up to groupings of software modules.

The structure of the AASC and its logical environment is broken down into the external networks with which it must interact, internal subsystems, module networks, and element networks. These structural layers are each analyzed and specified as networks under the OSI model.

System testing will continue through development, consistent with top-down design, implementation, and testing. The criteria for testing are presented in terms of the requirements for system reliability, flexibility, fault-tolerance, performance, autonomy, and conformity to FTC design principles. Provision of personnel, computer equipment, simulators, debugging aids, and test data generators are specified to meet the resource requirements of system testing.

1. INTRODUCTION

The purpose of this report is to present a detailed analysis of the functions and logical structure of the Advanced Autonomous Spacecraft Computer (AASC), the Functional Specification of which was given in a previous report [GOMI 83]. The functional analysis uses a layered model and decomposes functions to a level of detail at which known, clear, well-specified algorithms can be used to implement the functions. The rationale and meta-rules for the layered model are presented, including comparisons with other layered models.

The layers of AASC functions are grouped into two major capabilities: On-board Autonomy Management (OAM), and Fault-Tolerance Management (FTM), because of the natural boundary between the existing bodies of techniques used to implement functions in the two domains. In order to characterize the two domains, examples drawn from the AASC and other systems are presented.

The implementation of a set of functions requires the creation of a logical and physical structure which may or may not be isomorphic to other functional hierarchies. The logical structure of the AASC and the reasons for choosing it are presented. The structural description includes a hierarchy of the interconnections of system modules. Each level of decomposition of the AASC is described, from the external network environment down to the elementary level within modules.

The logical structural decomposition of the AASC stops at the point where decisions must be made about hardware/software tradeoffs. Hence, the output of this report becomes the input to the next phase of AASC development: hardware choice and software module design.

2. HIERARCHICAL STRUCTURE OF SYSTEM AUTONOMY

2.1 Introduction

With few exceptions the move towards a unified description of system autonomy is not yet evident. The benefit of establishing a form of common language among system developers is obvious. The availability of such a language will permit freer exchange of the results of research and development which are constantly producing output of wider interest that can be incorporated into the design of new autonomous systems. The probability of newly discovered methodologies in system building may only be realized if the knowhow is properly categorized and documented in a widely visible fashion. Accordingly, the AASC project attempts to organize system autonomy in a hierarchical framework, and defines its own autonomy using that framework.

It is obvious that there is a very close relationship between the autonomy of a system and its intelligence. A study by NASA confirms a linear relationship between the two, as shown in Figure 2.1. Intelligence is a necessary ingredient in the achievement of system autonomy while the reverse is not necessarily true. Input signals which affect the output of a servo system, blood cells attacking foreign organisms, automatic transmission adjusting the output speed and torque based on several inputs, reflex movements made by a simple chain of neurons in avoiding a physical threat, or a governing council voting on a specific subject, are some examples in nature and among man-made systems of intelligent bodies effecting the act of autonomy. The objective here is to establish a scale of system autonomy using intelligence imbedded in the system as a measure, and to classify autonomous systems according to their demonstrated intelligence.

The Hierarchy of Autonomy is modelled after the Open Systems Interconnection (OSI) reference model established in the mid 1970s for computer communication by the International Standardization Organization (ISO). Also affecting the AASC design in this regard is the view expressed by E. Dijkstra in 1968 [DIJK 68] in his definition of "The Operating System", which subsequently led to the universal adoption of layered structuring techniques in various subfields of computer and system sciences. Virtually all modern operating systems have been designed hierarchically ever since. Networking, including the OSI mentioned above, became viable only after a reasonable layering scheme was devised. Computer Graphics is presently being re-defined at an international level in terms of various levels of abstraction in representation. There is already a move to-

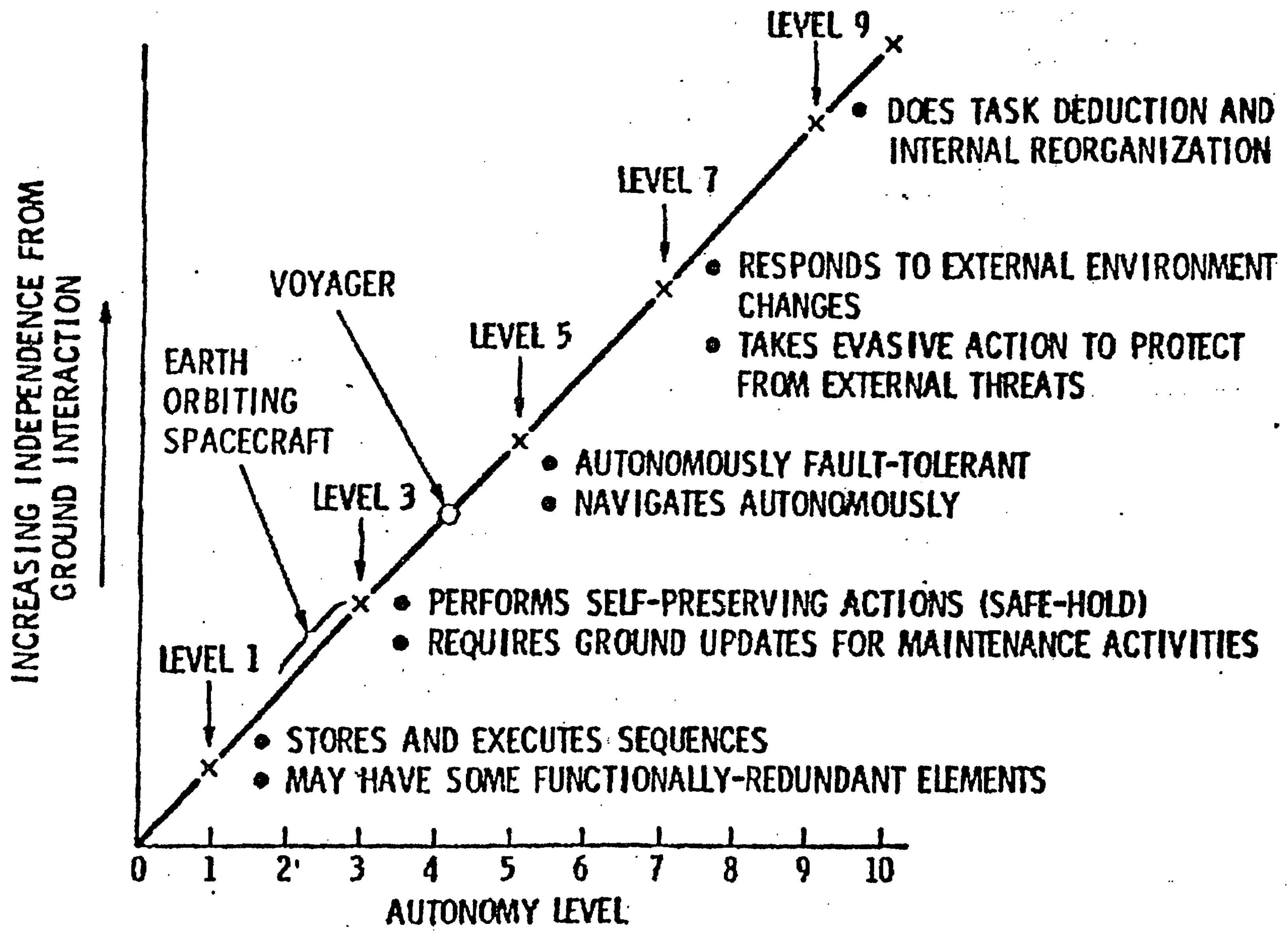


Figure 2.1 Relationship between autonomy level and on-board intelligence, presented by L. Holcomb of NASA at the Fault-Tolerant Computing Workshop, November 1982, Forth Worth, Texas.

wards applying the theory of layering to Artificial Intelligence [ALBU 81].

Compared to relatively well-established domains such as those discussed above, with a longer history of structuring system functions into a hierarchy, the hierarchy of system autonomy originated here is more likely to face revision in the foreseeable future. Such was the case with others when they were still young. It will be safe to state that the present definition described in Section 2.2 below only reflects the best-effort conceptualization of the hierarchy by the designers of the AASC. The Hierarchy of System Autonomy is intended to be independent from the AASC and expected to survive in various forms. It is obvious, however, that a considerable number of revisions will be necessary until the concept gains wider understanding. The structure shown at this time is the result of an extensive study of existing fault-tolerant techniques carried out by the authors [GOMI 82a,b].

Table 2.1 The Hierarchy of System Autonomy

```
                   --------------------------------
            6      | This layer performs the      |
                  _| system's required functions. |<-
Each layer       |‾|------------------------------| _|   Each layer
has some    5    ->|                              |  ‾    provides
form of           _|                              |<-     service to
control or       |‾|------------------------------| _|   the next
effect on   4    ->|                              |  ‾    upper
the next          _|                              |<-     layer.
lower            |‾|------------------------------| _|
layer.      3    ->|                              |  ‾
                  _|                              |<-
                 |‾|------------------------------| _|
            2    ->|                              |  ‾
                  _|                              |<-
                 |‾|------------------------------| _|
            1    ->|                              |  ‾
                   |                              |
                   --------------------------------
```

The AASC adopts the view that faults exist at every level of the functional hierarchy - a fault-tree concept depicted, for example, in [VESE 81]. It also takes the position that fault-tolerance must also exist in a hierarchical fashion - i.e., recognition of a fault-handling tree. This latter view is often incompatible with the view of some researchers who believe that, upon defining a fault-tree one can always design-in fault-tolerance during system development. They seem to conclude that a successful and comprehensive description of possible faults in the system necessarily leads to the definition of methods which would allow them to avoid faults beforehand. Such an approach is dominant, for example, in the design of on-board systems for commercial aircraft. The rejection of this approach is based on the heuristic stance the project has adopted through its study phases. It believes that the deterministic approach taken by those designers is insufficient in dealing with many unknown errors that a system will likely encounter in a hostile environment of space where, in most cases, on-board repairs will be extremely difficult and help will never arrive in time.

In the Hierarchy of System Autonomy, fault-tolerance is not localized in certain layers but implied in every layer. Occasionally, expressions such as Fault Tolerance Management (FTM), are used to describe an instantiation of certain layers of the hierarchy. The reason for the localized identification of fault-tolerance is, however, for convenience so that such a commonly understood concept as "Software Fault-Tolerance" is given an appropriate visibility in the design. The vertical distribution of fault-tolerance throughout the design of the AASC is never abandoned.

The following meta-rules are considered valid in regard to the Hierarchy of System Autonomy:

1. An ascending order of intelligence is observed when the hierarchy is followed upward.

2. An autonomy function defined at a given level must be capable of correcting a faulty condition detected by the next lower level in such a way that the result of the fault is transparent to the execution of functions at the next higher level. Note, however, that failure to accomplish this correction is a failure at this given level, and comes under Rule 3.

3. A function invoked at a given level is responsible for reporting failures it cannot recover from by itself, to the next higher level.

4. At a given level in the hierarchy, a protocol can be defined to achieve inter-level communications and control described in 2. and 3. above.

5. Similarly, a protocol can be defined for communication and control among functions (intra-level) operating within a given level.

2.2 Definition of the Hierarchy of System Autonomy

2.2.0 The Hierarchy

Table 2.2 shows the Hierarchy as defined. The layers 1 through 5, which are described in Sections 2.2.1 through 2.2.5 below, will be defined and implemented in detail in further stages of the project. Figure 2.2 depicts the General Hierarchy of Fault-Tolerant Functions.

Table 2.2 The Functional Hierarchy of System Autonomy

Layer	Name	Description
6	Intelligence 2	Sophisticated KE technology is used to support the intelligence of a system.
5	Intelligence 1	A system has a primitive intelligence based on Knowledge Engineering (KE).
4	Module relationship integrity	Maintain ability of module groupings to carry out major system functions.
3	Module integrity	Maintain ability of modules to execute without interference from other modules.
2	Deterministic control	Ensure only healthy processors take part in system.
1	Physical	Ensure only healthy physical components continue to contribute to processor operation
0	non autonomous	

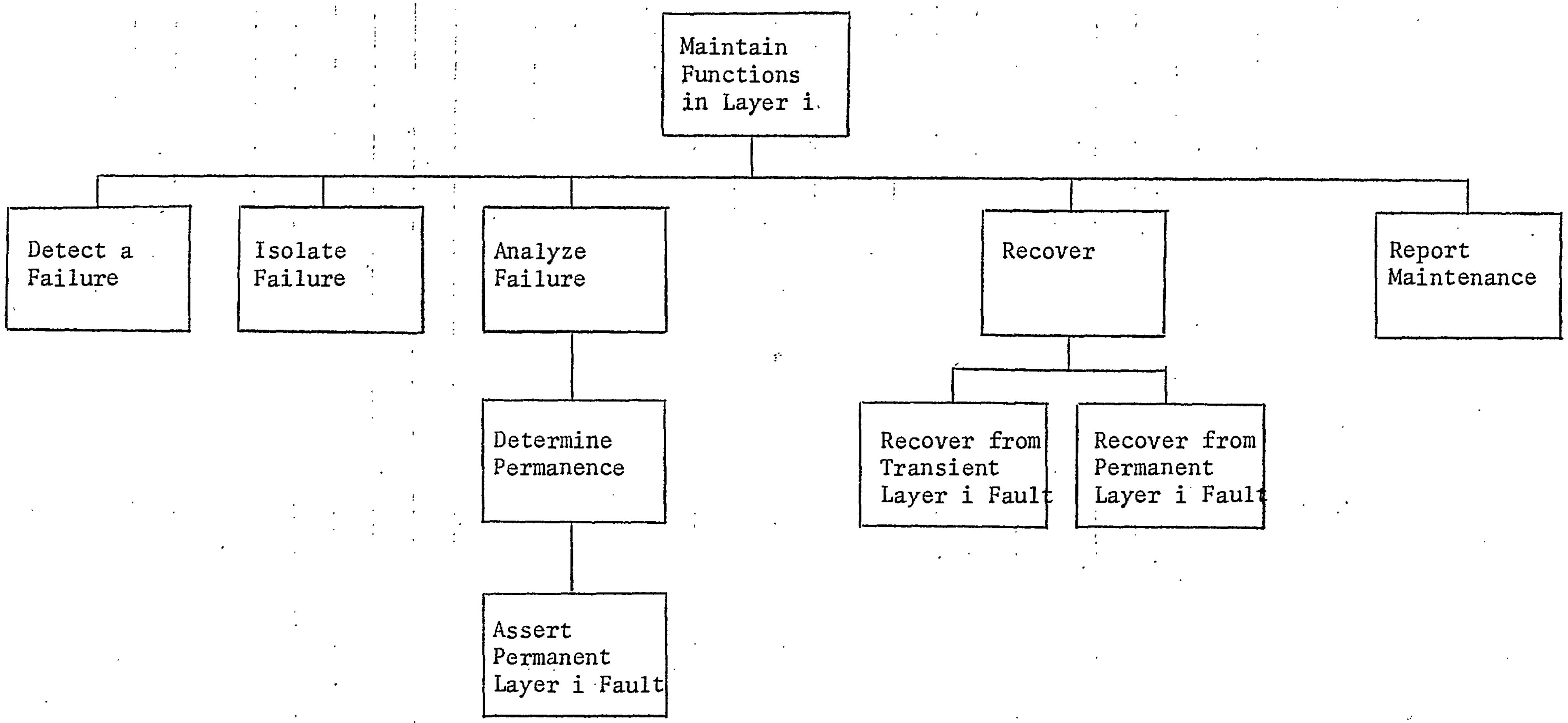

Maintain Functions in Layer i
Detect a Failure
Isolate Failure
Analyze Failure
Recover
Report Maintenance
Determine Permanence
Assert Permanent Layer i Fault
Recover from Transient Layer i Fault
Recover from Permanent Layer i Fault

Figure 2.2 General Hierarchy of Fault-Tolerance Functions at Layer i.

Of all the modelling methodologies used to describe computer systems, there is one that has emerged as the most generally applicable, lucid, understandable, and implementable: layering. The principle is that a function may be decomposed into an ordered set of N classes of subfunctions called layers. The relationships between two layers Li and Li+1 are:

1. Li provides information and/or services to Li+1

2. Li+1 does not have access to any information or service from layers L1 to Li-1, except by using the services of Li.

The objectives of layering, and the reasons for its success in system design are:

1. It divides the system design task into manageable units

2. It allows the compartmentalization of data structures and processes so that validation and fault-tolerance are easier to achieve.

3. It enables the progressive implementation of the system from the most primitive to the most abstract function.

In the AASC, the relationship between layers, with the emphasis on fault-tolerance, is defined as follows:

1. Li+1 may perform recovery from faults in Li.

2. Li reports permanent failures to Li+1.

3. Functions within Li are responsible for detecting the failure of other functions in Li with which they interact.

The main incentive for these meta-rules is fault-tolerance. Each layer may be viewed as a manager of the lower layer. Within each layer, cooperating functions observe a protocol in their relationship that ensures that:

1. They do not corrupt each other.

2. They know whether cooperating functions are behaving correctly.

When things go wrong within a layer, that layer is able to call on the next layer above for help in recovering. The following subsections describe the Hierarchy of System Autonomy in further detail.

2.2.1 Layer 1: Physical

Physical failures can occur when bits are incorrectly stored, fetched, or compared, or otherwise manipulated: signals are corrupted; or physical state changes are mistimed. An error in this layer can be transient or permanent. If it is transient, it can be recovered from, usually by retrying the operation. If it is permanent, it is reported in some way to Layer 2, so that recovery can be attempted. The functional hierarchy of Layer 1 is shown in Figure 2.3.

2.2.2 Layer 2: Deterministic Control

The label "determinism" is used here to provide a more general classification than the term "processor". While it does include processing units, it also includes bus controllers, memory controllers, and i/o device controllers, or a functional equivalent of these in software, which are all characterized by being aggregations of components which execute predictable complex sequences of primitive operations with well-specified timing characteristics. The job of Layer 2 is to ensure that the failure of a deterministic function within the AASC, such as a processor, is detected and recovered from without adversely affecting other elements of the system. Figure 2.4 shows the functional elements of this layer.

Layer 2 interacts with Layers 1 and 3 as follows. A permanent physical failure (Layer 1) must be recovered from by this layer. If recovery is not possible, then the physical failure causes a failure in a deterministic function. A transient determinism failure (such as a transient bit error in an internal processor data transfer) can be recovered from, and the deterministic function is restarted. However, a permanent failure must be reported to Layer 3 for recovery or to ultimately cause a task failure.

2.2.3 Layer 3: Module Integrity

The job of Layer 3 is to make sure that each module, herein also called "task", is allowed to perform its function correctly, without being adversely affected by any other task. Its functional hierarchy is shown in Figure 2.5. This means that

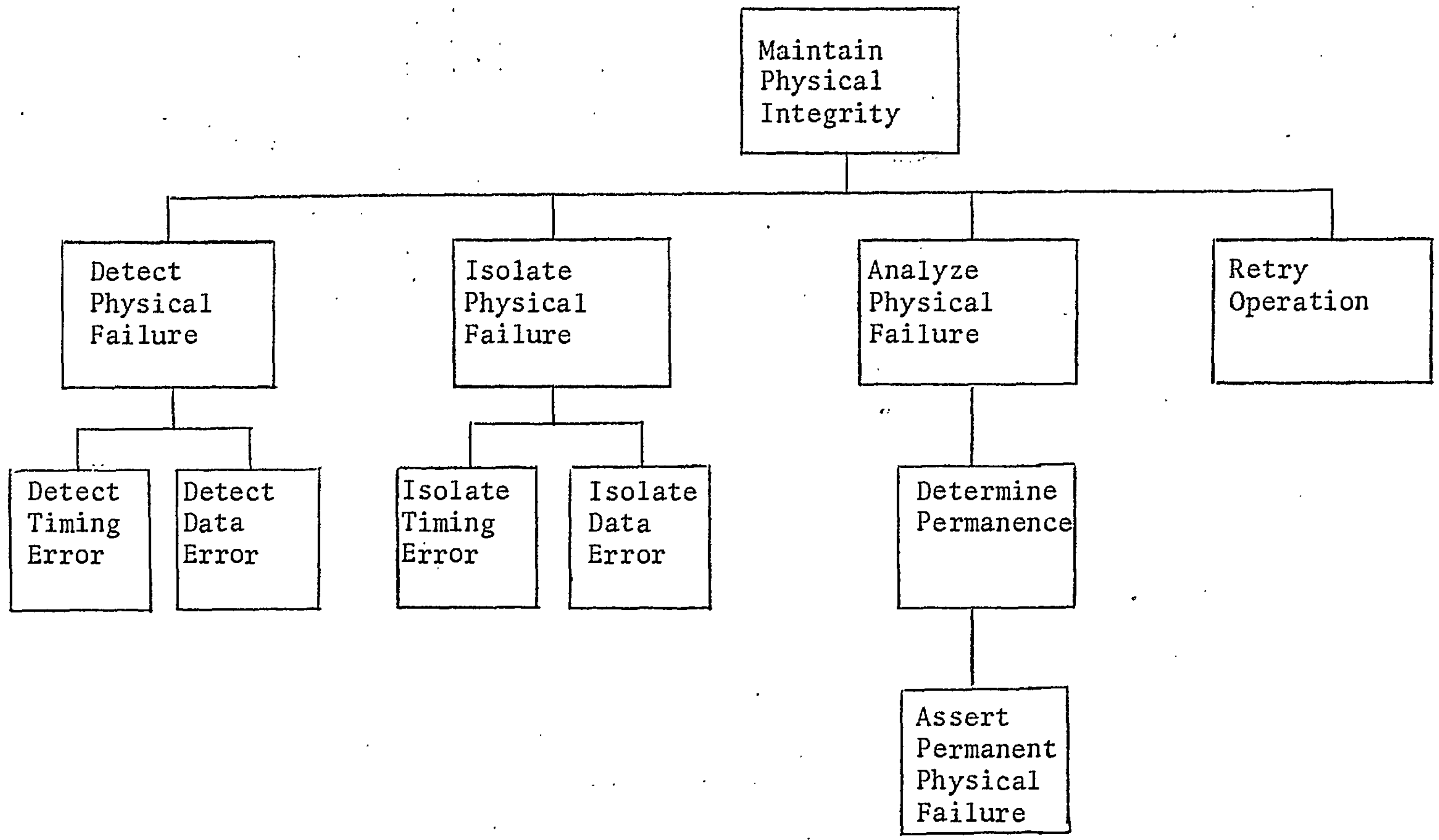


Figure 2.3 Layer 1 Functional Hierarchy

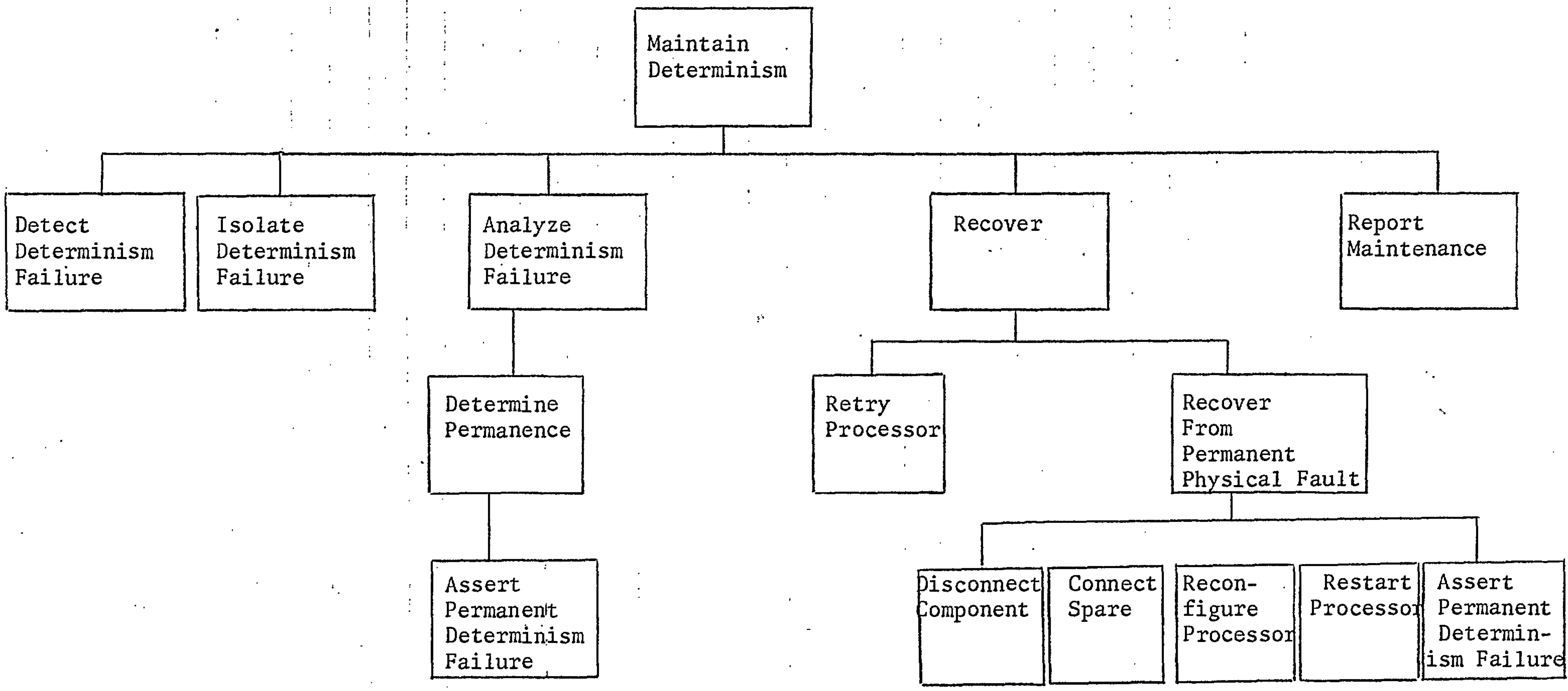

Maintain Determinism
Detect Determinism Failure
Isolate Determinism Failure
Analyze Determinism Failure
Recover
Report Maintenance
Determine Permanence
Assert Permanent Determinism Failure
Retry Processor
Recover From Permanent Physical Fault
Disconnect Component
Connect Spare
Recon-figure Processor
Restart Processor
Assert Permanent Determin-ism Failure

Figure 2.4 Layer 2 Function Hierarchy

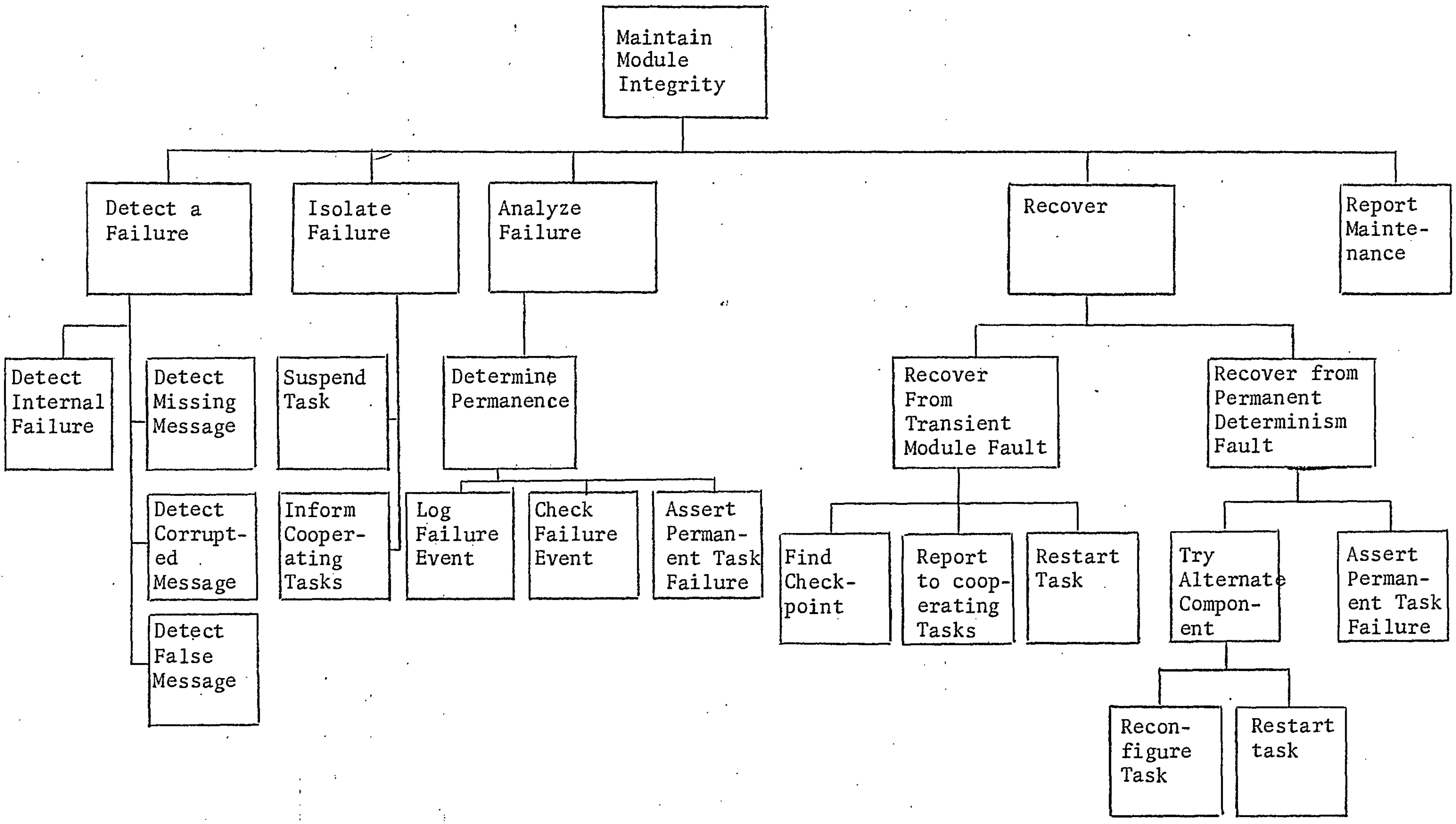


Figure 2.5 Layer 3 Functional Hierarchy

Task A must know when information originating in Task B is correct. Traditionally, this would imply that Layer 3 must handle both failure in B and a failure in the channel (A,B). However, Layer 3 need only consider the logical channel (A,B), and a logical channel behaves exactly like a task whose function it is to accept and deliver messages. Thus we may view all failures in this layer as task failures. Failures in actual physical links will be the responsibility of Layer 1.

A permanent fault in Layer 2, that is, in a deterministic function of the AASC (generally a processor) causes Layer 3 to try to recover. If this is not possible, a task failure is asserted. A task failure can also occur due to design or implementation faults. In any case, when a task fails, it may be permanent or transient. If it is transient, recovery is carried out. If it is permanent, this fact is reported to Layer 3, where it causes a failure in module relationship integrity.

2.2.4 Layer 4: Module relationship integrity

Figure 2.6 shows the hierarchy of functions within Layer 4. By module relationships we mean the ability of functional modules, such as tasks, processes, and logical communication channels, to cooperate to provide the functions required of the system. As long as this layer is able to provide recovery from permanent failure of tasks and channels in Layer 3, the system can maintain all functions. If, however, it is impossible to recover from such a failure, then degradation of the functional capabilities of the system occurs. Degradation may also occur without an explicit failure in Layer 3. The layer will check a series of assertions about the state of the various AASC functions. For instance, it may check queues of outstanding requests for resources from application programs and find either queue size or waiting time above the acceptable thresholds.

In either case, degradation is checked as to its permanence, and if it is found to be permanent, this fact is reported to Layer 5. A permanently degraded state requires goal-directed decisions based on knowledge, which is the domain of Layer 5, the On-board Autonomy Manager.

2.2.5 Layer 5: Intelligence_1.

The use of domain specific knowledge in achieving system autonomy characterizes the fifth layer of the hierarchy. It may be called the KE-layer, since the technology used to apply do-

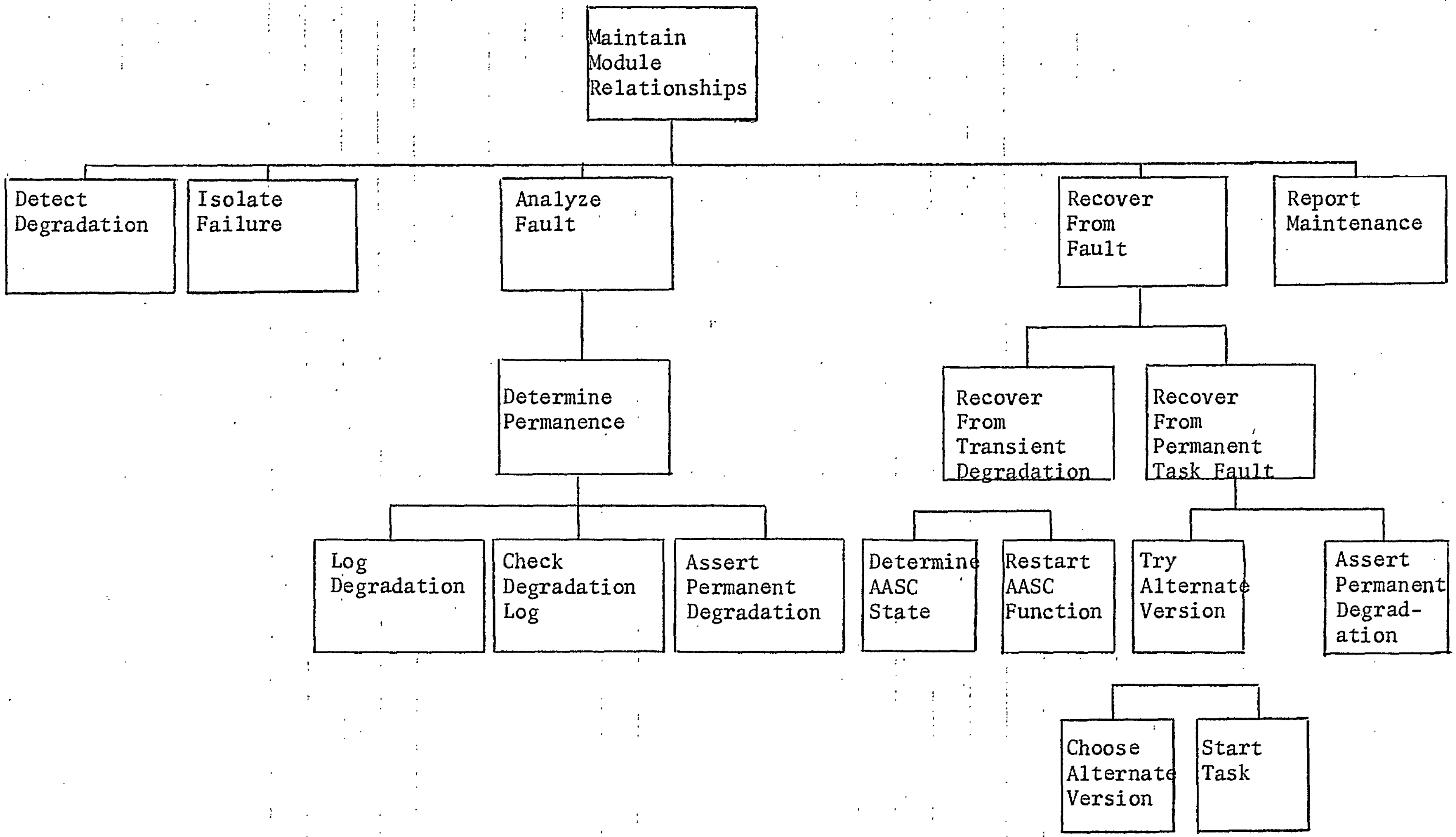


Figure 2.6 Layer 4 Function Hierarchy

main specific knowledge is termed Knowledge Engineering (KE) as it is conceived at the present time. It is anticipated that this will be the first of many intelligent layers to be built within the hierarchy. The implication here is that the capabilities of the first intelligent layer are limited to those supportable by using primitive Artificial Intelligence technology. However, the layer distinguishes itself from those conventional technologies often used in highly sophisticated computer equipment such as intelligent peripherals and intelligent memory subsystems, because these systems do not characterize themselves as knowledge-based systems.

To qualify as an Intelligence_1 system, a system must possess the following capabilities and/or characteristics:

1. limited natural language processing sufficient to interface a trained human operator;

2. inference mechanism that is limited in scope but reliable;

3. uniform application of an inference algorithm at all levels of the knowledge stored within a single knowledge representation scheme (generalized inference engine);

4. ability to explain, in limited form, the process of inference used on the specific problem for which the consultation was made;

5. support structure to operate and maintain a knowledge base;

6. ability to select suitable knowledge representation scheme for the given problem domain;

7. ability to select the appropriate control mechanism to access the represented knowledge;

8. ability to make limited use of heuristics in the inference process when necessary; and

9. possession of primitive knowledge acquisition mechanism which operates under strict human control.

The AASC supports Intelligence_1 in the form of On-board Autonomy Management (OAM). The OAM is described in section 2.3 below in further detail. Precise definition and implementation of an example of an autonomy management subsystem for on-board functions in the form of a consultation system, will be the first major milestone in the development of the OAM. The packaging of the subsystem in a form suitable for on-board use will be the next. The successful testing of the prototype will complete the development cycle.

2.3 The Onboard Autonomy Management (OAM)

2.3.1 Introduction

The fifth layer of the Hierarchy of System Autonomy will be implemented within the AASC as the Onboard Autonomy Management (OAM). It will form the highest intelligence on board the AASC and will be responsible for the satisfactory operation of the entire on-board system. As such, it represents the spacecraft in its relationship to the entire operating environment including its dealing with ground control.

A foundation for accommodating Layer 6 of the autonomy hierarchy (Intelligence_2 layer) in the future is already built into the present design of the AASC. It exists in the form of hooks of various formats distributed throughout the structure of the OAM.

2.3.2 Function of the OAM

The OAM performs, on board, the following management and house-keeping functions:

1. Reporting to Ground Control.

   The reporting may take place at pre-arranged windows in time; upon receiving command from the ground; or when the OAM recognizes one of a number of pre-defined conditions that needs to be reported.

2. Subsystem Monitoring

   The OAM receives distress reports from the Fault Tolerance Management (FTM in Section 2.4 below) in accordance with the inter-layer reporting defined in Section 2.2.0 above, and attempts on-board correction. If it fails, it will raise a condition that would require the OAM to report to the ground. If it succeeds, the event would be transparent to the ground except for the need to make an on-board log entry.

The OAM also sends out enquiries to subsystems. If the reply is not satisfactory, a similar declaration, as on the receipt of distress signals, is made and processed accordingly. A corollary to this is the voluntary, passive reception and analysis of outputs of subsystems collected through software and hardware sensors. Such sensors will be distributed throughout the spacecraft. A similar abnormality condition will be declared if the OAM deduces from other inputs (such as its own operation log described in 2.3.3.3 below), that an abnormal condition has occurred.

3. Enquiry Processing

   Enquiries will be received by the OAM from various external sources, such as ground control, orbital relay station, and other ground and in-orbit stations. It will disperse information concerning its on-board operation including its history of operation. Enquiries are accepted according to a prearranged authorization scheme.

4. Audit

   At any time in its operation, the on-board management is subject to audit by authorized stations. Upon acceptance of authorization, the OAM will establish and maintain a direct communication channel between on-board facilities (including those implemented in software and data files) and an auditor. The audit facility will be implemented as a natural extension of the enquiry facility described above.

5. Reconfiguration Control

   Restructuring of on-board functional modules (software or hardware) will be needed for various reasons such as:

   - detection of a permanent fault in system
   - suspected malfunction of a module
   - mission profile change
   - changing threat to the wellbeing of the spacecraft

- change in load distribution

A reconfiguration may be initiated by a command from ground control, a request from an on-board subsystem, or as the result of subsystem monitoring by the on-board management described above. In the case of a request from a subsystem, a qualification process must precede the reconfiguration. A special case of ground-initiated reconfiguration is the updating or reloading of on-board software. To have a basic system loader available on board even in an emergency, portions of the OAM system software will be implemented in secure memory modules with ample designed-in redundancy.

6. External Communication

As the OAM represents the entire spacecraft, it manages general communication between the spacecraft and external world. This includes authorization, establishment, and maintenance of external communications requested by on-board subsystems. In cases where other on-board communication arrangements are made, for example, the existance of a dominant on-board communications facility such as the payload of a communications satellite, the involvement by the OAM in external communication will be limited.

2.3.3 The Structure of the OAM

2.3.3.0 Introduction

Figure 2.3 shows the structure of the OAM It is made up of two sections: the Control Subsystem (OAM/CS) and the Onboard Consultation System (OAM/OCS). The Control Subsystem authenticates the exchange of information between on-board subsystems and the external world. It also decides if the on-board expert system (OCS) should be consulted on specific issues related to the autonomous control of the spacecraft. It is the nerve centre of the entire on-board control system, although it will maintain redundant spares in the actual implementation.

The OCS performs deductions based on Knowledge Engineering (KE) techniques and returns an answer to the given problem on which the consultation was made. The OCS is subordinate to the CS, and hence to ground control. Ground control can also issue an enquiry to the on-board expert system.

CS, and hence to ground control. Ground control can also issue an enquiry to the on-board expert system.

2.3.3.1 The Control Subsystem

As shown in Figure 2.7, the Control Subsystem consists of an Autonomy Control module, interface functions between on-board subsystems and external (ground) stations, and the Access Control that links them with the OCS. The Access Control shares a local data base called CTLDB with the Autonomy Control to maintain essential system data necessary to run the OAM.

The Control Subsystem performs two roles: interface for external (ground), on board, and consultation subsystems; and decision making for on-board autonomy issues. The Access Control regulates the flow of information among modules using protocols. For example, the ground cannot access on-board subsystems or the OCS without proper authorization imbedded in a protocol. An on-board subsystem will not issue an enquiry to the OCS unless it can use a protocol which is acceptable to the Access Control.

Autonomy Control is concerned with making decisions on issues concerning autonomous operation of the spacecraft. Distress messages passed on to it by Fault Tolerance Management will be assessed and measured using known control algorithms stored in the CTLDB. If it does not yield helpful results, a consultation request will be issued to the OCS. If that proves unsuccessful (no helpful information obtained), then it will report the OAM failure to ground control.

Two interface modules, the Subsystem Interface and the Ground Gateway, act as gateways in the sense in which they are used in networking; that is, gateways as defined in the OSI reference model. A gateway allows interchange between different communication modes at the NETWORK protocol layer. These interfaces may internally support higher layers of the reference model, i.e. TRANSPORT, SESSION, PRESENTATION, and/or APPLICATION layers.

The CTLDB carries several types of data essential to the operation of the OAM and some of the other on-board subsystems. There will be a mechanism in the OAM protocol to allow re-loading of the data base from an external source through the Ground Gateway and the Access Control so that in-flight recovery of the lost data base can be made. The information contained in the data base is as follows:

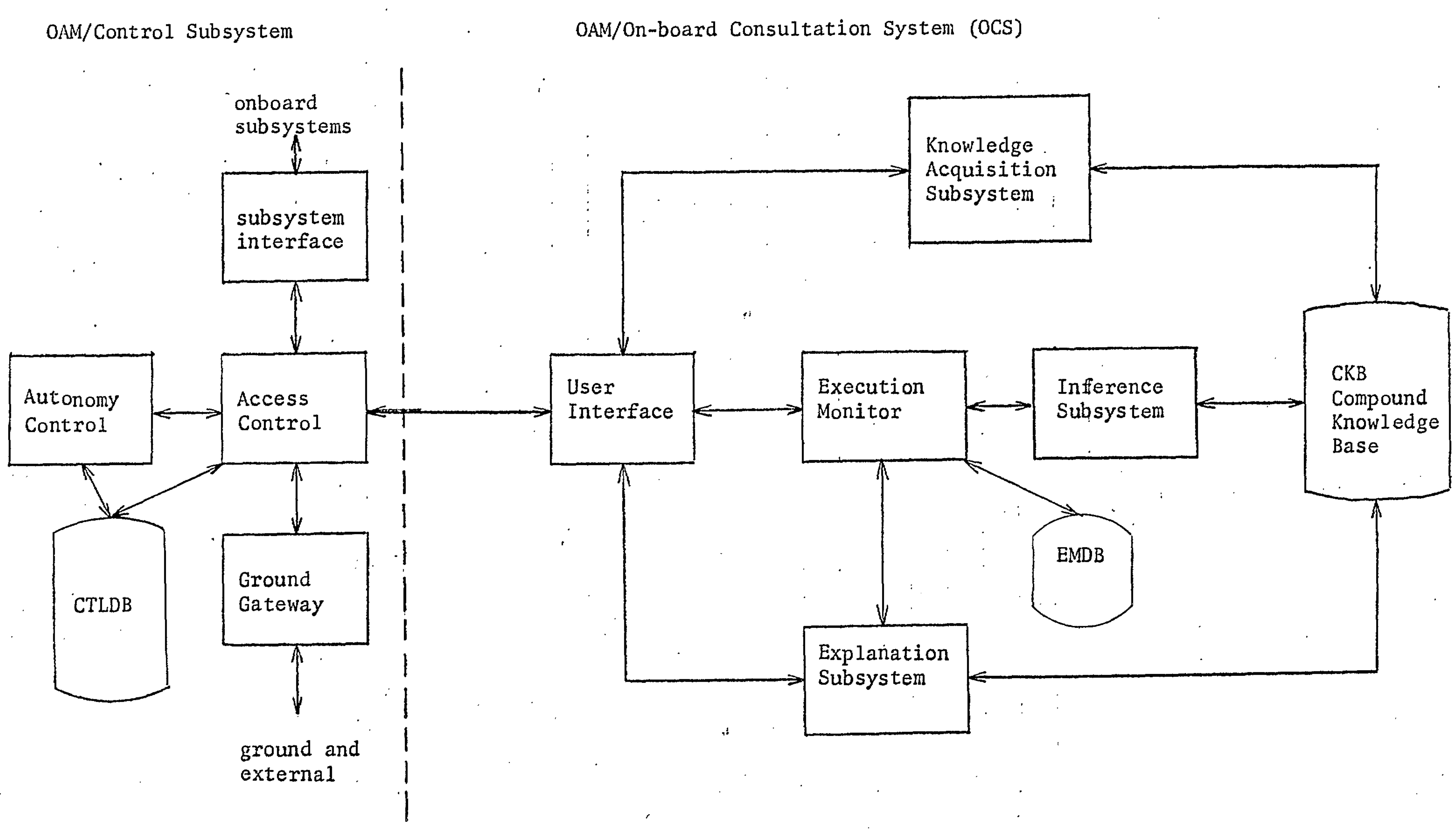


Figure 2.7 On-board Autonomy Manager (OAM) Block Diagram

- software that describes the OAM protocol

- authorization code files

- the entire OAM software except for the contents of the CKB (the CKB re-loading may be performed only when the other parts of the OAM are functioning properly)

- all reloadable software for other on-board subsystems.

- operation log of the OAM, stored in a hierarchical fashion, so that the most recent events are recorded in the greatest detail, while older records are increasingly abstracted.

2.3.3.2 The Compound Knowledge Base (CKB)

The CKB is a collection of knowledge bases (KBs) interconnected to form a distributed knowledge pool on-board the AASC. Each KB that constitutes CKB represents a domain in on-board health monitoring, control/monitor signal processing, mission profile control, emergency procedures, or the structure of the AASC. The method by which knowledge is represented for each of these constituent KBs will be chosen to best suit the domain. However, the frame-based representation will be mainly used because of its sophistication as a knowledge representation tool, similarity to known neurophysiological representations [KENT 81], and the availability of high quality development tools for research and development [TSOT 80, UOFT 82, MYLO 83]. The semantic networks and rule-based representation (production system) will be also used, as well as other forms of associative memory and retrieval mechanisms when appropriate.

The CKB may be expanded to eventually contain KBs in the following problem domains:

- diagnosis of module failure

- prescription of remedies for malfunctions

- failure forecasting

- dynamic reconfiguration management

- mission profile monitoring/change coordination

- effector monitoring

- prediction of interaction of spacecraft with external objects

- action planning

- situation assessment using inference on observed data.

Links will be established among frames within a KB and those in different KBs to form semantic relationships such as the causality of a specific event. Such links are used, for example, to establish association between an abnormality observed in a temporal domain and possible causes of the abnormality in an underlying event domain [SHIB 83]. A practical example would be linkages between fluctuation observed in the output signal level of a Spectroscopic Imaging System and known causes in the underlying event domain that might explain the abnormality.

2.3.3.3 The Execution Monitor, the EMDB, and the Inference Subsystem

Inputs from users are handed in to the Execution Monitor via the OCS User Interface. The Monitor then organizes them in the agenda subsection of the database (EMDB) that records the session specific execution data concerning the inference process. The Execution Monitor will also construct a plan for the execution and record intermediate results from the process. Both the plan and intermediate results will be recorded on the EMDB. The Execution Monitor is also responsible for initiating the inference process at the outset and guides it or otherwise controls its execution, as this becomes necessary. The EMDB is accessible during and after the process to the Explanation Subsystem (2.3.3.4 below) through the Execution Monitor.

The Inference subsystem performs inference using knowledge stored in the CKB. Other than when it requires occasional support from the Execution Monitor, the successive inference stages are carried out autonomously by the Inference Subsystems. Since the knowledge is represented in any one of several representation schemes, an appropriate inference mechanism would have to be chosen automatically by the Inference Subsystem. When association between two concepts takes place, the inference process automatically performs the transition and manages necessary context changes. There is provision for achieving a limited parallelism of the inference processes within the inference subsystem and the Execution Monitor.

Output from the Inference Subsystem is fed back to the

User Interface via the Execution Monitor in one of several representation protocols to be defined for the OCS.

2.3.3.4 The Explanation Subsystem

The objective of the Explanation Subsystem is: (1) to explain to the user how the OCS reached its conclusion on a given problem, and (2) to provide to qualified users the insight of the on-board KBs.

Output from the Explanation Subsystem is given to the requestor through the User Interface in one of the OCR interface protocol formats appropriate to the nature of the enquiry and the type of knowledge representation involved. As the intermediate results of a consultation are stored by the Execution Monitor in the EMDB, the Explanation Subsystem requests the Execution Monitor for the retrieval of this information. The Execution Monitor enters the state in which it can accept the request as soon as an inference process on a given problem begins, and stays that way throughout and after the completion of the process for a period of time determined by a system parameter.

On the other hand, if a user demands a "dump" of the current contents of a KB, the Explanation Subsystem will access the KB directly, even when the KB is in use. To be able to do this, however, the Explanation Subsystem must receive from the user information regarding the identification of the KB and its sections to be dumped. Alternatively, the user can ask the Execution Monitor for the identifiers of KBs, if the enquiry is only concerned with the KBs involved in the inference process presently underway. This form of dump request may be made to enhance the understanding of the details of a specific consultation process and also to debug the Inference Subsystem and the CKB.

2.3.3.5 The Knowledge Acquisition Subsystem

This subsystem will not be implemented in the current AASC design except for a simple KB update facility. Thus, the Layer 5 OAM will only have the ability to selectively rewrite portions of KBs, as needed. In special cases, the entire KB may be rewritten under remote control. Information given via the User Interface with appropriate control codes will be passed on to the Knowledge Acquisition Subsystem. Access control by the Access Control module of the OAM Control and verification of the authorization by the subsystem are especially stringent to avoid erroneous rewriting of KBs.

During the KB-updates, the OCS described above locks out normal accesses and no inference process can take place. The Explanation Subsystem, however, may still be invoked to permit the monitoring of the update process.

In the future when the OAM is redefined at Layer 6 (Intelligence_2 Layer), it will, among other things, have the following added capabilities:

- knowledge-base consistency check

- primitive learning based on limited heuristics

Yet later versions (Layers 7 and onwards) will have advanced learning facilities, and eventual evolutionary operations, such as automated updating of KBs based on the history of operation, experienced mission profile changes, and changes in the environment in which the spacecraft functions at a given time.

2.3.3.6 The User Interface

The User Interface is functionally the sole interface between the Control Subsystem of the OAM and the OCS. In implementing the OAM, however, to avoid creating a single point of failure, care must be taken to provide redundancy.

The User Interface regulates information flow in and out of the consultation system. The inputs are in the form of: a request for consultation directed at the Execution Monitor; a demand for explanation or a dump request directed to the Explantion Subsystem; and a control input to update a KB using the Knowledge Acquisition Subsystem. The outputs are: the inference output from the Execution Monitor; and a statement of explanation from the Explanation Subsystem.

A set of standard protocols that is used at this interface will be defined during implementation. The set will include several presentation formats and encompasses a few levels of abstraction in its contents. Proper protocol will be selected automatically according to the circumstances of the i/o operation or by request. The User Interface performs a limited formatting and translation on data being exchanged. It also applies a layer of access control to the information flow in order to enhance the security of the consultation system. For example, command sequences to update the CKB will be double-checked although they would already have been checked by the Access Control module of the OAM control subsystem before entering the OCS. The User Interface also takes added precautionary measures to prevent unauthorized delivery of explanations by the Explanation Subsystem.

2.4 Fault Tolerance Management

Layers 1-4 of the AASC autonomy hierarchy are grouped under the title "Fault Tolerance Management". There is a recognizable qualitative difference between theories, structures, and techniques used in Layers 1-4 and those used in Layer 5 and above. Furthermore, the functions performed by Layers 1-4 have traditionally been labelled "fault tolerance".

Within each layer of the AASC autonomy hierarchy, the general hierarchy shown in Figure 2.2 should hold. In comparing layers, it is clear that there are three classes of functions within the layers:

1. Functions whose characteristics are well-known and standard. These functions will not be described explicitly, since their implementation will be straightforward and probably off-the-shelf.

2. Functions that occur under different labels in several layers but that are virtually isomorphic in characteristics. These functions will be described generically, with annotations for individual differences induced by each layer.

3. Functions whose characteristics are significantly different in each layer. These will be described individually.

Each function description will include the following components:

1. Function name.

2. Brief description of the steps used to accomplish the function, or reference to the algorithm.

3. Data structures required.

4. Output of the function.

5. If this is a generic function, a series of annotations for the various instances of the function in specific layers.

Layer 4: Module Relationship Integrity

Detect Degradation

Techniques

1. Check each resource and service provided by the AASC, and time-stamp it.

2. If a resource is below its threshold, and has been so for an unacceptable time, assert function_failure.

3. If a service has applications waiting on it for unacceptable time, assert function_failure.

4. If a specified service does not exist, assert function_failure.

5. If possible for the service or resource, issue a test request. If the test result is not within its specified threshold, assert function_failure.

Data Structures

1. List of services and resources, with threshold parameters and test request procedures.

Output

1. function_failure
   -- boolean

2. function_failure description
   -- a record describing the function, failure type, and quantitative description of the failure.

Isolate Failure

Techniques

No action need be taken to accomplish this in Layer 4, since the functions are independent. The functions to be maintained are:

Provide resources to applications

-- secondary storage
-- primary storage
-- communications channels
-- computation

Provide services to applications:

-- application process management

Log Degradation

Techniques

1. Create degradation_description.

2. Write degradation_description on event_log.

Data Structures

1. degradation_description

```
-- record containing
--    function name
--    description of degradation
--    time
```

2. event_log

```
-- data base of events
--    keyed on time, function name
```

Output

1. degradation_description

Check Degradation Log

Techniques

1. Search event_log database for degradation_descriptions with same name as currently degraded function.

2. Determine from this search the parameters of the pattern of degradation for this function, such as:

   - frequency, recent and historic
   - rate of change of frequency
   - rate of change of level of degradation.

Data Structures

1. event_log

```
-- data base of events
--    keyed on time, function_name.
```

Outputs

1. degradation_parameters

```
-- record containing
--    function_name, recent_frequency, historic_
--    frequency, frequency_change, level_change.
```

Determine AASC State

Techniques

1. Examine AASC resource lists, Application service queues.

2. Reinitialize the resource lists and service queues for the degraded function.

Data Structures

1. Resource lists

```
-- list of AASC resources
--    primary storage
--     secondary storage
--     processors
--     communications channels
```

2. Service queues

```
-- queues of applications awaiting service
--     or being served.
```

Output

1. resource list and queue entries for degraded AASC function.

Restart AASC Function

Techniques

1. Load tasks required for function.

2. Update function_configuration_table.

3. Start tasks.

Data Structures

1. Task_library

   -- library of all tasks required by AASC.

2. task_directory

   -- directory of tasks required for each
   --   AASC function.

Output

2. function_configuration_table

   -- table of functions currently executing in AASC

Choose Alternate Version

Techniques

1. Check task_directory for alternate version of failed task. The version may be specified and implemented in the same way as for N-version programming (see Detect Internal Failure below).

2. Check task_descriptor of failed task for list of cooperating tasks.

3. Update list of cooperating tasks in alternate version.

Data Structures

1. task_directory

   -- directory of tasks required for each AASC function.

2. task_descriptor
   -- list of characteristics, status, etc.
   --   maintained for each task by operating system,
   --   including cooperating_tasks list.

3. cooperating_tasks

   -- list of all tasks with which this task
   --   exchanges messages.

Start Task

Techniques

1. Determine the state of cooperating tasks.

2. Establish communication channel with cooperating tasks.

3. Begin specified task.

Data Structures

1. cooperating_tasks

   -- list of tasks with which this task exchanges messages.

Report Maintenance

Techniques

1. Create maintenance_event_descriptor

2. Write maintenance_event_descriptor on event_log.

Data Structures

1. maintenance_event_descriptor

   -- record containing
   --   layer_name, time, failure_
   --   description, failure_permanence,
   --   recovery_status

2. event_log

```
-- data base of events
--      keyed on time, layer_name or
--     function_name.
```

Layer 3: Module Integrity

Detect Internal Failure

Techniques

1. Determine assertions to be tested at significant points in each procedure. If an assertion is false, assert task_failure.

2. Determine an assertion to be tested at completion of each procedure. If the assertion is false, assert task_failure.

3. On processor interrupt, assert task_failure.

4. Bracket each procedure with pre- and post-conditions. If either condition is false, assert task_failure.

5. N-version programming: Two or more specifications and implementations of the same function are executed. Their outputs are compared. If they do not agree within specified tolerance, assert task_failure. If there are two versions, the task_failure_description must name both tasks and indicate that one or both have failed. If there are three or more versions, and a majority agree within tolerance, the task_failure_description names the minority task(s). The voting on outputs may be accomplished in at least two ways:

   1. distributed voting

   2. an independent comparator task.

   In both cases, voting is done by message exchange.

Data structures

Dependent on procedure.

Output

1. task_failure
   -- boolean

2. task_failure_description
   -- record naming task, procedure,
   --   assertion_label, other parameters.

Detect Missing Message

Techniques

1. Check flow_control_field on message. If it is not successor of previous flow_control_field, assert task_failure for channel task. This is a standard feature of the network layer of the communications protocol to be used in the AASC. (See networking section.)

Detect Corrupted Message

Techniques

1. Compute check sequence on message and compare with sender's check sequence.
   If not equal, assert task_failure for channel task. This is a standard feature of the link layer of the communications protocol. (See networking section.)

Detect False Message

Techniques

1. Acknowledge the message. If the sender sends an error message in response to the acknowledgement, then the message was false.

Data structures

1. Error_message
   -- message indicating previous acknowledgement was
   --   redundant.

Suspend Task

Techniques

1. Raise an exception in the task. This causes invocation of an exception_handler and suspends execution of the task.

Inform Cooperating Tasks

Techniques

1. For each task on cooperating_tasks list, send a message terminating cooperation. This is done by the exception-handler for the task.

Data Structures

1. cooperating_tasks

```
-- list of all tasks with which this task
--    exchanges messages
```

Output

1. termination_message

```
-- message containing
--    flow control information for last
--    message received and processed, and
--    termination indicator.
```

Find Checkpoint

Techniques

1. Search task_history for most recent checkpoint entry for this task. If found, assert checkpoint_found, otherwise negate checkpoint_found.

Data Structures

1. task_history

```
-- data base of checkpoint_entries
--    keyed on task name, time
```

2. checkpoint_entry

```
-- record containing task_name, time,
--    task_context_description
```

3. task_context_description

```
-- the format of this item is dependent on
--    the task and known to the task.
--    It contains all data items
--    needed to restore the task to its state
--    at the time of the checkpoint, except
--    for messages sent to other tasks after
--    the checkpoint time.
```

Output

1. checkpoint_found

   -- boolean

Report to Cooperating Tasks

Techniques

This function will be carried out in the exception-handler for the failed task. It is similar to Inform Cooperating Tasks, except that the message sent is an initiate_cooperation message.

Reconfigure Task

Techniques

1. Examine processor_configuration_table to determine which processors are available.

2. Assign a free processor to the task. The processor may be a GDP, IP, NPU, or other deterministic unit. This re-assignment may be done by hardware or by the operating system nucleus.

3. Update the processor_configuration_table entry for the processor assigned.

Layer 2: Deterministic Control

Detect Determinism Failure

Techniques

1. Redundancy: Run the processors simultaneously in lock-step, with a comparator function on processor outputs. When outputs disagree, generate a signal. If redundancy is triple, then generate a signal indicating the faulty processor.

2. The hardware will execute a transition to an "error detected" state.

Data Structures

1. faulty_processor

   -- boolean, associated with each processor.

Isolate Determinism Failure

Techniques

With the deterministic component in an error_detected state, and the fault_detected indicator for the failed processor asserted, the component is disabled from produce output, hence is isolated from the rest of the system.

Layer 1: Physical

The functions at this level are almost invariably carried out by hardware.

3. STRUCTURAL HIERARCHY

3.1 Introduction

The structural essence of a computer system is generalized as a network. Modern computers cannot function properly without some form of linkage between themselves, or between themselves and other functional nodes of a system. Computer networking is a rapidly maturing discipline. Most issues which used to be a major impediment a few years ago have been solved or are being solved. Today, networks of various size, form, and characteristics are either readily available or may be built out of high performance building blocks.

Almost two decades of experience in the active use of computer-based communication systems has resulted in various levels of standardization. Amongst them, the Open Systems Interconnection (OSI) reference model serves the role of standard bearer. The layered structure of the model has amply demonstrated its adaptability to the reality of machine to machine and machine to human communication. An excursion in concepts supported by this experience is the creation of a meta-layer structure of networks. As described in Section 3.3 below, there are several networks of different geographic coverage that span any conceivable spread of human activity. The know-how of proper use of computer communication is rapidly becoming the synonym for finding one's place in this hierarchy.

Because one can depend on the OSI and as the OSI is a generalized scheme of interconnection, the size of the network has a decreasing impact on the design of the interconnection scheme for a specific application. In fact, in many cases, the same set of software and hardware modules may be used to arrange interconnections using networks of several different sizes. The recursive nature of the network control structure (e.g., the same interconnection modules may be used on networks of different sizes) will not only simplify the network usage but will result in a drastic increase in flexibility and reliability of the building blocks. The trend is being expanded to provide flexibility to other aspects of the protocol, such as speed, error detection schemes, access methods, data size, and in some cases, network topology. We are progressing towards the day of building networks to really suit the application, and not vice versa, while maintaining strict adherence to global standards.

Interpreted in fault-tolerant computing, this will only increase our chance to implement in reality, discoveries made in theoretical fields, particularly those achieved in the area of software fault-tolerance. A drastically increased reliability of network building blocks due to their implementation in VLSI, is also helping the cause.

Sections 3.2 and 3.3 below summarize observations made on the state of the art in networking standardization. Sections 3.4 through 3.7 cover the way the AASC will implement interconnections.

3.2 Definition of Structural Hierarchy

Section 2 described the detailed funtional decomposition of the autonomy of the AASC. In implementing this complex of functions, it is necessary to fit this decomposition to the constraints imposed by the following:

1. Spatial location
2. Timing
3. Availability of algorithms
4. Historical division of structure.

Nevertheless, it is possible to induce a layering hierarchy of structures in the AASC and its environment.

The structural layering is based, because of constraints 1 and 4 above, on the physical size of the structures involved. The divisions between layers are not clear-cut but certain meta-rules can be defined.

1. A layer Li consists of a set of nodes and a set of links (channels) connecting nodes; in other words, a network.
2. A node in layer Li is a network in layer Li-1.

   Conversely, a network in layer Li is a node in layer Li+1.
3. The protocol for communication is independent of i; that is, uniform across all layers.

The advantages of layering have been stated in Section 2. The advantages of the above meta-rules for the structural layers are:

1. Economy of design effort

2. Minimization of inter-layer conversion

3. Simplicity, which implies reliability and greater potential for enhancement.

3.3 Structural Hierarchy

For each layer of the hierarchy, we will give the physical size-range and characterize the node types and link types. The hierarchy as applied to ground-based networks is shown in Figure 3.1a.

The nodes will be single spacecraft and a gateway on an Extra Global Network. The links will be established by radio beams.

Extra Global Network (EGN)(10**2-10**6km)

The nodes will be single spacecraft or a spacecraft in earth orbit, and a gateway on a Global Network. The links will be supported by radio beams.

Global Network (GN)(40,000km):

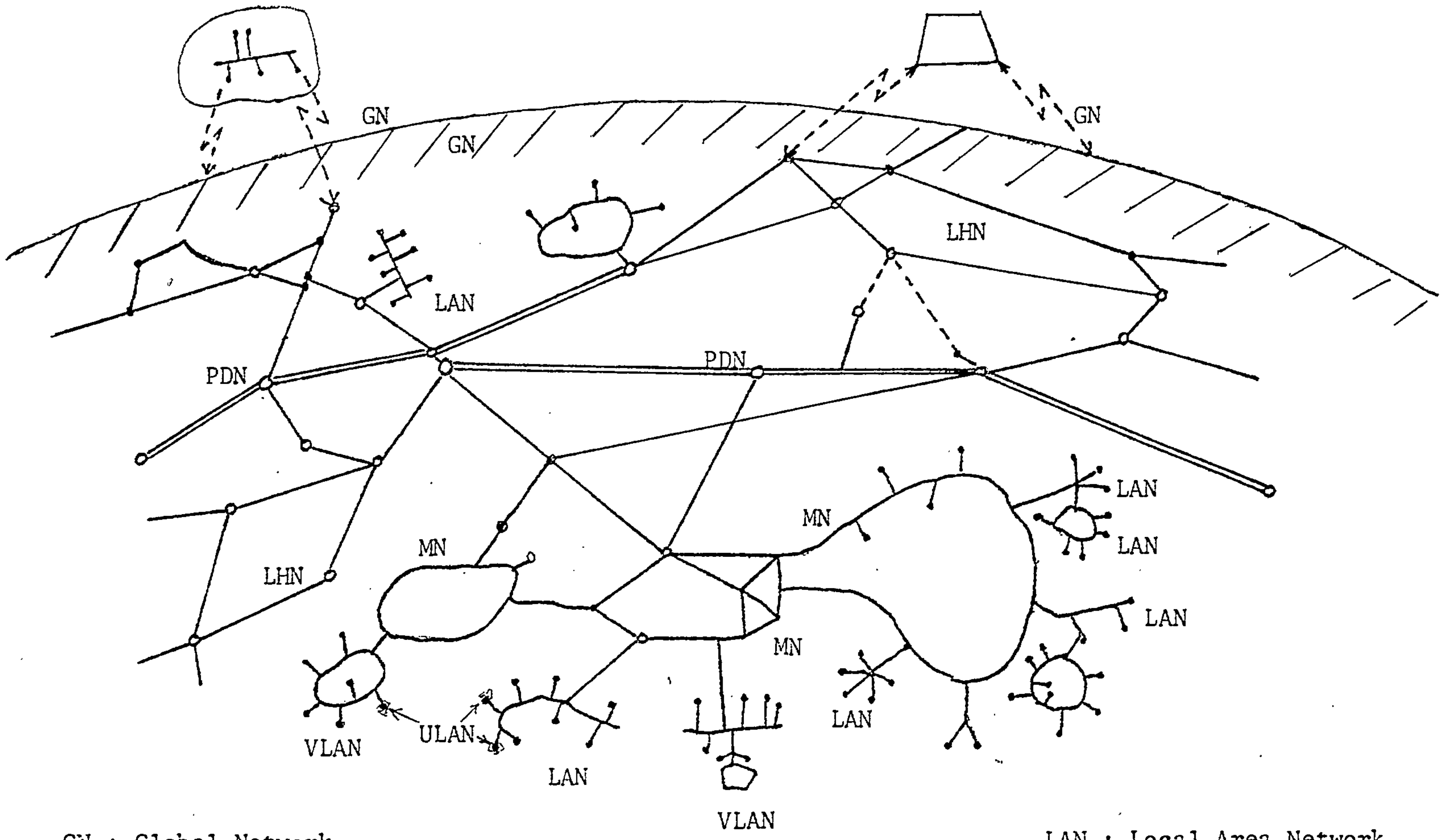
The nodes are gateways on networks of all sizes smaller than GN. The links are radio-frequency, microwave, wire, or coaxial cable. The AASC may interact with a global network in order to communicate with ground control or with another orbiting satellite that is obscured by the earth.

Cross-Country Network (CCN)(150-6000km):

The nodes are smaller networks as described below. The links are the same as for GN. The AASC may interact with CCN when it supports national communications functions.

Long-Haul Network (LHN)(100-500km):

Nodes are smaller networks and links are the same as for CCN. The AASC may participate in a LHN as part of a group of spacecraft engaged in a coordinated function within the given distance range, for example, the rendezvous of a satellite with a shuttle.



GN : Global Network

CCN: Cross-Country Network

LHN: Long Haul Network

MN : Metropolitan Network

LAN : Local Area Network

VLAN: Very Local Area Network

ULAN: Ultra Local Area Network

Figure 3.1a The Hierarchy of Networking as applied to Ground-based Systems.

Metropolitan Network (MN)(10-150km):

Nodes are LANs or smaller. The links are the same as for LHN, plus laser and infra-red. The AASC may consist of a metropolitan network if it is used during construction of a space station structure in the given size range.

Local Area Network (LAN)(50m-10km):

The nodes are processor clusters and the links are wire, optical fibres, infra-red, laser, or coaxial. The AASC must be able to be configured as a LAN, since spacecraft size could well be in this range.

Very Local Area Network (VLAN)(1-50m):

The nodes are processor clusters or single processors. The links are wire or coaxial or optic fibre. Individual AASC subsystems, for instance an antenna control system, fall into this category.

Ultra Local Area Network (ULAN)(10cm-1m):

The nodes are processors or processor complexes or deterministic devices. An individual physical backplane within the AASC falls into this category. The links are wires.

Module Network (MDN)(10**-1mm - 10cm):

The nodes are tasks executing in processors or microscopic connections within electronic components. The links are hence logical channels between tasks, though it is possible to define the physical basis upon which such logical channels execute. All AASC functions will be supported by modules and networks of communicating modules.

3.4 External Links

The AASC achieves external communication through the control of the Ground Gateway of the OAM. The actual exchange of signals is, however, achieved through the on-board communication subsystem. The link will be established through directed radio-beams. The technology to combine the OSI with the packetized radio system is already available. By adopting a gateway scheme which supports the OSI structure, the AASC will become a node of a global network scheme. This simplifies the exchange of information between the AASC and other stations anywhere in the global networking scheme of the external link of the AASC. The slight deviation from the concept implied in the Packetized Telemetry developed by NASA [NASA 82] is due to the awareness of the importance of this global incompatibility.

The high bandwidth link will be sufficient to satisfy all on-board needs and most ground needs, particularly if on-board data reductions are considered.

3.5 Subsystems Interconnection

3.5.1 Introduction

The most dominant communications within the AASC are those between on-board subsystems. On-board subsystems are those spacecraft modules with clearly defined independent functions. Examples of a subsystem are: attitude control subsystem, thermal control subsystem, antenna orientation control, power management control. They typically consist of hardware and embedded software.

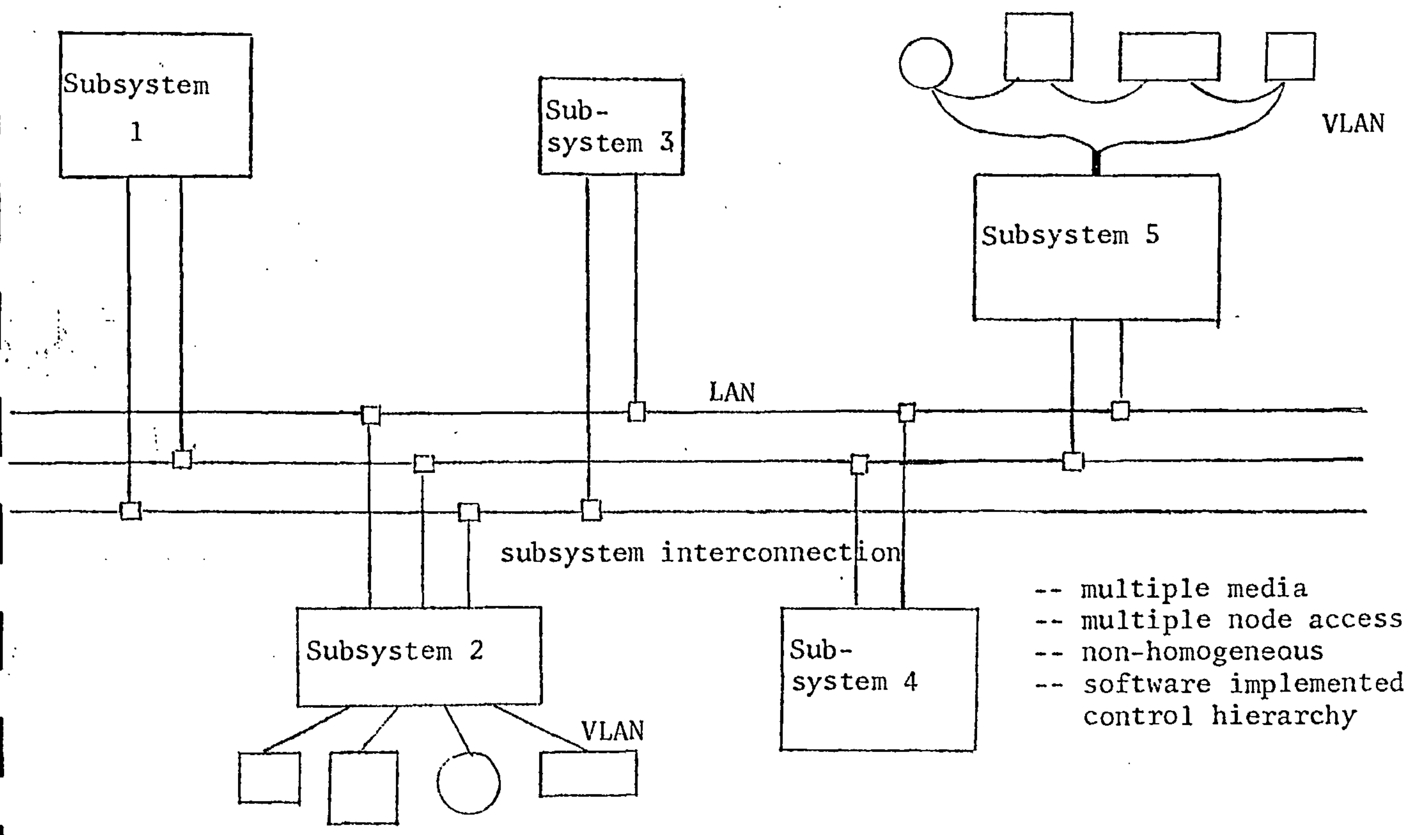
3.5.2 Mode of Communication

Figure 3.1b shows a typical on-board inter-subsystem communication. Such communication is characterized by a relatively high level of abstraction at which stations (subsystems) exchange messages. In terms of the OSI reference model, middle to upper level protocols become of importance here. Procotol layers TRANSPORT, SESSION, PRESENTATION, and sometimes APPLICATION, will have to be implemented to support the subsystem inter-connection on-board the AASC. In contrast, lower layers such as the LINK and the PHYSICAL layers will be almost invisible ,as such layers will be treated at module level.

3.5.3 Traffic Volume

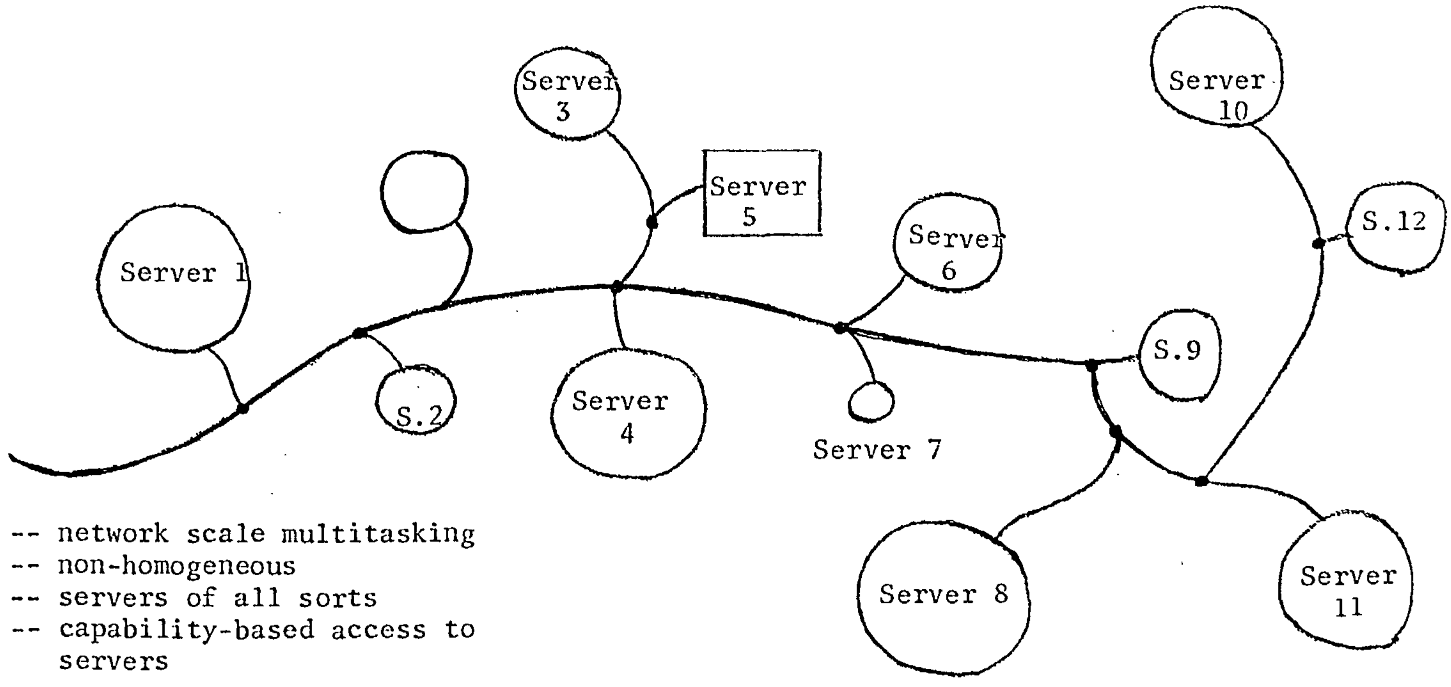
The majority of messages exchanged will be short and relatively abstract (preprocessed) in nature, except in some localized areas. In and around image processing subsystems and other high data-rate systems, bandwidth requirement will have to be higher. To cope with this imbalance, VLANs should be used to absorb locally heavy traffic. Subsystems which require higher bandwidth are encouraged to process high volume data locally and exchange only the results of the data reduction using inter-subsystem linkages.

By limiting the level of abstraction at which exchange



-- multiple media
-- multiple node access
-- non-homogeneous
-- software implemented control hierarchy

Figure 3.1b Typical On-board Inter-Subsystem Communication



-- network scale multitasking
-- non-homogeneous
-- servers of all sorts
-- capability-based access to servers

Figure 3.2 The Dynamic Server Model

takes place between subsystems, high traffic management becomes simpler. The lack of the need to transmit high volume raw data across the spaceship (except in emergencies such as auditing of the on-board vision system from the ground) will contribute to improved reliability of subsystem level communication.

3.5.4 Homogeneity and Dynamism

There will be a wide variation in physical size, shape, and function among subsystems (e.g. an AOCS and a power regulator will be drastically different from each other in these attributes). For this reason, the on-board network becomes non-homogeneous. Since the subsystems will be exchanging messages of relatively short size, the mode of operation will look much like conventional multi-tasking models seen in single processors, except in this case, the operation is expanded to the entire network. The implied level of dynamism in the network, therefore, will be high in spite of the lower over-all traffic need. This "multitasking" mode of operation is obvious as virtually all subsystems will be operating concurrently most of the time. In a situation like this, the server-oriented modelling proposed by the Liberty Net project [KING 82] fits nicely, as seen in Figure 3.2. There will be workstations (subsystems) of various sorts conceptualized in functional terms on board. Such workstations are, indeed, servers (e.g., power-regulation servers, on-board communication control servers, on-board file servers). These will be intermixed with logistic servers such as message authentication servers, file update servers, ground traffic monitor, error-log servers. Personification fits well in this type of modelling.

3.5.5 Application

In future spacecraft, on-board data handling will become increasingly sophisticated. Activities seen in offices, factories, and laboratories on the ground will be gradually seen in orbit. The implication is that whatever technology we are developing now in Factory Automation (FA), Office Automation (OA), and Laboratory Automation (LA) will eventually find its way into space.

The dynamic server concept is ideal for all these applications, as it operates at a higher level, independent of the underlying topology. A hypothetical wired laboratory is shown in Figure 3.3. Ultimately, this is an example of the realization of several concepts similar to those contemplated by the communication industry today. "Messages" in these cases are defined as "anything that can be digitized". Thus, a "message"

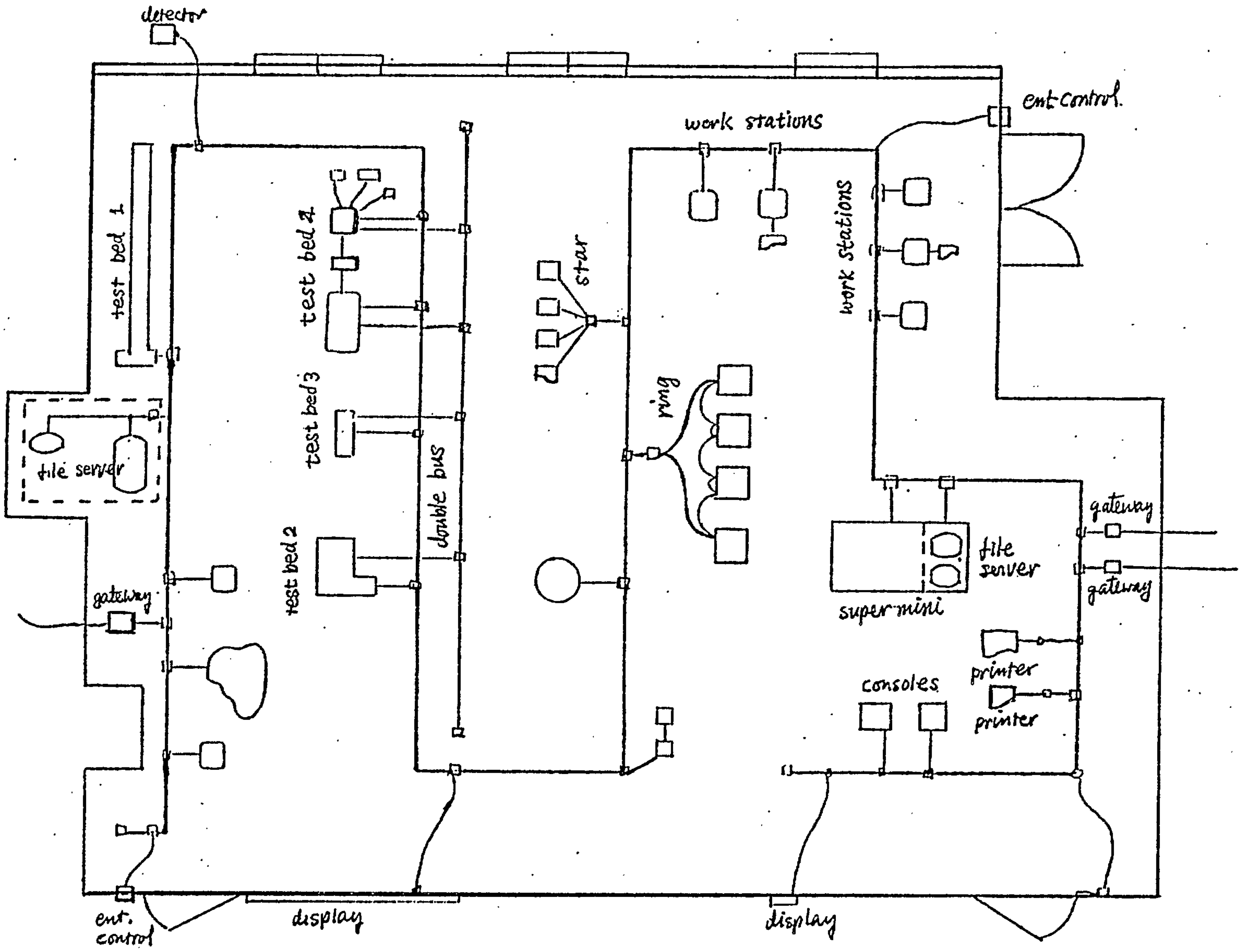


Figure 3.3 An Example of a Wired Laboratory

may be as complex as a frame of high precision color graphics with a voice comment.

3.5.6 Type of Network

3.5.6.1 Topology

In most cases, no "store and forward" processing will be needed in networking within the AASC as the topology is relatively fixed and, hence, routing will not be a major issue. Freedom to choose a suitable topology (bus, multiple bus, ring, star, or any combination of these) according to application need is guaranteed, thanks to the layering scheme of the OSI. On-board networks will be formed to fit the application, and not vice versa. In applications where the network reliability is stressed, effort to avoid singularity (such as seen in a single bus-type network) in the system will be made.

3.5.6.2 Size of Network

MN, LAN, and VLAN, as defined in Section 3.3 above, are considered as effective means of inter-subsystem communication. This hierarchy of networks will permit the exchange of messages between on-board stations as far apart as several kilometers or those as close as ten centimeters.

3.5.6.3 Procotols

The OSI reference model will be strictly followed. The TRANSPORT layer will be implemented in Classes 0, 1 and 4. The PRESENTATION and APPLICATION layers will be defined and implemented whenever needed, according to individual applications. The TRANSPORT layer will absorb all idiosyncracies that will exist at the lower three layers, as shown in Figure 3.4. The figure demonstrates that various network types may be interconnected without affecting layers above the NETWORK layer.

3.5.6.4 Speed and Bandwidth

Bandwidth requirements will be generally low except where equipment with high data rate (e.g image processing systems) are involved, where high bandwidth channels will be implemented. In many cases, baseband transmissions supporting simpler message exchanges seem adequate. However, considering the need for providing direct access between on-board subsystems, and ground control, a broadband network will be considered for implementation as the trunk inter-subsystem link, at least for the portion of the AASC, where such needs are obvious. Adher-

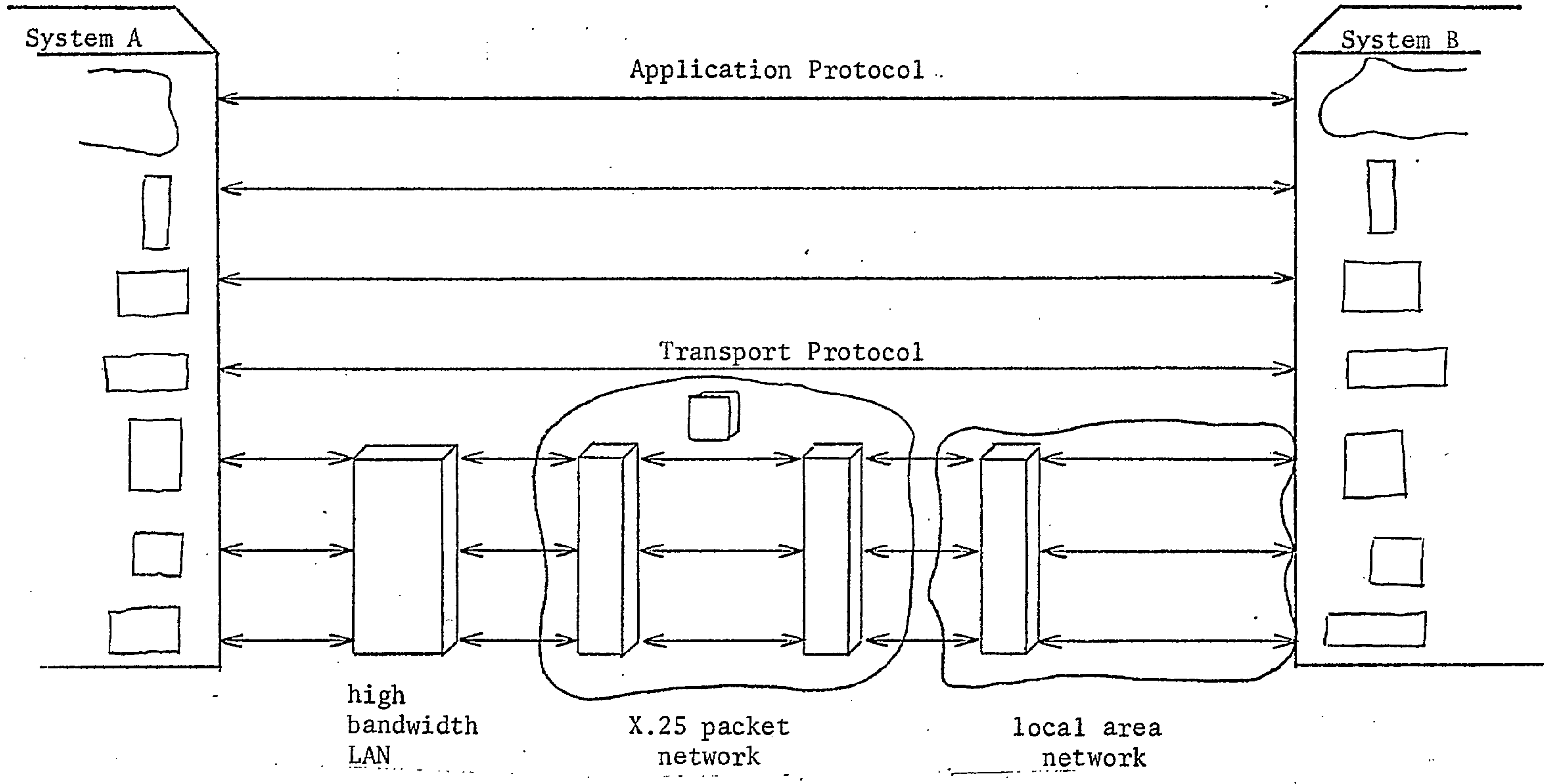


Figure 3.4 TRANSPORT level absorbs lower level idiosyncracies

ence to the standards, even for the lower OSI layers, will be followed.

3.5.6.6 Internetworking

The OSI reference model supports internetworking at the NETWORK layer. Since a mixture of topology, access method, and bandwidth are expected within the over-all AASC, and also the use of gateways as the main method of external communication, extensive internetworking will be used. Standard internetworking schemes supported by the international networking community will be implemented.

3.5.6.6 Construction

The subsystem interconnection will be supported mostly using commercially available components. Components for layers 1 and 2 (the PHYSICAL and LINK layers), are now available commercially. Also, those for layers 3,4, and 5 (NETWORK, TRANSPORT and SESSION layers, respectively) are expected to be available in a short while. Mainly for reliability reasons, these off-the-shelf components will be used in preference to privately developed modules. The resulting immediate cost saving will be noticeable. Of most importance, however, the long range benefits from complying with the world standard in terms of easier maintenance and training, wider product availability, greater expertise-base to tap on, steadily increasing realibility, and drastically reducing the cost of components, should be recognized. If our language is different from that of the majority of the community, we will not be able to benefit from their experience. The opportunity will be far more important than benefits that would be realized by trimming the standard privately for our own needs.

3.6 Module Interconnection

Within the AASC, several modules cooperate to accomplish a given function or service for the application processes which perform the mission-oriented functions of the spacecraft. The term "module" can refer to a single task or a package of tasks, and is synonymous with the word "process".

The fact that several modules cooperate implies that there is a logical link between modules, in that the modules must exchange information. Hence, a set of cooperating modules can be viewed as a network. When the modules are in the different processor complexes, the network is a conventional one, in the sense that there are physical links joining the processor complexes, and hence joining the modules. However, when the modules are within a processor complex, the physical links are transparent, being the internal circuitry of the processor complex. Hence, the links joining modules within a processor complex are strictly logical as far as system design goes.

The indeterminate nature of the link between two modules must be made transparent, since one of the techniques used to attain reliability is the ability to execute modules in any of several processor complexes.

Fortunately, the OSI model chosen for networking at all layers allows exactly this kind of transparency.

We define communication between two modules at the Transport layer of the OSI reference model, shown in the Appendix. This provides all the services required for inter-module communication:

- connection establishment
- full duplex
- flow control
- unbounded size data units
- connection termination.

The possible links between modules are established when the modules are designed. That is, for each module there is a list of cooperating modules defined, and a possible channel between the module and each possible cooperating module. A channel is a module itself, and its functions are:

1. accept a request from a module to
    - establish a connection with the other module
    - send a message
    - send a message if the channel is not full
    - receive a message
    - receive a message if the channel is not empty
    - terminate connection.
2. Perform flow control checks.
3. For connection establishment, determine if the other module is in this processor complex or another. If it

is in this complex, create a Network and Link module between the requesting module and the destination module. The link module can be implemented, for instance, as an Ada port.

If the destination module is in another complex, establish a connection with the Transport layer in the destination processor complex via the LAN Network, Link, and Physical layers.

In either case, record the routing method so that future Send and Receive requests will use the correct route.

4. For Send, Receive, and Terminate requests, use the intra-processor or inter-processor Network and Link levels as established above.

Connection establishment requires the specification of the receiver's logical address defined within the AASC.

The Transport level will be coded with class 4 service. That is, it must recover from resets or disconnects and retransmit lost data. Other error detection and recovery is done at the link level, using a cyclical redundancy check sequence.

3.7 Element interconnection

3.7.1 Introduction

The term "element" is used here to describe functional units within the AASC that are smaller in dimension than modules described in Section 3.6 above. While elements can be either software or hardware, the issue of interconnection software elements of this dimension, are well covered by the discipline of software engineering. Hence only the interconnection of hardware elements within the AASC is discussed.

3.7.2 Serial Backplanes

As shown in Figure 3.5, the size of the network is extremely small. It fits the description of Ultra Local Area Network, or ULAN, described in Section 3.3 above. Typically, the entire length of such network is less than a few meters. Since this arrangement neatly replaces the traditional backplane that held hardware components in a cage-like arrangement, the increased flexibility and the associated ease of construction is ob-

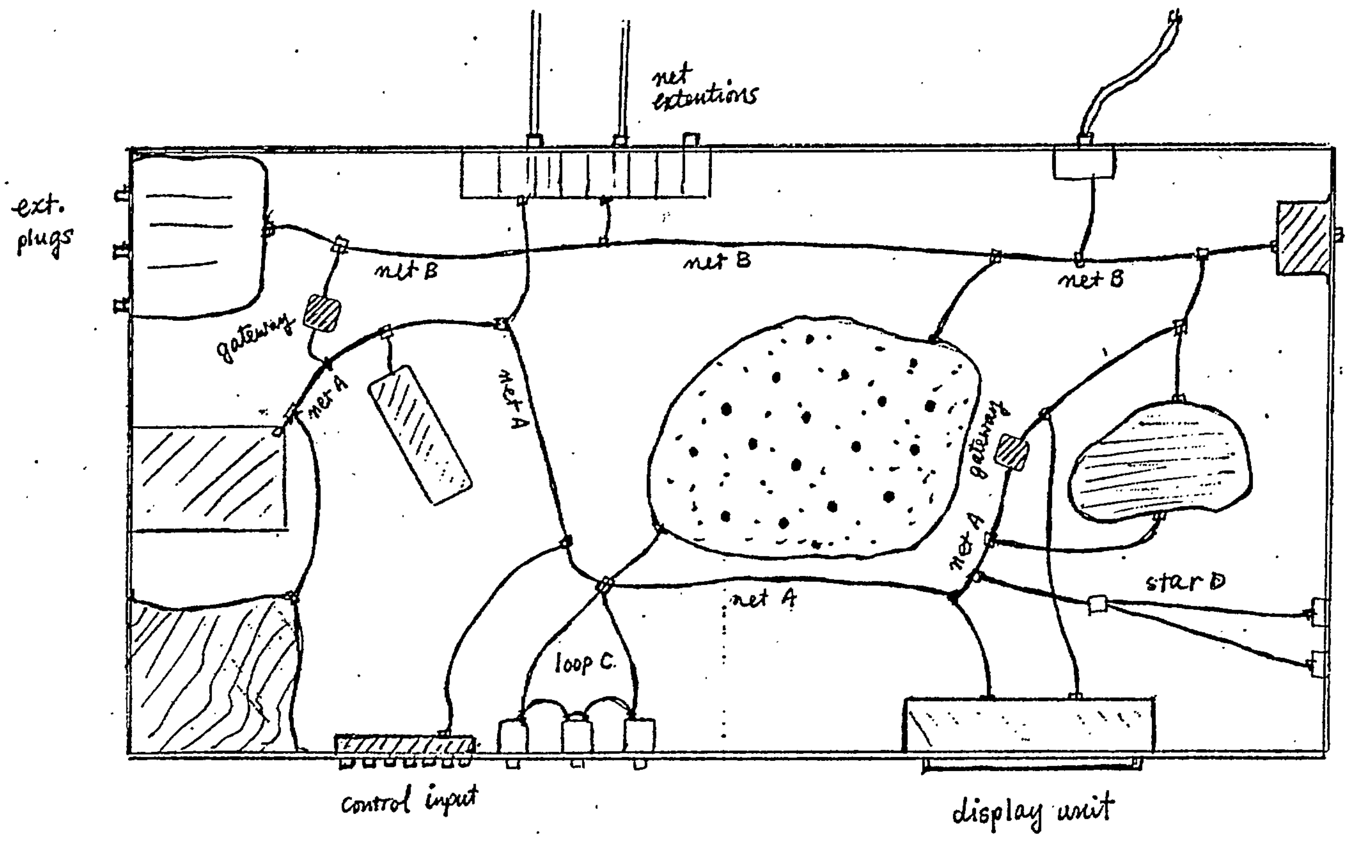


Figure 3.5   A Hypothetical Application of the Serial Backplane Concept

vious. Another aspect of the network is its high speed, as it replaces the transmission media which were running at or around the execution cycles of conventional processors and memory units. The network will probably be run at a speed between 5 and 30 MBPS. Lower speeds may be possible in some cases, as the level of intelligence supported by each element is steadily getting higher, and hence the likelihood of exchanging "messages" in place of blocks of raw data is high. By replacing the arrangement of separate address, data and control lines common in conventional backplanes, functions of all these channels must be implemented within a protocol of packetized network. This is not a problem as most existing protocol schemes, including the OSI, already have this provision in their definition. Addressing is achieved by the destination address field in the protocol frame, with the added flexibility of being able to specify the source address. Similarly, the data field of a packet replaces the data lines, with the added flexibility which variable length data fields can provide. The functions of the control lines are simply supported by control codes that fit in the data field of the packet, only with a far greater selection of control possibilities than those which the normal dozen or so control lines can support. The replacement of several dozen copper strips by a pair of flexible wires implies a greater fault-tolerance, as improved decoupling between elements and less exposure to the environment are achieved.

3.7.3 Flexibility

In addition to the remarkable flexibility stressed above in packetized data handling, the flexibility achieved in other domains is obvious. For instance, the cable can be twisted at will, allowing a more functionally-oriented arrangment of components within a package.

The packet exchange is completely asynchronous, as opposed to the relatively strict timing requirements still existing in so-called asynchronous backplane buses. Elements are completely independent in their physical position on the bus. The types of transmission media that can be used with the scheme is also left to the choice of the system designer. Some of the VLSI controllers available for this use would permit the choice of coaxial cables of various specifications, twisted wires, or even naked wires. Noise hazards from EMR (electro-magnetic radiation) would be no worse than the conventional backplane, while the greater ease with which the channels can be shielded is a welcome option, as this will, in addition to protecting the environment from EMR, increase the fault-tolerance of the network against external noise.

Flexibility is also seen in the software controlled parameters of such networks supported by VLSI controllers. Not only are the size of packet, addressing field, data field, and the transmission speed programmable, but parameters such as the lengths of the preamble sequence, and CRC code are also controllable by the software. They can also be programmed to choose the location of buffers, the size, and the number of packets being received or transmitted. This flexibility may be used to organize buffers and control blocks in such a way as to increase the fault-tolerance of data structures.

4. SYSTEM TEST PLAN

4.1 Test Concepts

System testing is seen in the same context as system development. The initial view will be one of functional testing - a functional operation will be viewed as a "black box" with a set of inputs and outputs which will be tested against its operational requirements [HOWD 80]. As testing proceeds through the layers of functional decomposition, each succeeding operation will become the next "black box", while the structure of the layer preceding it will now become visible. Functional testing is defined as having an infinite number of tests which, if applied, will find the total number of errors in the system. Structural testing, conversely, has a finite number of tests which will only find a limited number of errors. Testing of the system, therefore, will attempt to provide a balance between functional testing and structural testing.

A survey of the literature has revealed that the most critical stages of system development, as far as error occurrence is concerned, are those of specification and design. The errors occurring at these levels are the most difficult to detect, costly to correct, and critical in effect. It is proposed, therefore, that considerable effort should be made to reduce and trace such errors. The use of a specification language is of great importance in this attempt. A language such as Ada, used for specification purposes, is in itself diagnostic, as it provides a means of identifying design errors at a very high level by prototyping the system.

A detailed checklist shall be created from the functional specification. This shall be achieved through hierarchical decomposition of the specifications. A hierarchical tree shall be created independently of the functional decomposition done for design purposes. Comparison between the two decompositions will form a test of both processes, since it must be remembered that the testing process, itself, is as prone to errors as that of system development.

4.2 Methodology

- System testing shall begin at the outset of development and continue throughout.

- Testing shall be conducted in a top-down, structured fashion.

- Functional elements, as defined by the requirements, shall be viewed as a "black box".

- Provision must be made for the injection of test data into the system and extraction and recording of the system's operational status, if this is not provided under normal operating conditions.

4.3 Test Requirements

The purpose of testing is to ascertain that the system conforms to the requirements. The system shall be tested until these requirements are met.

System Reliability

The system test shall ascertain that:

a) the MTFF will be 10**9 hours
b) the system failure rate will be approximately 10**9 failures per hour
c) the availability of the system will be (1 - 10**9)
d) the error rate of the system will be 1 incorrect symbol in 10**9.

Flexibility

The system test shall ascertain that:

a) the system will be able to serve an abitrary set of applications on each mission.
b) the system will be able to adapt in flight to changes in mission profile.
c) the system will be able to upgrade or enhance system components and relationships.
d) the system configuration will be modifiable to cover component failures during a mission and has the ability to cope with non-homogenous load distribution.

Fault-Tolerance

The system test shall ascertain that the system has:

- distributed fault detection.
- cooperative fault diagnosis.

- node/link fault discrimination.
- message/state conflict detection.
- physical isolation
- logical isolation
- backward recovery
- forward recovery
- physical reconfiguration
- logical reconfiguration

Performance

The system test shall ascertain that:

a) the minimum throughput of a network linkage between two clusters will be 10 MBPS, or as defined by the requirements for image or voice processing. The communication channel must meet the access requirements for a combination of applications.

b) the through-put of the processor complex unit responsible for the logic and arithmetic operations will be sufficient for all appropriate applications, i.e. a minimum of 1.0 MIPS when properly configured and measured in terms of a 16-bit instruction set or equivalent.

c) the processor unit shall have at least 0.25 MIPS of throughput when executing a mix of basic floating point operations.

d) memory transfer rate between the processor in the cluster and on-unit memory array shall be more than 5.0 megabytes per second. The data transfer rate between the unit and off-unit memory (secondary memory) shall exceed 2.0 megabytes per second.

e) the operating system for the processor unit shall provide a multitasking environment which is transparent to any multitasking scheme, and which will support basic multitasking functions with less than 25% overhead. The maximum time requirement for context switching shall be less than 50 microseconds.

Autonomy Requirements

The system test shall ascertain that:

a) the On-board Autonomy Manager shall maintain the well-being of the the on-board operation of the space-craft, including the proper handling of minor faults. The solving of severe on-board faults shall be attempted in cooperation with ground control.

b) the OAM shall establish communication asynchronously with the ground under the following conditions:-

   - when ground control wishes to query or monitor any aspects of on-board operation

   - when ground control attempts to take over any or all of the on-board management

   - the OAM decides that a significant on-board event requires reporting

   Sufficient archiving storage shall be maintained on-board to accommodate the keeping of information subject to ground audit for whatever period of time is determined for each mission. This storage will be organized in a hierarchical manner to allow on-board archiving to occur in diminishing frequency and quantity with respect to time.

c) The design of the OAM will enable:

   - the later incorporation of an explanation subsystem.
   - the later incorporation of the functions of "knowledge acquisition" or "learning"
   - the maintenance of a reasonable architectural distinction between the knowledge base and the inference mechanism
   - backtracking to be performed on the "global data base" or "scratch-pad"
   - a mechanism to deal with "fuzzy situations" to be attached.

Conformity to the FTC Rules

The system test shall ascertain that conformity to the FTC Rules is maintained at all levels.

Networking Principles and Standards

4.4 Resources

4.4.1 Personnel

The process of testing the AASC should be conducted by person(s) independent of the design team. This element of independence will, of itself, provide a means of testing the design. Any differences between interpretations of the requirements will point to errors of conception, omission, or ambiguity of specification.

4.4.2 Computer Equipment

These features are seen as a necessary part of the environment in which the AASC is to be tested:

- integration: requiring common internal program representation, uniform inter-tool interfaces, and achievability of tools from other tools.

- open-endedness: the ability to provide the user with mechanisms for new tool definitions.

- granularity of tools: every tool corresponds to a specific simple function.

- inter-activeness: based on a simple, powerful command language, a uniform user interface, user-oriented interaction devices.

- multi-user support: a project data base, personal work-stations, an inter-user communication facility.

- host environment: supported by a dedicated machine different from that on which the produced system will run. This would include object code generators, source language program optimizers, target language linker and loader. The specific distribution of tools between host and target environment should be considered in conjunction with the development plan. (A example of an suitable environment is shown in Figure 4.1.)

4.4.3 Simulators

It is proposed to minimize the use of software simulation

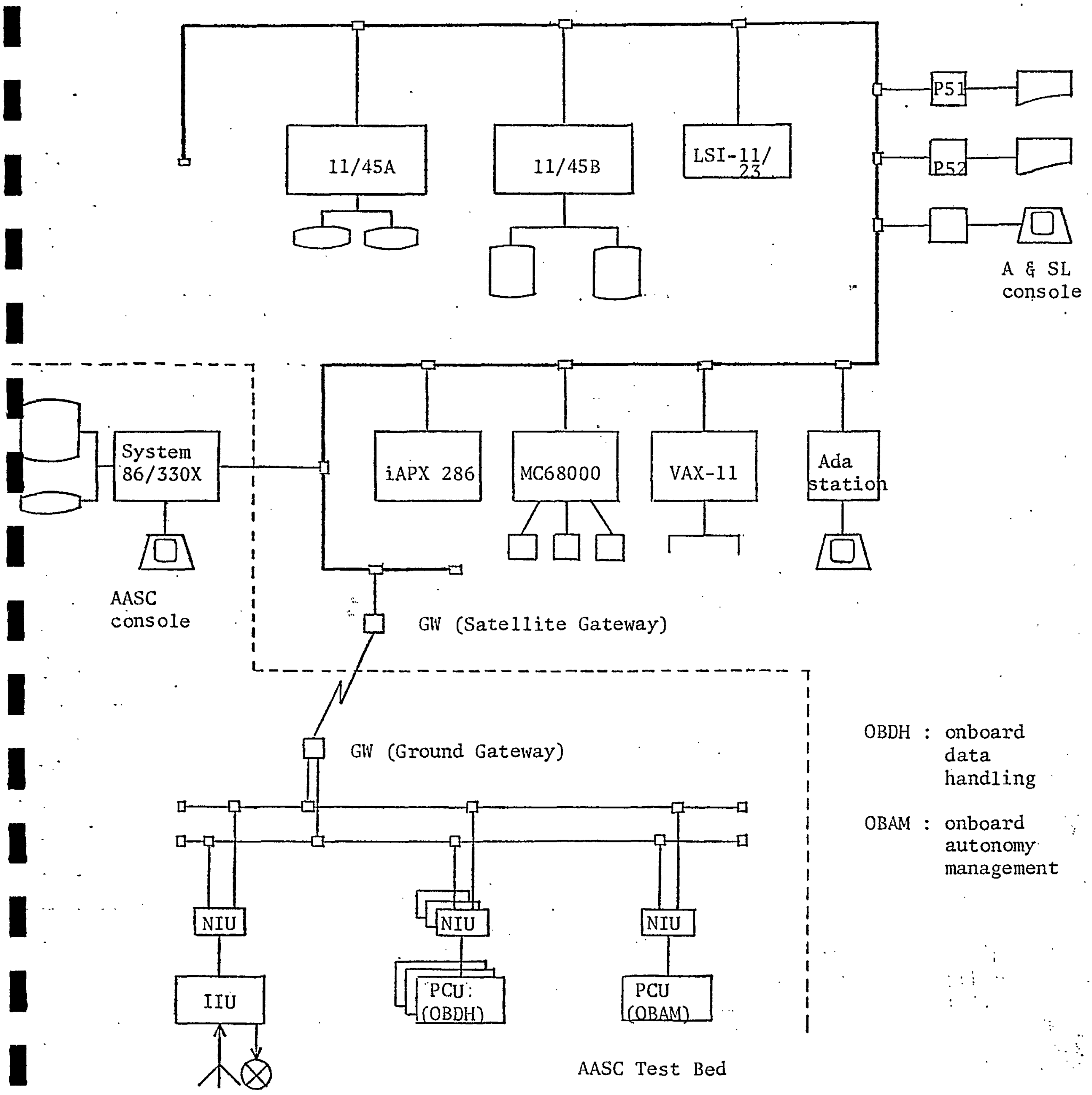


Figure 4.1 An Example of a System Test Environment

of hardware components, since the verification of the behaviour of a program in a highly simulated environment is not adequate when performance is concerned [DEGA 81]. The use of Ada as a development language provides for the use of software packages as a means of simulating the design as it is created, while allowing validation to take place concurrently.

4.4.4 Debugging Aids

In order to achieve the interactive analysis and testing of the system concurrent with its development, the following aids are required:

- a structured syntax directed editor: this allows early detection of syntactic errors, in addition to standard facilities.

- tools for static analysis: granular tools, which allow verification of specific semantic aspects; such as type-checker, symbol table builder, scoper, variable set-use analyzer, aliasing checker, interface checker, static source level linker, without the insertion of test data.

  These tools can be integrated into more powerful testing tools and can also be used to single out and define test data.

- tools for dynamic analysis: these allow programs to be run using specific input data and are based on an abstract syntax interpretive execution tool. The following facilities should be provided by the interpreter: breakpoint insertion and management, single-step execution, in-line statement evaluation, state inspection, assertion evaluation, control over the execution resumption after a break, and program modification.

  A symbolic execution capability (obtained from extending a conventional interpreter) takes symbolic input expressions which describe all possible input data and, through the resulting program execution, determine the admissible input data partitions based on program semantics. The output values of this execution are described in terms of the input values, and show more easily the adherence of the program to the requirements by semantic analysis of the program behaviour. The results of the symbolic execution can also be used to generate standard input test data.

Mutation analysis, an interesting new approach to testing, is cited in a study as achieving great success in the discovery of software errors and is claimed as being a major advance in the area of software testing [BUDD 78]. It has, however, only been implemented for use with FORTRAN, with extensions currently underway for COBOL and C Language, and its adaptation for use with other languages, such as Ada, would be costly at this time.

4.4.5 Test Data Generators

It will be necessary to provide a suitable program suite to generate sets of test data.

5. Conclusion

A generalized hierarchy of system autonomy has been developed, based on a layered model, from the physical layer to an intelligent, knowledge-based layer. This hierarchy has been applied to the design of the Advanced Autonomous Spacecraft Computer (AASC), by projecting the constraints and requirements of the AASC onto the layered model.

A generalized hierarchy of distributed system structure has been developed, also based on a layered model, in the dimension of network size, and this model has been applied to the AASC and its operating environment to produce a structural system design.

Finally, a top-down system test strategy, test criteria, and specification of testing resource requirements has been presented.

REFERENCES

ALBU 81	Albus, James S., "Brains, Behaviour and Robotics", BYTE Books, McGraw-Hill, Peterborough, N.H., 1981.
BUDD 78	Budd, T.A. R.J. Lipton, F.G. Sayward and R. DeMillo, "The Design of a Prototype Mutation System for Program Testing", AFIPS Conf. Proc., Vol.47, 1978 NCC, pp.623-627.
DEGA 81	Degano, P. and G. Levi, "Software Development and Testing in an Integrated Programming Environment" from "Computer Program Testing", SOGESTA 1981, eds. Chandrasekaran and Radicchi, North-Holland Pub. Co.
DIJK 65	Dijkstra, E., "The Structure of THE multiprogramming system", Comm. ACM 11, 5 pp.341-46, May 1968.
GOMI 82a	Gomi,. T. and M. Inwood, "FTBBC - The Fault-Tolerant Building Block Computer", Eidetic Systems Corporation, 1982.
GOMI 82b	Gomi, T. and M. Inwood, "A Fault-Tolerant On-Board Computer System for Spacecraft Applications", Eidetic Systems Corp., 1982.
GOMI 83	Gomi, T., M. Inwood, I.McMaster and I.Cunningham, "A Functional Specification for the Advanced Autonomous Spacecraft Computer", Eidetic Systems Corp., 1983.
HOWD 80	Howden, W.E., "Functional Program Testing", IEEE Trans. Software Engineering, Vol. SE-6, No. 2, March 1980, pp.162-169.
KENT 81	Kent, E., "Brains of Man and Machines", BYTE Books, McGraw-Hill, 1981.
KING 82	King, K.J., P.A. Lee, and F.Maryanski, "An Architecture for Local Network Servers", COMPCON 82, September 1982.
MYLO 83	J. Mylopolous, T. Shibahara, and J. Tsotsos, Dept. of Computer Science, University of Toronto, 1982
NASA 82	NASA/ESA Working Group for Space Data Systems Standardization (NEWG), "Joint NASA/ESA Packet Telemetry Guideline", January, 1982.

SHIB 83	T. Shibahara, Dept. of Computer Science, University of Toronto, "CAA: Computer Diagnosis of Cardia Rhythm Disorders from ECCs", Ph.D Thesis.
TSOT 80	J. Tsotsos, Dept. of Computer Science, University of Toronto and Div. of Cardiology, Toronto General Hospital, "Temporal Event Recognition: An Application to Left Ventricular Performance"
UOFT 82	Dept. of Computer Science, University of Toronto, "PSN/1 User"s Manual: "Above all - DON'T PANIC", April 1982.
VESE 81	Vesely, W.E. et al, "Fault Tree Handbook", Systems and Reliability Research Office of Nuclear Regulatory Research, U.S. Nuclear Regulatory Commission, Washington, D.C., January 1981.

APPENDIX

TRANSPORT Service and Protocols

Figure A.1 shows the relationship of the TRANSPORT Services and Protocol with the OSI reference model.

TRANSPORT Services

- connection establishment
- full duplex (normal and expedited)
- flow control (normal and expedited)
- expedited data (a separately flow-controlled path in each direction that permits small units of data to be sent). This is shown in Figure A.2.
- normal data: allows an unbounded sized data unit to be sent.
- termination is via "clear" i.e. data may be lost (SESSION has a "close" which ensures no data is lost).
- sequencing.

TRANSPORT Protocol Class Strategy

- Where the NETWORK service provides the services required by the use of TRANSPORT and the quality of the NETWORK layer service is adequate to meet the needs of the application, then a TRANSPORT protocol may be selected that has minimum functionality (i.e. it propogates upward the NETWORK service).
- Where the service or quality is not adequate then a close of TRANSPORT protocol is selected to enhance the NETWORK services and quality.

TRANSPORT Protocol Classes

Class 0:

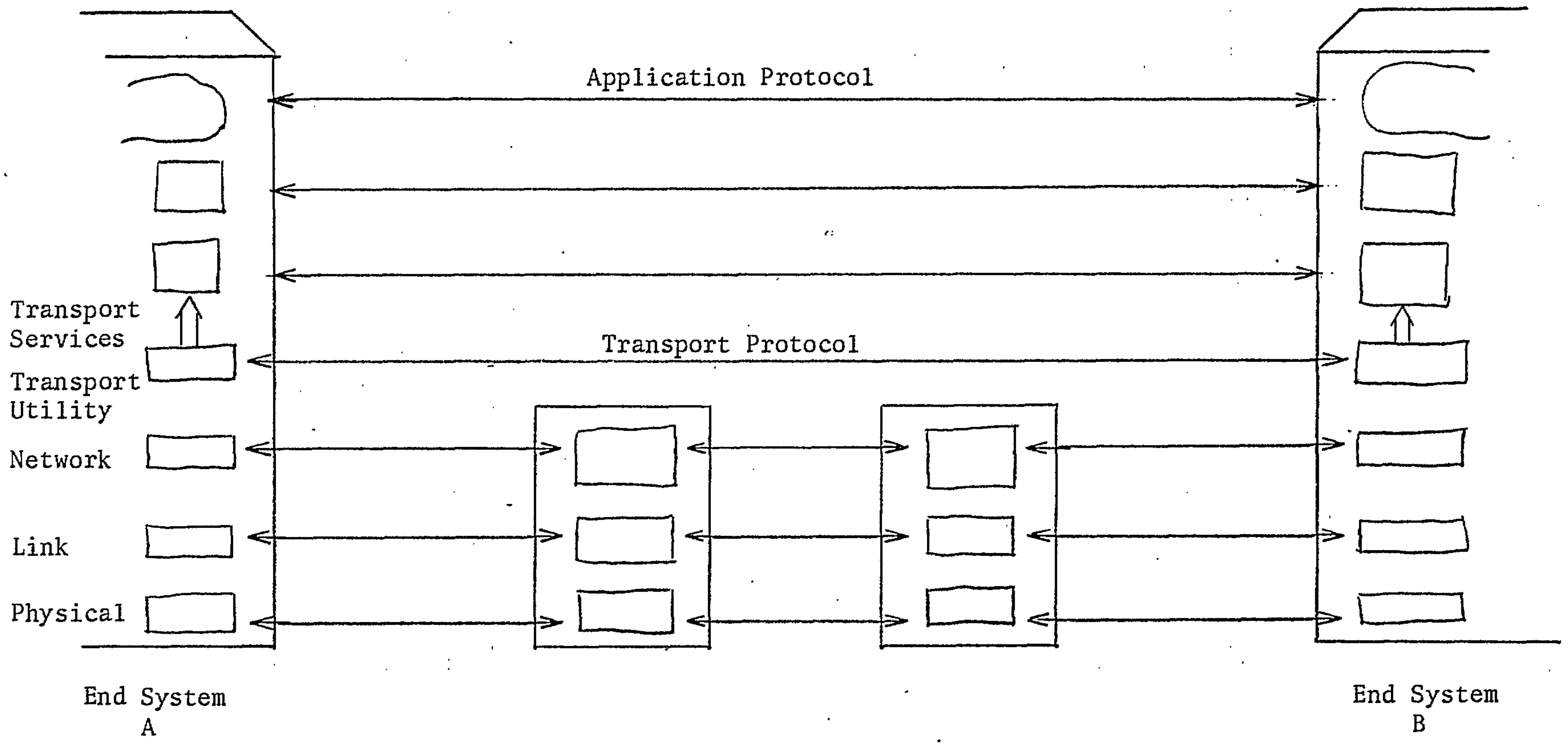


Figure A.1 TRANSPORT Service and Protocols

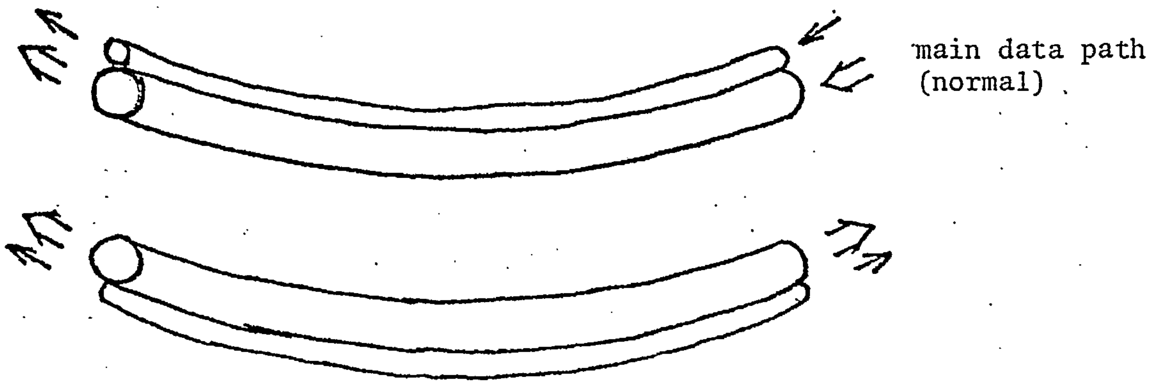


Figure A.2 Expedited Data Path

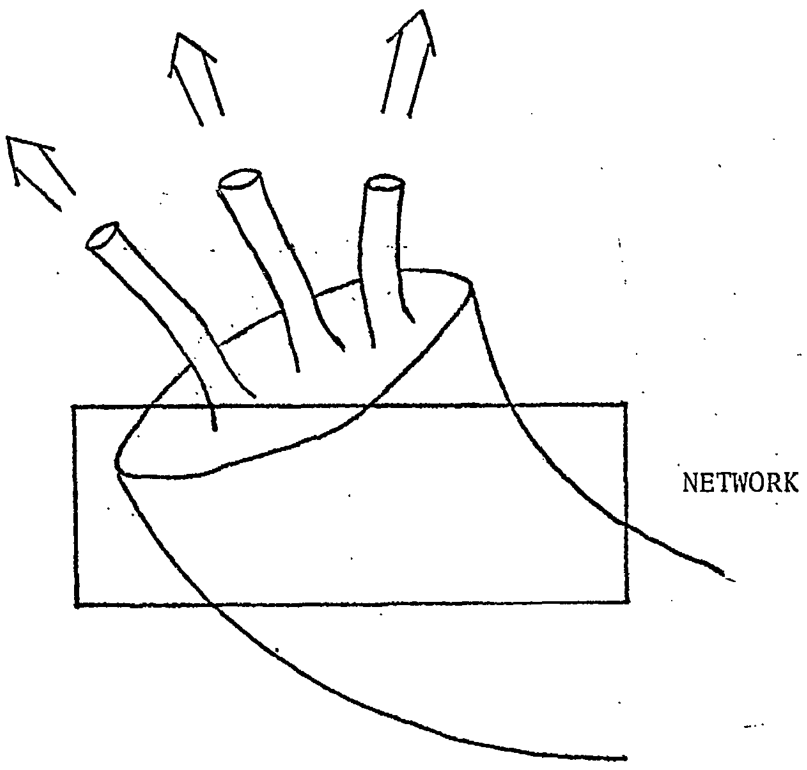


Figure A.3 Multiplexing of Data Between the NETWORK and TRANSPORT Layers.

Service	- provides full duplex normal data - no expedited - clear
Functions	- none, all services are propagated up from the NETWORK layer.

Class 1:

Service	- provides full service
Function	- propagates all services up from the NETWORK layers. - if there is a reset or disconnect of the NETWORK layer connection, TRANSPORT will initiate another NETWORK layer connection: determine what data was lost (re-synchronize) and re-transmit missing data.

Class 2:

Service	- full service
Function	- multiplexes a number of transport connections on to a single NETWORK layer connection. This saves money where many long duration, low volume contains must be maintained on a time charged NETWORK connection - failures are not recoverable - segmentation.

Class 3:

Service	- full service
Function	- multiplexing as in Class 2 - failures are recoverable - segmentation.

Class 4:

Service	- full service
Function	- same as Class 3 - error detection - error recovery - sequencing - segmentation

Notes

1. Multiplexing requires that the TRANSPORT entities send protocol data units to control the flow on each connection to ensure that flow control applied to one connection does not affect the flow on others, as shown in Figure A.3. If there is no multiplexing, then the transport entity can provide flow control by using the flow control of the NETWORK layer.

2. Error recovery in Classes 1 and 3 relies on errors being detected in the NETWORK layer. Class 4, however, does its own error detection.

3. Class 4 assumes a minimal NETWORK layer service, i.e. datagrams.

4. Class 1 relies on an "ACK" provided by the NETWORK layer. Classes 3 and 4 send their "ACKs" as protocol data units.

Table A.1 Summary of TRANSPORT Functions

Class	FC	Mpx	Seq.	Seg.	Error Det.	Error Rec.	Explicit Ack.
0							
1						v	
2	v	v		v			
3	v	v		v		v	v
4	v	v	v	v	v	v	v

GOMI, T.
--The design of an advanced autonomous
spacecraft computer: technical report

P
91
C655
G644
1983
