SAE Aerospace
An SAE International Group

AEROSPACE INFORMATION REPORT

SAE AIR5315

Issued 1998-04
Reaffirmed 2011-04

Overview and Rationale for GOA Framework Standard

FOREWORD

This SAE Aerospace Information Report (AIR) was developed as a supplement to SAE AS4893 Generic Open Architecture (GOA) Framework standard. This AIR provides an overview and rationale of the GOA Framework. This AIR was prepared under the direction of:

Chuck Roark Chairman, AS-5 Committee
Bosch Telecom, Inc.
P.O. Box 742466
Dallas, TX 75374

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

1. SCOPE:

This document is a companion document to SAE AS4893 “Generic Open Architecture (GOA) Framework Standard” and provides an overview and rationale for SAE AS4893. The GOA Framework establishes an architectural framework to assist in the application of open systems interface standards to the design of specific hardware/software systems. The GOA Framework standard is intended for use by both system designers and system implementers in the development of open systems architectures. It is intended that domain specific guidelines be developed to provide clarification for application of the GOA Framework.

The Generic Open Architecture (GOA) Framework was initially developed by the SAE to provide a framework which could be used to classify interfaces needed in airborne avionics systems. At the time of the development of the GOA Framework, development of such a classification was considered crucial to the application of open systems standards to military avionics. However, it was recognized during the development of the GOA that the GOA Framework might be applicable to domains other than avionics. For that reason the framework is entitled Generic Open Architecture instead of the original name, Generic Open Avionics Architecture (GOAA).

This document is one of several documents contained within the GOA document set. “Introduction to the GOA Family of Document Set”, SAE AIR5314, provides an overview of the GOA Family of Documents. SAE AIR5314, the GOA Framework Standard, and this document make up the domain independent set of documents within the GOA Family of Documents. These domain independent documents are augmented with domain specific sets of document. The Domain specific sets of documents consist of a Catalog of Preferred Standards document, Rationale for Catalog of Preferred Standards Document, and GOA Guidance Document for each domain. The first domain specific set of documents within the GOA Family of Documents being developed is for the aviation domain.

1.1 Purpose:

The purpose of this document is two-fold:

1. provide detailed explanation of the key concepts contained within the GOA Framework standard.
2. provide rationale for key concepts on which the GOA Framework standard is based.

1.2 Intended Audience:

The intended audience for this document is anyone that has an interest in using or understanding the GOA Framework standard. This typically would include system engineers, hardware engineers, software engineers, engineering managers, project managers, academia, and procurement personnel. This document assumes the reader is familiar with the GOA Framework, SAE AS4893.

1.3 Document Structure:

This document is organized as follows:

Section 1:	Scope - provides an introduction and purpose to this document.
Section 2:	References - provides documents referenced within this document.
Section 3:	GOA Framework Description - provides a description of the GOA Framework.
Section 4:	GOA Framework Development Approach - provides an overview of how the GOA Framework was developed.
Section 5:	Examples - provides several examples of the application of the GOA Framework.
Section 6:	Comparison With Other Models - compares the GOA Framework with other well-known reference models.
Section 7:	Compliance - discusses issues of compliance with the GOA Framework.
Appendix A:	Overview of the Space Generic Open Avionics Architecture (SGOAA) – the baseline documents for the GOA Framework.

2. REFERENCES:

2.1 Standards:

ISO	International Organization for Standardization 7498: Information Processing Systems - Open Systems Interconnection - Basic Reference Model, 1984
POSIX91	“Draft Guide to the POSIX Open Systems Environment”, P1003.0/D14, IEEE Computer Society, November 1991
SAE AS4893	“Generic Open Architecture (GOA) Framework”, SAE AS4893, SAE AS5, Jan 1996
SAE AIR5314	“Introduction to the GOA Family of Document Set”, SAE AIR5314, SAE AS5, [TBD]

2.2 Specifications:

2.2.1 Government Specifications:

TAFIM97	“Technical Architecture Framework for Information Management”, Defense Information Systems Agency, Version 3.0, April, 1997.
SGOAA	[SGOAA]Wray, R.B. and Stovall, J.R., "Space Generic Open Avionics Architecture Standard Specification", LESC-30354-C (NASA CR-188290), Lockheed Engineering & Sciences Company, June 1994

2.3 Other Publications:

BOE91	Flanagan, Rich and Van Ausdal, Art, "SATWG Flight Data System Architecture Specification Outline" briefing, 25 October 1991
GD90A	General Dynamics "Space Avionics Requirements Study", 21 October 1990 as briefed to the SATWG
Levine	Levine, L., PCMCIA Primer, M&T Books, 1995
PRU90	Pruett, D., "Avionics Software Open System Environment Reference Model", JSC, March 1990.
WRA91	Wray, R.B., "Requirements Analysis Notebook for the Flight Data Systems Definition in the Real-time Systems Engineering Laboratory (RSEL)," Job Order 60-430 for the JSC, NASA-CR-185698, LESC- 29702, December 1991.
WRA93	Wray, R.B. and Stovall, J.R., “Space Generic Open Avionics Architecture (SGOAA) Reference Model Technical Guide”, Job Order 60-430, Contract NAS9-17900 for the JSC, NASA CR-188246, LESC-30347, April 1993. (Note: some of the figures in this reference have been superseded by the figures in revision C of the standard.)

3. GOA FRAMEWORK DESCRIPTION:

3.1 Purpose of GOA Framework Standard:

The purpose of SAE AS4893 is to provide a framework to identify interface classes for applying open systems to the design of a specific hardware/software system. This framework is used to define an abstract architecture based on a generic set of interfaces. The generic set of system interfaces facilitate identification of critical interfaces.

It is intended that the GOA Framework be specialized for varying domains. A domain specific implementation of the GOA Framework will increase the chance that components/capabilities produced independently will “plug and play” and evolve affordably. The GOA Framework provides a basis for commonality for both vendors and users of components/capabilities. Application of the GOA Framework imposes constraints on implementations that increase the likelihood that independently produced products will interoperate.

Application of the GOA Framework together with the appropriate open system interface standards is expected to provide the following benefits to future programs:

- Provide the basis for establishing a set of specifications, standards and procedures that will become common to all elements of a major system.

- Ensure that future systems can be upgraded and maintained with minimal redesign impact to the existing system by identifying the interfaces required to enable modular replacement of hardware and software.

- Promote availability of multiple sources of needed software and hardware, especially commercial off-the-shelf components.

3.1 (Continued):

- Provide a pool of hardware and software modules for multiple program commonality and re-use.
- Insure access to the architecture and its design documentation for any vendor or agency desiring to propose new uses and applications, and to facilitate competition to contain cost growth.
- Aid in the definition of an open architecture for the system.

3.2 Baseline Document Overview – SGOAA:

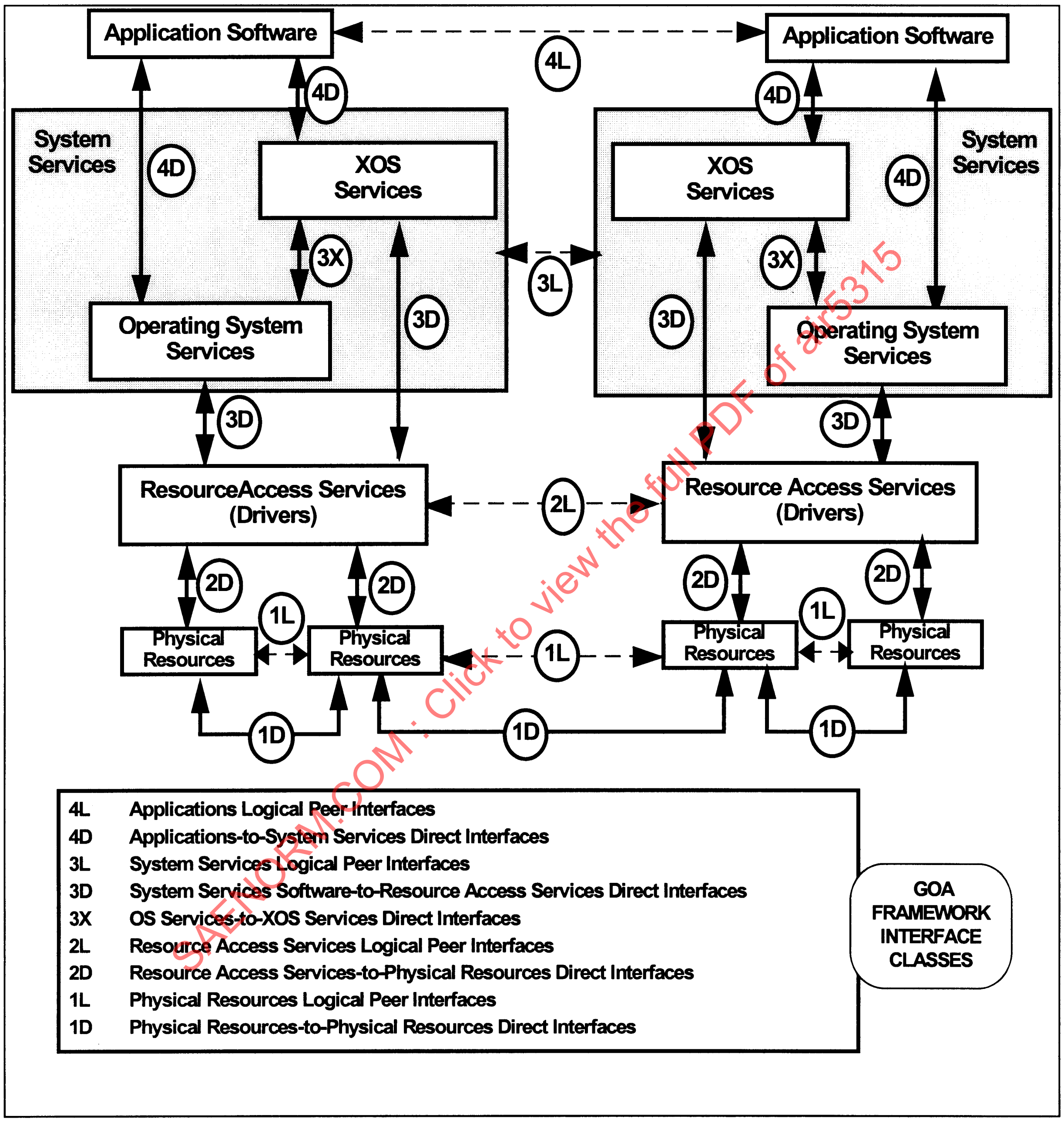
The Space Generic Open Avionics Architecture (SGOAA) was the base document for the GOA Framework. Appendix A provides a synopsis and rationale for the SGOAA.

3.3 Basic Concepts and Perspective:

This section presents basic concepts and perspectives on the GOA Framework. The GOA Framework is depicted in Figure 1 for reference. A definition of the GOA Framework can be found in SAE AS4897. The overriding reason for the definition of the GOA Framework was to provide an architectural framework to aid in the identification of critical components and the interfaces between these critical components. The identification of such interfaces supports reuse and seamless upgrades that address obsolescence, the need for increased functionality, and leveraging emerging technology.

A system consists of a topology of nodes (or modules), some of which may be used for processing, i.e., containing one or more processors. A node is a self-contained assembly of electronic components and circuitry. A typical avionic system is comprised of multiple nodes, each individually addressable via a backplane bus. An instantiation of the framework associated with a particular processing node is referred to as a “GOA stack”. As will be discussed later, there are views of the system in which a “full” GOA stack view is not appropriate - what is appropriate are the 1L-4L logical peer-to-peer (same level) interfaces and the 1D direct physical interface between GOA stacks. This latter view is a type of “black box” view of a system while a full GOA stack view is appropriate when viewing a portion of the system that contains a processor together with the software that executes on the processor.

The GOA Framework is defined by nine interface classes and four GOA layers. One of the GOA layers, the Systems Services Layer, contains two types of sublayers - Operating System and eXtended Operating System (XOS). The following provides an overview and reasons for the chosen classes and layering.



4L	Applications Logical Peer Interfaces
4D	Applications-to-System Services Direct Interfaces
3L	System Services Logical Peer Interfaces
3D	System Services Software-to-Resource Access Services Direct Interfaces
3X	OS Services-to-XOS Services Direct Interfaces
2L	Resource Access Services Logical Peer Interfaces
2D	Resource Access Services-to-Physical Resources Direct Interfaces
1L	Physical Resources Logical Peer Interfaces
1D	Physical Resources-to-Physical Resources Direct Interfaces

FIGURE 1 - GOA Framework

3.3 (Continued):

The GOA Framework interfaces are based on two types of interfaces: logical and direct interfaces. A logical interface defines requirements for peer-to-peer interchange of data. A logical interface identifies senders and receivers, the type of data to be exchanged, when the data is to be exchanged, and formats if appropriate. A direct interface identifies the characteristics of the information transfer medium/facility. No information transfer actually takes place as part of a logical interface; instead, information transfer takes place as part of one or more direct interfaces. Simply stated, logical interfaces define what information is transferred, and direct interfaces define how the information is transferred.

One example of a logical interface is the definition of interface requirements for exchange of latitude and longitude information from a Navigation application to a Stores Management application. The logical interface defines information that would become the basis for an Interface Control Document (ICD) for Navigation to Stores Management communication. Another example of a logical interface is the message header used by a Communications XOS service to encapsulate message data. This message header contains information that is interpreted by the peer Communications XOS service entities in order to make a message transfer occur. For example, logical designations of the message destination(s) and type might appear in such a header. The definition of the message header is a logical interface that tells the two peer Communications XOS services how to initialize and interpret the message header when a message transfer actually takes place.

Direct interfaces specify how components in one layer invoke services in the layer immediately below it within the same GOA stack or how the physical layer in one GOA stack "physically" interfaces with the physical layer of another GOA stack (e.g. between processor modules). Logical interfaces occur as the result of invocations of direct interfaces that support communications. For example, consider the latitude and longitude logical interface example presented in the previous paragraph. If message-passing communication is used, the actual transfer of the latitude and longitude information would occur as the result of the Navigation application invoking a “send” operation in the System Services layer. The invocation of the send routine would cause the system to physically transfer the data to make it available to the Stores Management application. The Stores Management application would then invoke a “receive” System Services operation to actually get the message containing latitude and longitude information. These send and receive routines are example of direct interfaces. However, there are also direct interfaces that have nothing to do with communications - these direct interfaces just invoke a service such as reading the time, allocating or deallocating memory, etc. The interface specification of a component within a GOA layer designates the direct interfaces to that GOA layer. For example, POSIX defines a standard Operating System component interface; bus standards define the electrical characteristics of the bus.

Thus, logical interfaces are considered important to define communication and synchronization requirements between peer-to-peer components. Direct interfaces are important for defining the interface to services (including communication services that make logical interfaces possible.)

Before one can define interfaces, one must know what is being interfaced. The following provides an overview of these components - the GOA layers and some rationale for choosing the GOA layers.

3.3 (Continued):

The Application Software layer contains the platform level application specific code (or function). The application layer within a specific GOA stack can contain multiple software application components. The requirements for communication between different application components (whether within the same or different GOA stacks) is a logical interface. As noted above, the communication occurs through direct interfaces. For message passing paradigms, the communication occurs by invoking message-passing services. If the interface is through shared memory (instead of message passing), the direct interface is simply a load or store operation, and the logical interface is the association of the address and size of the data. The 4D interfaces, i.e., Application Program Interfaces (APIs), are critical since they are important from a reuse perspective and a seamless upgradeability point-of-view. APIs support reuse by having the same Systems Services implemented the same way on different hardware platforms. Seamless upgrades can be supported by 4D interfaces since they abstract the Applications Software from the lower GOA layers. In particular, the Systems Service layer abstracts the Applications Software from changes in the Physical Resources layer, such as processor modifications/upgrades.

The Physical Resources layer is the bottom-most layer and provides the interface between different physical components. The Physical Resources layer within a specific GOA stack can contain multiple Physical Resource components. The following are examples of Physical Resource components that might appear in the same GOA layer: Pi-Bus interface unit for intermodule data communication, TM-Bus interface unit for intermodule test and maintenance, IEEE-488 interface unit for debug, and CPU device for program execution. The 1D direct and 1L logical interfaces are clearly critical interfaces from a black box point-of-view. These interfaces include the physical and data link level definitions of busses where the electrical/mechanical requirements are direct (1D) interfaces and how the bits are interpreted are logical (1L) interfaces.

The System Services Layer is provided to encapsulate common services accessed by the Application Software Layer. The Operating System (OS) Service is the minimal component within the System Services Layer. XOSServices designate other components contained within the Systems Services Layer besides the OS. This layer is important since it allows an abstract view of common services required by applications. As discussed in section 4.2 Issues and Resolution, the 3X interface provides an interface between XOS and the OS services to provide either capabilities not included in the OS 4D interface (i.e., privileged operations) or optimized 4D interface capabilities. The optimization is for performance and could occur through an interface that has reduced checking or has less call overhead (such as avoiding a system trap). The logical 3L interface provides System Service Layer peer-to-peer interface communications interfaces. For example, the interpretation of the fields within the communications header added to a message between two applications is an example of a 3L interface. It should be noted that, even though not explicitly mentioned in SAE AS4893, an XOS Service may use 4D interfaces to invoke the Operating System or other XOS Services resident within the same GOA stack.

3.3 (Continued):

The Resource Access Layer contains components that directly access the hardware. This layer includes classical device drivers and memory mapped I/O definition translation, for example. The GOA Framework supports the idea that this layer abstracts the remaining layers from the Physical Resources. The 3D interface is provided between the System Services Layer and the Resource Access Layer to allow the non-processor dependent (i.e., non-I/O dependent) algorithms within the Systems Services Layer to be abstracted from specific Physical Resource Layer implementations. This supports OS and XOS portability between target platforms. The Joint Integrated Avionics Working Group (JIAWG) Input/Output Built-In-Test Description Specification (IOBIDS) specification and the Uniform Device Driver Interface (UDI) interface are example of 3D interfaces. The 2D interface defines the direct interface between the Resource Access Services Layer and the Physical Resource Layer. This is typically the direct interface between software and hardware. Memory mapped I/O definitions and processor instruction-set architecture (ISA) definitions are two examples of a 2D interface. The 2L interfaces are Resource Access Services peer-to-peer interfaces. An example of a 2L interface is the interpretation of error information returned by the Resource Access Service layer in the destination GOA stack for a data communication activity between two stacks.

The GOA Framework’s inherent flexibility allows a system to be viewed from several different perspectives:

- Application point-of-view
- Systems software point-of-view
- "Classical" hardware point-of-view.

Except for 1D direct physical interfaces, the direct interfaces are between contiguous GOA layers within the same GOA stack residing on the same hardware module. Logical interfaces can be between peer GOA layers residing:

- within the same hardware module
- in different hardware modules within the same backplane
- in different hardware modules within different backplanes within the same system
- in different hardware modules within two subsystems, e.g., Mission Computer and Vehicle Management System (VMS)
- between different hardware modules in different systems (e.g., aircraft and a ground based system).

As shown in Figure 2, only the logical interfaces and 1D direct interface are of interest when considering interfaces between GOA layers in different hardware modules. In other words, if one has two entities considered as black boxes (i.e., the internals of each entity are not visible to the other), only the logical interfaces and the direct physical interface are visible. However, when one looks at a complete GOA stack executing on a hardware module, all the GOA layers are visible and hence the direct interfaces between adjoining layers are of interest.

3.3 (Continued):

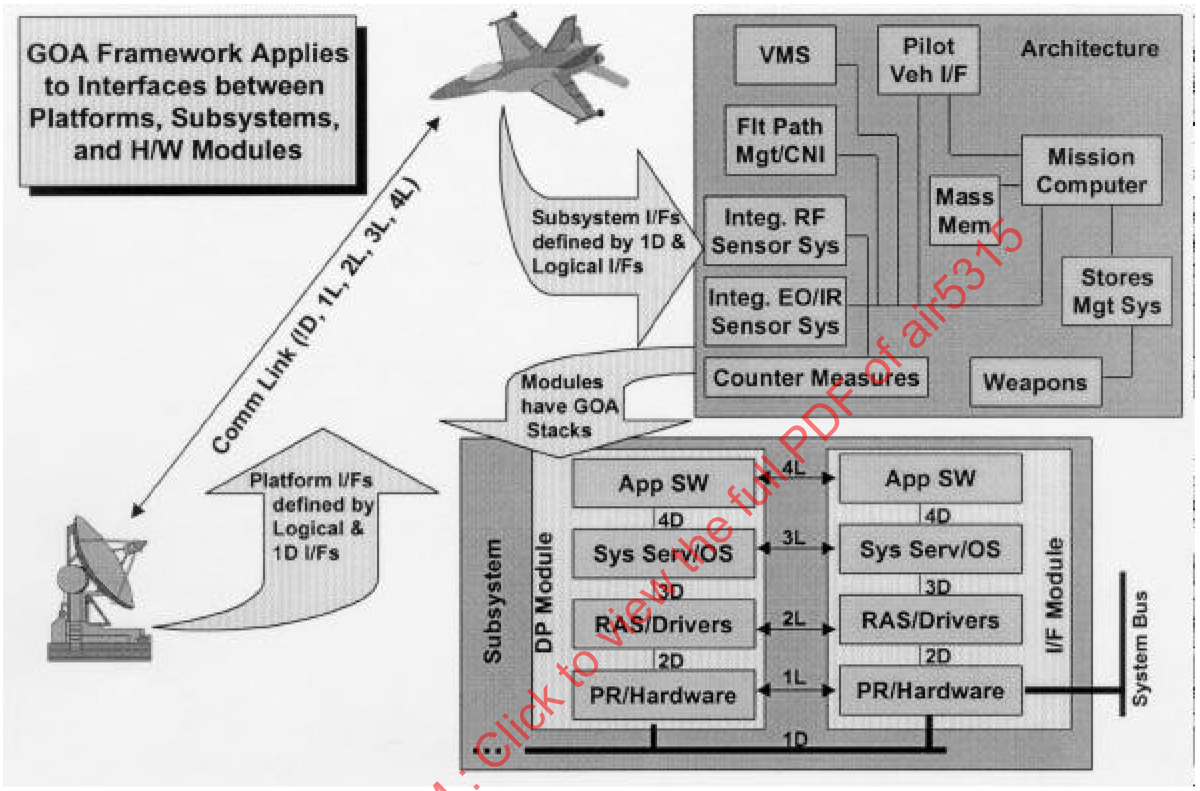


FIGURE 2 - Different Ways of Viewing GOA Framework

The following is a list of basic principles in applying the GOA Framework.

a. Within the GOA Framework only the Resource Access Services should have knowledge of Physical Resource (2D) interfaces. Therefore, software in the system services layer is generally allowed to access the physical hardware only by invoking 3D interfaces provided by the RAS layer. Application software must call the appropriate 4D interfaces to request that system services invoke RAS interfaces to access physical resources. This allows abstraction of software in higher levels of the GOA Framework from physical resources.

b. An XOS service is either a privileged application (i.e., allowed to invoke 3X interfaces) or code that is used by multiple applications, i.e., application services. Note that XOS service code may have originally been developed as application code but it was later noted that it became commonly used (i.e., was viewed as a service) by multiple applications and thus became an XOS service. Application code cannot invoke the OS via 3X interfaces.

3.3 (Continued):

c. An application may be transformed to an XOS component in System Services as follows:

1. Within the GOA Framework, application services that are not part of the System Services GOA layer are considered part of an application; i.e., invocation is considered an application-to-application 4L interface. If such libraries evolve to be shared between applications, then these libraries can become XOS services and their internal call interfaces must be converted from a 4L interface to an explicit 4D interface.

2. An application's communication with other applications is a 4L interface which is implemented through a 4D interface. For communication, this means a send/receive interface or remote procedure call interface can be used, for example. The parameters to the services (e.g., message data) are specific to a particular 4L interface. If an application subsequently becomes an XOS service, the 4L interfaces to the application are converted to procedural (4D) interfaces instead of using the 4L interface and exchanging data via the 4D OS interface. This is done for optimization and to make calls to the service explicit in application code, instead of being hidden in parameters to 4D communication interfaces. If this is not done, then even though the code is being used as a service by multiple applications, it is still an application and not considered a System Service in the GOA Framework. Note this means that applications that previously communicated with the application that became an XOS service must be modified to use the newly created 4D interface to the service.

d. There are no 4D interfaces between applications. The internal application design/structure is invisible in the GOA Framework; i.e., the GOA Framework does not address direct interface between applications. The 4D interface is by definition an interface between Application Software and System Services. It is possible an application may be constructed as a collection of component applications layered on top of one another; i.e., an hierarchical layering. The communications between the layers are considered 4L interfaces in the GOA Framework. This makes sense if one remembers that 4D interfaces are used to implement the 4L interfaces.

e. Direct ISA interfaces fit within the GOA Framework in the following manner:

1. Processor ISA interfaces are 2D interfaces in the GOA Framework since an ISA interface is a software to physical resource interface. An ISA can be considered to define I/O related instructions and non-I/O instructions (i.e., CPU internal instructions). The I/O related ISA instructions define the interface between a processor and the driver portion of the Resource Access Services. The non-I/O instructions are invoked due to execution of application code written generally in a Higher Order Language (HOL) (or possibly assembly language). The HOL (or assembly language) defines a 4D interface. But since the compiled (assembled) code executes directly on the processor, there is no System Service or Resource Access Service support. Therefore, the 4D, 3D, and 2D interfaces are null at run-time because their execution really occurred during compile time (see below). In essence the 4D interface has been compiled (or assembled) directly to a 2D interface.

3.3 (Continued):

2. From a toolset point of view, there is an alternate view of the application of the GOA Framework to non-I/O ISA instruction interfaces. Higher Order Languages abstract the ISA from application code and the abstraction is implemented via a compiler. The non-I/O related ISA characteristics are what the HOL abstracts from typical application code. The compiler can be considered as consisting of a front end which does parsing and semantic checking and a back end which does code generation. Under these assumptions one can think of the front end as an XOS service (since it does not care about the particular target) and the back end as a Resource Access Service (since it cares about the specific ISA). Thus, the HOL is a 4D interface to the compiler front end, which has a 3D interface to the compiler back end, which has a 2D interface to the ISA. (Note that an assembler is analogous to a compiler back end in this discussion, where the assembly language is a 3D interface.) These direct interfaces occur at compile time in which source code is translated to object code. The 2D interface is executed at run-time on the target processor.

f. Shared memory interfaces fit within the GOA Framework as follows:

1. The communication between application software is via assignment statements (i.e., via load/store operations). The direct interface would be a 2D interface implemented as a load/store non-I/O instruction as discussed above. There is an implied 4L interface that specifies the address of the shared data.

2. There may be 3X or 4D interfaces in support of systems that support caching. For example, such interfaces might include operations to flush or invalidate a cache. The 3X interface option is recommended for most systems, since such cache control operations can be unsafe for direct application control.

g. Logical interfaces are exclusively peer-to-peer relationships.

1. The GOA Framework logical interfaces are peer-to-peer, analogous to the peer-to-peer interfaces defined in the ISO OSI 7-layer reference model. Since logical interfaces are peer-to-peer, a constraint is placed on defining the GOA layers from an end-to-end and recursive point-of-view. Specifically:

(a) From an end-to-end point-of-view, applications on each end communicate only via 4L messages. System Service capabilities on each end communicate only via 3L messages. Resource Access Services on each end communicate only via 2L messages. Physical Resources on each end communicate only via 1L messages. There is no logical communication between capabilities in different GOA layers.

(b) From a recursive point-of-view, capabilities within the same layer may have logical communication with one another. For example, a System Service may communicate logically with a System Service in a recursive layer but a System Service may not communicate logically with an application in a recursive layer.

3.3 (Continued):

2. The peer-to-peer relationship constraint must be considered when assigning a capability to a particular GOA layer. The reason for this is the peer-to-peer relationship must hold not only from an end-to-end perspective but also when the recursive application of the GOA Framework is applied. For example as depicted in Figure 3, suppose a data processing module contains a full GOA stack and wants to communicate with an application resident on a module in another backplane. Also, assume that a store and forward device is used as a gateway between the two backplanes. On the gateway assume there is a four level stack consisting of the physical resource backplane bus and backplane to backplane interconnect, Resource Access Services consisting of drivers for the bus and inter-backplane interconnect, microkernel OS for the gateway module, and software that controls the store-and-forward process. To the gateway module, the control software could be considered application software. However, it must be considered XOS software within the defined system as a whole since this code will have a logical interface with XOS network OS type code in the data processing module. If the control software was considered application software, the peer-to-peer relationship would be broken.

3. There are actually two ways recursion in the GOA Framework have been discussed. (See Figure 4.) The first is the type of recursion described in the previous example in which data transfer occurs through various GOA stacks prior to arriving at the final destination GOA stack. This is referred to as linear recursion. The second type of recursion that has been discussed is recursion within a GOA layer. This means a layer can be expanded into a GOA stack. If this latter view is taken, care must be observed in viewing peer-to-peer logical interfaces. Linear recursion through the system should follow the non-recursed peer-to-peer layer relationships.

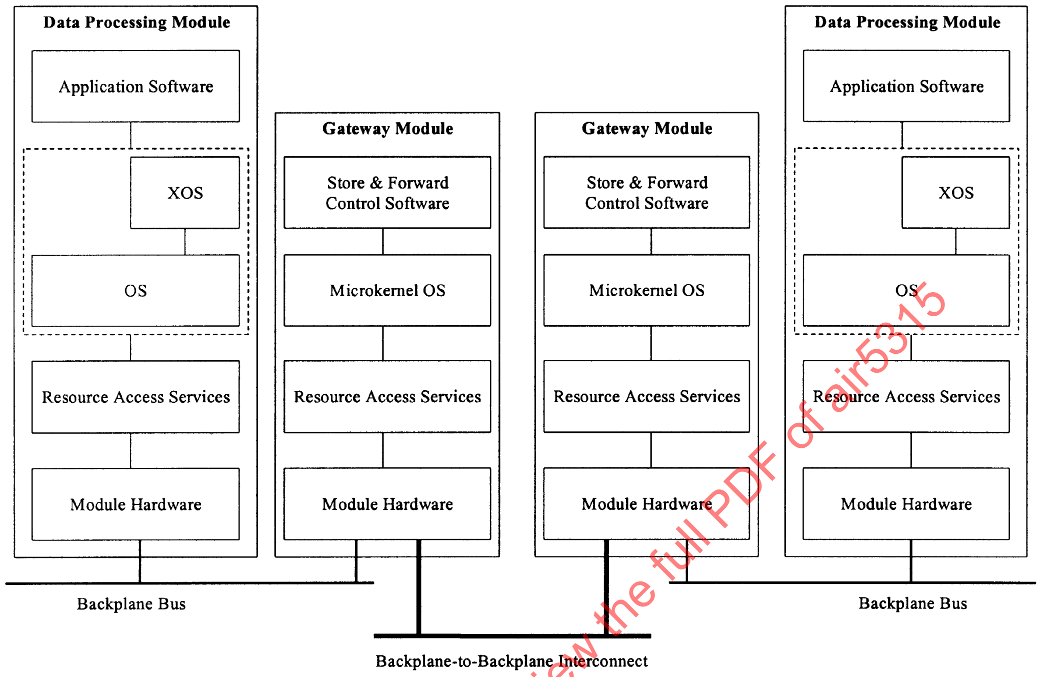


FIGURE 3 - Example Inter-Backplane Communication

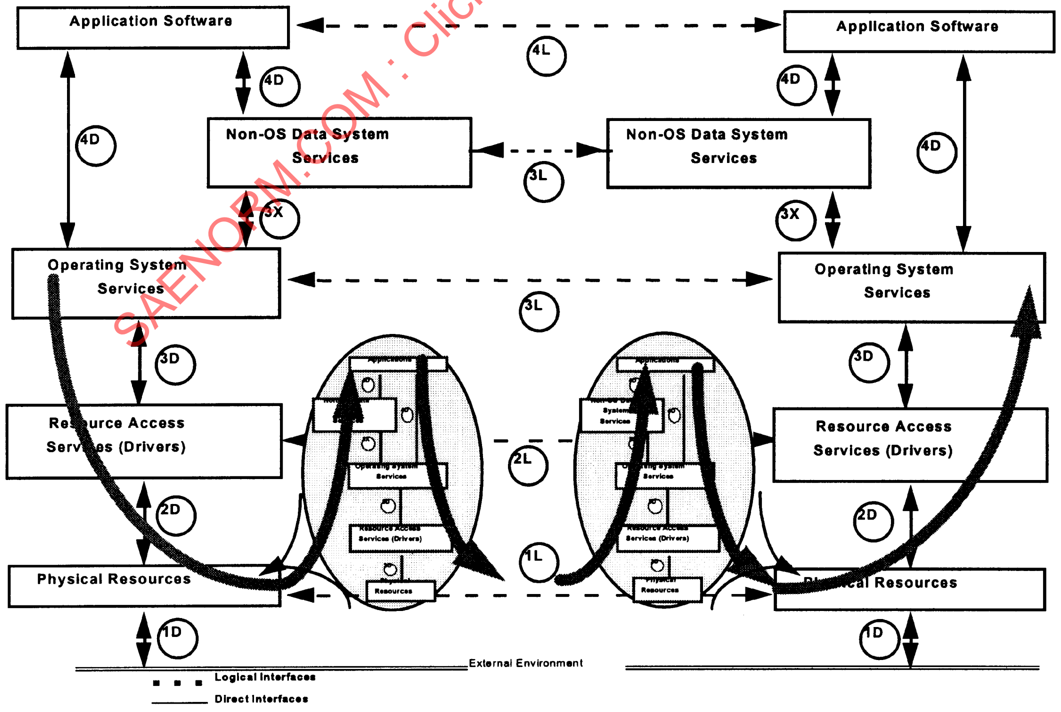

Application Software
Application Software
4L
4D
4D
4D
4D
Non-OS Data System Services
Non-OS Data System Services
3L
3X
3X
Operating System Services
Operating System Services
3L
3D
3D
Resource Access Services (Drivers)
Resource Access Services (Drivers)
2L
2D
2D
1L
Physical Resources
Physical Resources
1D
1D
External Environment
Logical Interfaces
Direct Interfaces

FIGURE 4 - Recursion Using the GOA Framework

4. GOA FRAMEWORK DEVELOPMENT APPROACH:

4.1 Evolution from SGOAA:

The development of the GOA Framework began with the authors of the SGOAA (Richard Wray and John Stoval) requesting that the SAE AS-5 Subcommittee publish a standard based on the SGOAA. Both Mr. Wray and Mr. Stoval presented to the SAE several times to help bring the SAE up-to-speed on the SGOAA. As noted in Section 3.2, the SGOAA is a massive document set of which the SGOAA framework is a very small part. During the discussion it was realized that the framework could be used as the basis for identification of interface types that should be included in a preferred set of standards document for a domain. At the time of the GOA Framework development (as is still true), domain specific preferred sets of standards documents were considered key in promoting reuse and establishing open systems architectures that include interfaces that were as wide-spread as possible for the domain. It was also determined during these discussions that there were widely varying views on the systems engineering concepts being promoted within the SGOAA document. Because of this, it was felt that to develop a standard based on information beyond the basic framework would be very time consuming. Since one government customer, the Avionics Engineering SubBoard (AESB) within the Joint Commanders Aeronautical Group, encouraged standardization of the GOA Framework in order to accelerate development of a preferred set of standards for the avionics domain, the SAE AS-5 concentrated its efforts on evolving the SGOAA framework to what is now the GOA Framework.

4.2 Issues and Resolution:

The following provides a summary of the major issues and the corresponding resolutions that occurred as part of the GOA Framework standard development.

- The original SGOAA framework definition included the following names for what became the GOA layers: Hardware Layer, Driver Layer, Data System Service Layer, Operating System Services Layer, and Application Software Layer. It was realized that different layers of the GOA Framework could be implemented in hardware or software. Therefore, names of the two lower levels were changed from “Hardware Layer” to “Physical Layer” and “Driver Layer” to “Physical Resource Access Layer” to recognize that the lower layers could be implemented in either hardware or software. The Application Software layer remained unchanged since a majority of the group believed that the upper layer of a GOA stack can always be assumed to be software. Part of the rationale is that firmware implementations were considered software. The Data System Services Layer and Operating Systems Services Layer were grouped together into a single GOA layer - the Systems Service Layer since both the OS and Data System Services were examples of service components called by application software. Data System Services was changed to eXtended Operating System (XOS) Services since there were more non-OS services than the name implied by Data System Services. OS Services was continued as a sublayer since OS is considered an important component that manages basic processor resources such as memory and processor execution.

4.2 (Continued):

- The original SGOAA model included six interface classes instead of nine classes and the numbering was also done differently. The initial SGOAA interfaces consisted of class 6 (corresponds to GOA class 4L), class 5 (corresponds to GOA class 4D interfaces to XOS Services), class 4 (corresponds to GOA class 3L), class 3 (corresponds to GOA class 4D interfaces to OS and GOA class 3D), class 2 (corresponds to GOA class 2L), class 1 (corresponds to GOA class 1L and 1D combined). Additionally, the SGOAA had the following layers: Applications, Data System Services, Operating System Services, Hardware Extensions/Drivers, and Hardware. The GOA group caused three changes in the original SGOAA framework. First, realizing the basic SGOAA model is based on logical and direct interfaces between the SGOAA layers, a full set of direct and logical interfaces was defined between all layers. Additionally as discussed below, a 3X interface was added internal to the Systems Services Layer. Second, the interfaces were relabeled. The relabeling occurred in two ways. First, except for the 3X interface, a label was appended with D if it was a direct interface and with L if it was logical interface to make it more obvious the type of an interface. Second, for ease of understanding, the same cardinal number was associated with the logical interface between peer components and the direct interface from one GOA layer to the layer immediately below. (The Application Software layer is the highest layer and the Physical Resources layer is the lowest. Even though Physical Resources are at the lowest level in the model, direct interfaces between physical resources (1D) are considered analogous to components in the Physical Resources layer interfacing with a lower layer.)

- The baseline definition of the framework did not include the 3X interface initially. The 3X interface was added to allow more than just 4D interfaces for access to Operating System services. Experience has shown that an eXpress interface is needed between eXtended Operating System Services (XOS) and the OS for several reasons.

  - Some OS service interfaces support functionality that should not be made available to all application software but may be needed to support XOS services. Such interfaces are considered "privileged" interfaces. In general, applications are not considered to have the appropriate privilege to execute these services. For example, an XOS service such as a Communications Service might need to effect task context switch(es) based on sending or receiving of messages. In practice an optimized suspend and resume interface capability has been used in such instances, instead of a more costly interface such as the Ada rendezvous. In some systems developed in Ada, availability of these low-level interfaces was deemed inappropriate for non-privileged application; i.e., suspend and resume were deemed inappropriate as 4D interfaces.

  - In some cases, an XOS service may have originally been developed as an applications program and later, based on its services being needed by multiple applications, it became an XOS service. When it became an XOS service, it may have been rehosted to kernel space instead of application in order to allow more efficient execution. If an XOS is moved from user space to kernel space, it may need an optimized 4D interface that is applicable to software that is resident in kernel space.

4.2 (Continued):

- The SGOAA was aimed at space vehicle avionics. The SAE's initial interest was in evolving the SGOAA to be applicable to avionics in general - not just Space related avionics. It was clear that the framework had nothing that restricted it to Space applications. The name for the SAE Framework at this time was Generic Open Avionics Architecture (GOAA) Framework. It further became apparent that the framework was generic in nature and not specific to avionics applications. Also realizing that acceptance of the framework would be enhanced if its name did not tie it just to avionics, avionics was dropped from the name and the SAE framework became known as the Generic Open Architecture (GOA) Framework. It should be noted that there was much heated debate over this topic since many in the SAE did not want to dilute the focus of the SAE Avionics System Division (ASD) relating to avionics. In addition to the realization that there would be better overall acceptance of the framework if the name were changed (for example, the AESB wanted this name change since the Open Systems Joint Task Force desired it), the avionics focus concern was resolved when the concept of a GOA family of documents arose. The family of documents, as noted in Section Purpose, contains domain specific documents for application of the GOA Framework, where avionics is one of the domains.

- A major question with the application of the GOA Framework was whether components in one GOA layer should be allowed to directly access services in non-contiguous GOA layers, without using intermediate direct interfaces. If this situation were allowed, should a waiver be required to claim GOA Framework compliance? The general rule that developers of the GOA Framework want to stress is that direct interfacing between non-contiguous GOA layers should not be practiced since it nullifies the abstraction of the higher layers from the lower layers, thus reducing the reuse and seamless evolution attributes of using such a framework. However, even though not explicitly shown in the GOA Framework, the GOA Framework does not preclude one GOA layer from invoking a direct interface in a non-contiguous GOA layers since it was realized that this may be necessary in some cases for meeting real-time requirements. In such cases, the 4D and 3D interfaces (as appropriate) are considered null interfaces. If such bypassing of interfaces is done, a conscious decision for doing this should be made and it should be documented and obvious to reviewers of an implementation.

- The original SGOAA model did not make it clear whether XOS services could interface directly with the RAS or had to go through the OS to interface with the RAS. Realizing that services such as Communications Services in practice directly talk with the RAS without going through the OS, the GOA Framework explicitly shows that the RAS may have 3D interfaces from either the OS or XOS within the Systems Services layer.

- Much of the time developing the GOA Framework standard dealt with agreement on the definitions within the standard. The baseline for the definitions were from the SGOAA documentation. These definitions were generalized to be non-avionics specific and in some cases were modified to be more explicit in the eyes of the GOA Framework development team.

4.3 Types of Companies and Agencies Involved:

The GOA Framework development team consisted of a diverse set of companies, agencies, and academia that met as part of the SAE AS-5 that provided an industry forum for documenting the GOA Framework as an industry standard. The types of agencies involved in development of the GOA Framework included representatives from the U.K. Ministry of Defence, French and German military establishments, NASA, U.S. Air Force, U.S. Army, and U.S. Navy. The types of industries represented include: aircraft primes, system integrators, subsystem vendors, military computer vendors, large electronic houses that support military developments, primes and electronics suppliers of space products, military consulting industry, and some commercial industry representatives.

5. EXAMPLES:

Two examples of application of the GOA Framework are shown in the appendices of SAE AS4893. Two more examples are presented in this section.

5.1 Pi-Bus Mapping to GOA Framework:

The Pi-Bus is defined in SAE AS4710. The Pi-Bus defines a parallel backplane communications bus. The Pi-Bus has two major sections defining the bus protocol: Section 3 defines the physical layer protocol and Section 4 defines the data link layer protocol. Refer to both SAE AS4710 and its companion handbook SAE AIR4903 for a definition and discussion of the Pi-Bus. Section numbers listed in the remainder of this paragraph refer to section numbers in SAE AS4710.

The physical layer definition is defined in two major subsections: Section 3.2 provides the line definitions and Section 3.3 provides the electrical definition. Section 3.2 defines part of the 1D interface and 1L interface. The identification of the number of lines and defining the lines to be wired-or specify part of the 1D interface, while the logical meaning provided to the lines and protocols of data placed on the line, such as when even parity is required, specify part of the 1L interface. Section 3.3 also specifies part of the Pi-Bus 1D interface since it defines electrical characteristics.

Section 4 of SAE AS4710 defines the data link protocol for the Pi-Bus and specifies part of the 1L interface for the Pi-Bus as well as requirements for the 2D interface and 2L interface. The data link protocol defines the meaning of bits sent across the Pi-Bus during its operations – this is part of the 1L interface. The definition of the Pi-Bus headers also provide requirements for the 2D and 2L interfaces since these headers are understood by Pi-Bus drivers. These definitions are used to define the software interface to Pi-Bus Controllers. Pi-Bus Controllers provide a mechanism for software to invoke Pi-Bus message transfers, receive Pi-Bus messages, and interpret status such as acknowledgement of a message or errors. The interpretation of fields within a Pi-Bus message header that are seen by the drivers in either the originating or destination(s) of a Pi-Bus message is a 2L interface.

5.2 GOA Stack Definition and Interface Control Documents (ICDs):

This section exemplifies how a GOA stack might be defined and how it relates to architecture interface specifications, i.e., Interface Control Documents (ICDs). Refer to Figure 5 below for this discussion.

5.2.1 Application Interfaces: Application software is shown in the top layer of Figure 4. In this example, the Navigation application resides on one module and the Stores Management application resides on another. The Operator Interface application is distributed between two modules. Two 4L interfaces are shown - a 4L interface between Navigation and Stores Management and a 4L interface between the two partitions of the Operator Interface application.

The external interface between Navigation and Stores Management is documented in an ICD that describes the functional information content (data parameters and units of measure) and the format of messages transferred between the two. For example, suppose Navigation passes aircraft position data (say, latitude, longitude, and altitude) to Stores Management once every 50 millisecond frame. The Navigation/Stores Management ICD describes the format of a message containing this data; e.g., latitude in degrees in word 0, longitude in degrees in word 1, and altitude in feet in word 2. The ICD also specifies the data type for each value (e.g., integer, IEEE floating point, or some packed representation) and frequency of the communication (i.e., 20 Hz). At the appropriate time, the Navigation application creates a message that conforms to the ICD specification and calls a 4D interface to the System Services layer to cause the message to be transferred to Stores Management. The System Services layer invokes the appropriate driver interface (3D), the Pi-bus driver in this example. The Pi-bus driver formats the message to conform to the hardware interface (2D) described in the Pi-bus ICD and performs operations to effect transfer of the message to the module on which Stores Management resides. At the destination module, the same process occurs in reverse. The Resource Access Services layer receives the message and informs the System Services layer that a new message is available. The System Services layer then makes the message available to the Stores Management application.

The Navigation/Stores Management ICD does not contain information about any of the direct interfaces necessary to cause the information to be transferred from Navigation to Stores Management. Furthermore, components in the System Services, Resource Access, and Physical Resources layers have no knowledge of the Navigation/Stores Management ICD and so attach no meaning to the sequence of bits that contain latitude, longitude, and altitude.

Similarly, the interface between the two partitions of the Operator Interface application is specified in an ICD. Transfer of data between them would be identical to transfer of data between Navigation and Stores Management, even though the two partitions are part of the same application.

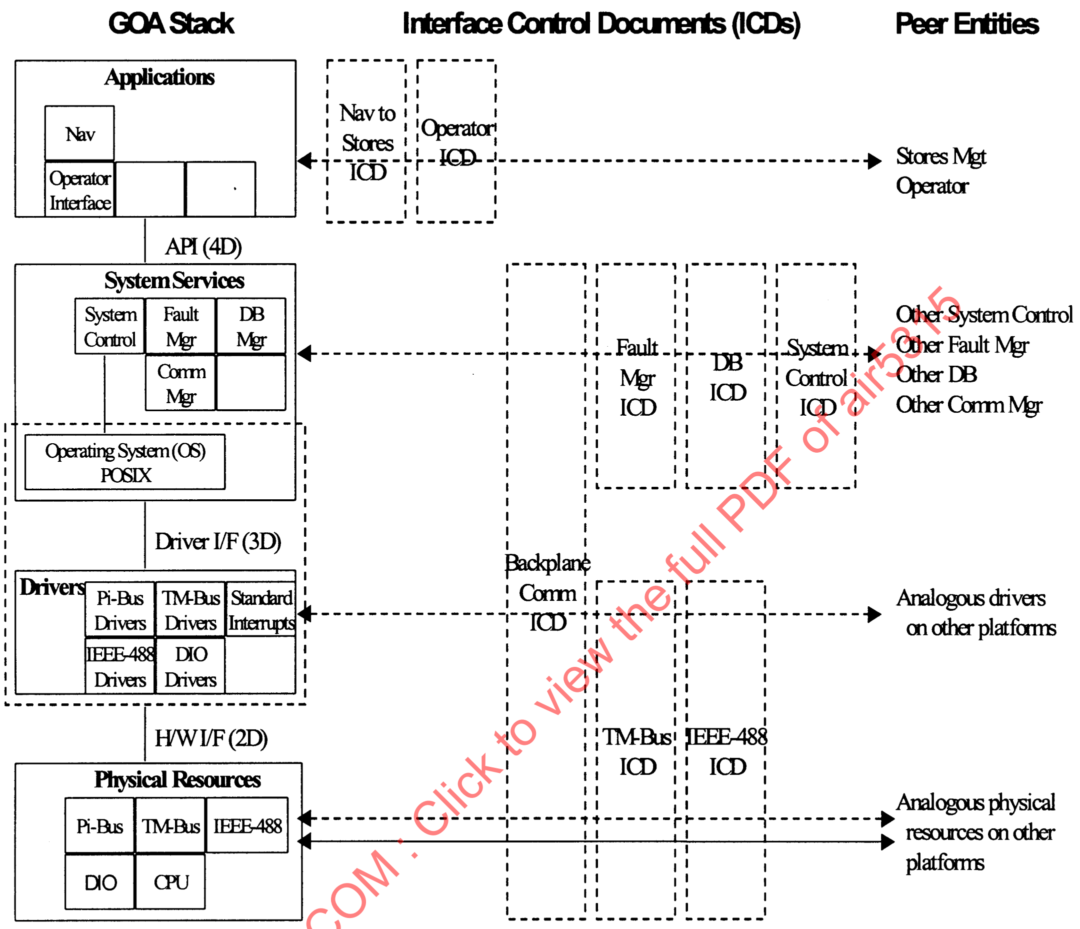


FIGURE 5 - Example GOA Stack and ICDs

5.2.2 System Services Interfaces: Figure 5 illustrates interfaces between components in the System Services layer. Five System Services components are shown:

- System Control
- Fault Management
- Database Management
- Communication Management
- Operating System

The first four are XOS services. ICDs exist to define the interfaces between each of these components and their counterparts on other modules. These ICDs are similar to the application ICDs discussed above. The System Control ICD might describe the format of messages containing system configuration information, such as the assignment of application processes to hardware modules. The fault management ICD might describe the format of heartbeat messages periodically transferred among the Fault Management services resident on each module. As mentioned in Section 3.3, the Communication Management ICD describes the format of communication headers concatenated to messages.

The Database Management ICD might describe the format of messages transferred among components of a distributed database. For example, suppose one module hosts a database that application software on other modules must access. Even though the physical database resides on only one module, each module hosts a part of the database access software that allows applications on that module to access the remote database. This Database Management Service sends a message to the remote database on behalf of the requesting application. If necessary, the Database Management Service on that module formats a reply to the requestor, which the Database Management Service on the original module returns to the application. The format of these messages between the distributed portions of the Database Management Service are documented in the ICD. Thus, application software is not aware of the lower-level details of requesting database operations on a remote module. Note that the 4D interface that allows the application to invoke the Database Management Service is not described in this ICD, but in a document describing the API for Database Management.

5.2.3 Resource Access Services and Physical Resources Interfaces: As described in Section 5.1, the Pi-bus ICD defines the 2D, 2L, 1D, and 1L interfaces between the Pi-bus drivers and physical components. TM-Bus and IEEE-488 ICDs contain analogous information for the TM-Bus and IEEE-488, respectively.

6. COMPARISON TO OTHER MODELS:

The Portable Operating System Interface Standard (POSIX) and ISO Open System Interconnect (OSI) reference models are two popular reference models. The Personal Computer Memory Card International Association (PCMCIA) is a popular open systems architecture model that supports PC card interchangeability. This section shows how the GOA Framework, ISO OSI reference model, and POSIX model complement one another and shows how the PCMCIA model maps onto the GOA Framework. Figure 6 illustrates the relationship between the GOA Framework, ISO OSI reference model, and POSIX model.

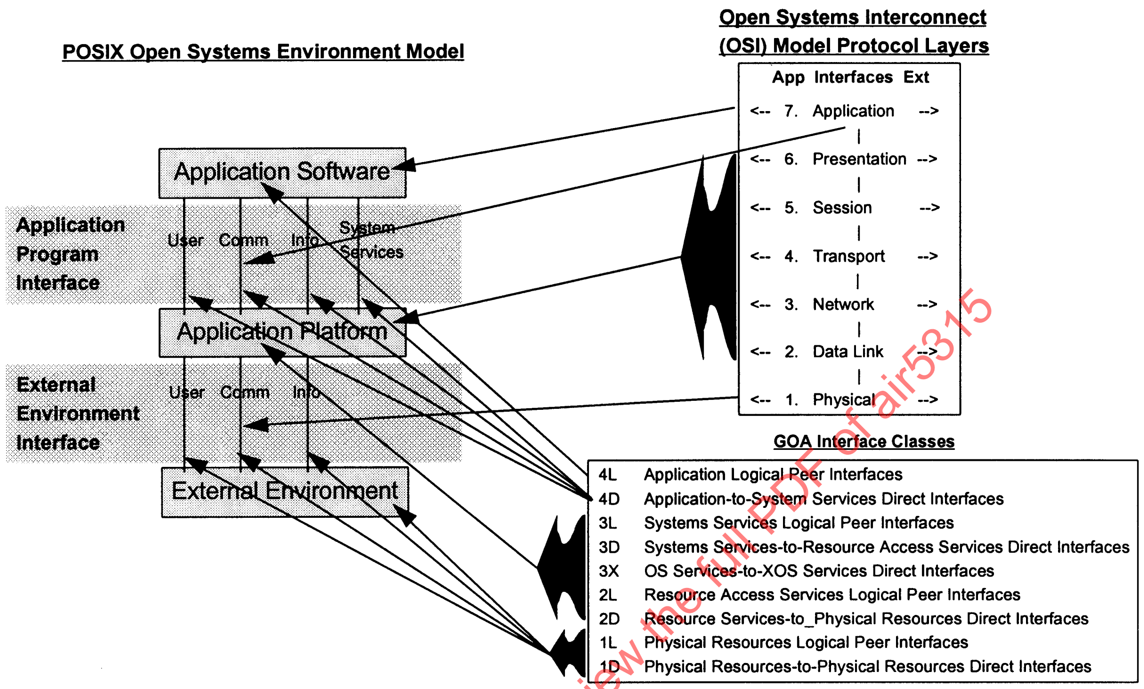

POSIX Open Systems Environment Model
Open Systems Interconnect (OSI) Model Protocol Layers
App Interfaces Ext
<-- 7. Application -->
<-- 6. Presentation -->
<-- 5. Session -->
<-- 4. Transport -->
<-- 3. Network -->
<-- 2. Data Link -->
<-- 1. Physical -->
Application Software
Application Program Interface
User Comm Info System Services
Application Platform
External Environment Interface
User Comm Info
External Environment
GOA Interface Classes
4L Application Logical Peer Interfaces
4D Application-to-System Services Direct Interfaces
3L Systems Services Logical Peer Interfaces
3D Systems Services-to-Resource Access Services Direct Interfaces
3X OS Services-to-XOS Services Direct Interfaces
2L Resource Access Services Logical Peer Interfaces
2D Resource Services-to_Physical Resources Direct Interfaces
1L Physical Resources Logical Peer Interfaces
1D Physical Resources-to-Physical Resources Direct Interfaces

FIGURE 6 - Comparison of GOA Framework with POSIX and OSI Reference Models

6.1 POSIX Comparison:

The GOA Framework was developed using the POSIX model as a baseline. The Application Software GOA layer is analogous to the POSIX Applications Software entity. However, the Application Platform entity within POSIX is expanded to include the three other GOA layers: Systems Services, Resource Access Services, and Physical Resource layers. The POSIX model does not include 4L, 3L, and 2L nor does it include the 3X, 3D, and 2D direct interfaces. The POSIX Application Programmer Interface (API) is synonymous to the GOA 4D interface, and the POSIX External Environment Interface (EEI) is synonymous with the GOA 1L and 1D interfaces treated as a pair. Thus, the GOA Framework extends the POSIX model by including language describing application-to-application logical interfaces, expanding the POSIX Application Platform, and by including language to discuss Application Platform-to-Application Platform logical interface (3L and 2L interfaces).

6.2 ISO OSI Comparison:

The ISO OSI reference model is a communications model only, while the GOA Framework encompasses both communications and non-communications services. Figure 6 notes that the OSI model and the GOA Framework's application logical and direct interfaces are analogous. However, there are cases in the application of the OSI model that the application peer-to-peer interface would be considered a 3L interface and the application direct interfaces would be considered either a 3D interface or as an internal Systems Service interface not called out in the GOA Framework.

6.2 (Continued):

The Common Management Information Service (ISO 9595/9596) (CMIS) to Remote Operations Service Element (ROSE) OSI application layer interface used in the Telecommunications Management Network (TMN) applications is such an example. CMIS is a component of the OSI network management standardization effort (ISO 9595) that provides OSI management services such as Accounting Management, Configuration Management, Fault Management, Performance Management, and Security. ROSE is an application layer service which supports communication between different application processes (e.g., located on different computers) on an interactive basis. Generic structure is typically a request/reply type of interaction.

The OSI physical interface is a combination of the GOA 1L and 1D interface. In some cases, the OSI datalink peer to peer interface would be considered a GOA 1L interface, such as in the Pi-Bus standard. In such a case the OSI data link to physical direct interface would be internal to the Physical Resources layer and not visible in the GOA Framework. The remainder of the OSI interfaces are either logical or direct interfaces within the Application Platform. The logical interfaces can individually or collectively (for an appropriate subset) be grouped as the appropriate GOA logical interface. For example, transport and network peer-to-peer interfaces for some architectures might be combined as 2L logical GOA interfaces between System Service layers and the presentation and session layer peer-to-peer interfaces might be collected as 3L logical GOA interfaces between Resource Access Services layers. The OSI direct interfaces may be internal within a GOA layer that are not visible in the GOA Framework or they may correspond to GOA 3D or 2D interfaces, depending on how the representation of the architecture uses the GOA Framework. In summary, the OSI reference model and the GOA Framework complement each other regarding communications services.

6.3 TAFIM Comparison:

Figure 7 depicts the DoD Technical Reference Model (TRM) defined in the Technical Architecture for Information Management (TAFIM). The TAFIM was the basis for the Army Technical Architecture (ATA), which was in turn the basis for the DoD Joint Technical Architecture (JTA). The TAFIM provides a framework for defining technical architectures for C3I/C4I systems and its derivatives define technical architectures based on that framework, including preferred set of standards catalogs. The TRM is based on the POSIX model, as can be see in Figure 7. The TRM extends the POSIX model by categorizing Application Software into mission area applications and several support application areas. Unlike POSIX or the GOA Framework, the TRM also categorizes services available in the Applications Platform. The Application Platform service areas defined by the TRM include both run-time and pre-run-time services. The TRM addresses only 4D API interfaces and 1D/1L EEI interfaces. The TRM does not address 2L, 3L, or 4L peer-to-peer logical interfaces, 3X, 3D, or 2D direct interfaces, nor does it address Resource Access Services.

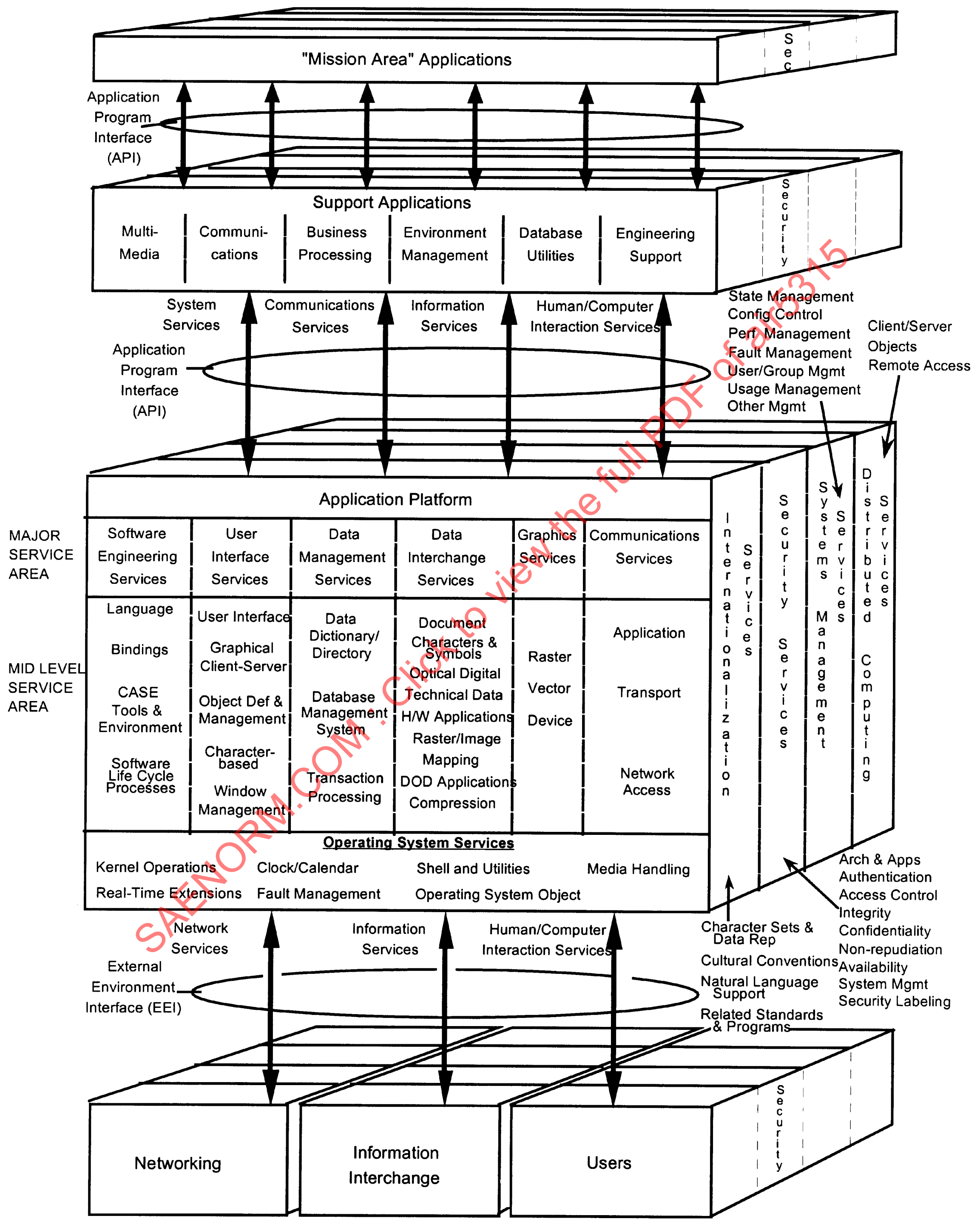


FIGURE 7 - DoD Technical Reference Model (from TAFIM v3.0)

6.4 PCMCIA Comparison:

The following are two alternatives for how PCMCIA maps onto the GOA Framework. Alternative 1 is an example of a Physical Resource that includes both hardware and software (i.e., Socket Services). This discussion assumes the reader is familiar with the PCMCIA model. Refer to [Levine] for a tutorial on the PCMCIA model. Figure 8 and the following definitions are based on material included in [Levine].

- PC Card: A device carrying out PCMCIA functions requiring Memory, I/O, and/or IRQ resources. PC Cards are inserted into PCMCIA slots.
- PCMCIA Controller: The piece of hardware responsible for mapping PCMCIA bus signals to the system bus. It is also responsible for intercepting and handling of addresses and data destined for PC Cards.
- Socket Services: The lowest level of the PCMCIA architecture providing abstracted PCMCIA controller services.
- Card Information Structure: A set of data structures present within the PC Card that describes specific information about the manufacturer and configuration options of the PC Card.
- Card Services: Highest services level of PCMCIA specifications responsible for resource and client management. It defines the API and protocols available for client use. After Card Services assigns resources to a PC Card, the PC Card can be accessed as if it were native to the system since the assignment of resources implies the PCMCIA controller has been programmed to respond to specific memory or I/O cycles and route them to the PC Card.
- Memory Technology Drivers: A PCMCIA handler directly responsible for the reading and writing of specific memory technology memory PC Cards.
- Client: The piece of software in the PCMCIA architecture that is responsible for recognizing and configuring PC Cards. There may be several clients in any one system, each of which is responsible for a subset of PC Cards.
- Memory Clients: PCMCIA clients usually responsible for memory PC Cards.
- I/O Clients: PCMCIA clients, usually responsible for recognizing and configuring I/O PC Cards.
- Super Client: A PCMCIA client that usually contains a database of PC Cards that it can recognize and configure. Super clients are usually delivered with the PCMCIA handlers.
- AT Attachment (ATA): A definition for extending disk services on IBM PC and compatible platforms.

One can argue that Socket Services is part of the PCMCIA Physical Resources since the interface to Socket Services is the only part of PCMCIA standard that is specified as to how to interface to the PCMCIA Physical Resources from higher levels of the GOA Framework. The interface between Socket Services and the PCMCIA Controller is not specified. Thus, abstractly, the Socket Services together with the PCMCIA Controller makes up the Physical Resources for PCMCIA. This is illustrated in Figure 9.

On the other hand, since Socket Services is software that directly interfaces with the PCMCIA Controller, it is valid to view it as part of Resource Access Services and only the PCMCIA Controller is part of Physical Resources. This is illustrated in Figure 10.

6.4 (Continued):

Alternative 1 could be argued as the preferred interpretation since, from a PCMCIA perspective, it is the interface to Socket Services where the interface from higher layers of the PCMCIA model is standardized.

Card Services and Memory Technology Drivers understand the PC-Card Physical Resources. Additionally, they both include drivers. Therefore, they are part of Resource Access Services.

Clients are either part of Resource Access Services or an XOS service. Memory clients may be considered as XOS Services and I/O clients may be considered as Resource Access Services. This follows since I/O clients contain drivers that actually communicate with the associated PC-Card after PCMCIA initialization, while memory clients are higher level software that use Memory Technology Drivers for direct driver interfacing.

Thus, the two alternatives are

1. Socket Services and PCMCIA Controller are part of Physical Resources, Card Services and Memory Technology Drivers are part of Resource Access Services. Clients can be either XOS services or Resource Access Services.
2. PCMCIA Controller is part of Physical Resources, Socket Services, Card Services, and Memory Technology Drivers are part of Resource Access Services. Clients can consist of both XOS and Resource Access Services.

The following are notes about the standard PCMCIA interfaces for each case

1. Case 1

   - The Socket Services API is a 2D interface
   - The Card Services API is both an internal Resource Access Service direct interface (with I/O Clients and Memory Technology Drivers) not specified in the GOA Framework and is a 3D interface (with memory clients).

2. Case 2

   - The Socket Services API is an internal Resource Access Service direct interface (with Card Services).
   - The Card Services API is both an internal Resource Access Service direct interface (with I/O Clients and Memory Technology Drivers) not specified in the GOA Framework and is a 3D interface (with memory clients).

Note the Card Information Structure (CIS) is also a PCMCIA standard. Parts of the CIS are used by Socket Services, Card Services, and clients. Thus, the CIS definition is used as part of the parameter definition within the interface of the Socket Service and Card Services APIs (and by the Socket Services interface to the PCMCIA Controller). Therefore the CIS helps define some of the direct interfaces related to PCMCIA. The CIS is NOT a logical interface in the sense of the GOA Framework.

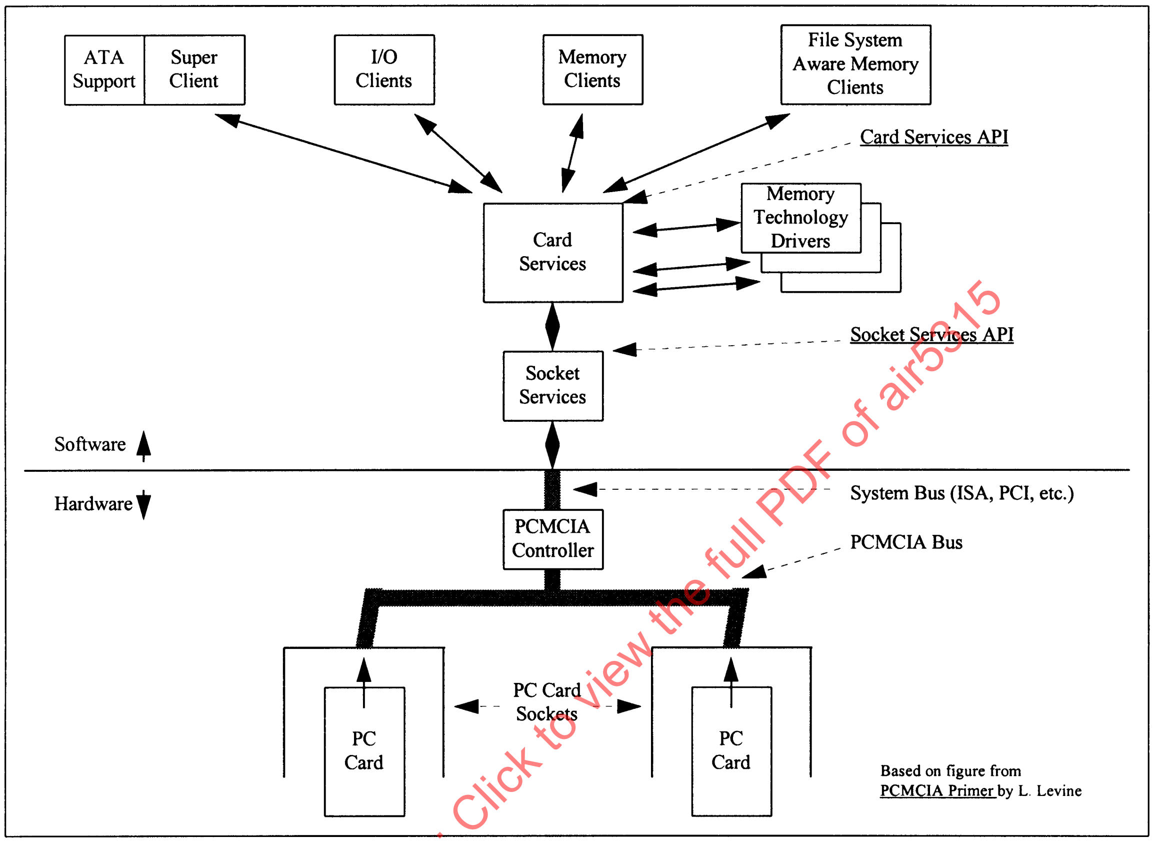


FIGURE 8 - PCMCIA Architecture