

An Open Architecture for Air Transport

ABSTRACT: This document has been prepared with the objective of defining an open architecture for inter-operable air transport systems within an environment based upon the OMA and adopted OMG specifications.

An architecture aims to identify important players, roles and relationships of air transport participants. It defines the major components needed to support participants in the execution of air transport transactions. Through the definition of interfaces and semantics of supporting services and facilities, an architecture provides an object framework to software developers that allows the implementation of coherent parts of an overall air transport system. The external interfaces permit interoperability with other means of transport, i.e. intermodal transport. The internal interfaces are intended where possible to support reuse across other domains.

DATE: 9 April 1997

AUTHORS: members of the Transport Air Working Group of the Transport SIG

RELEASE: 1 (DRAFT)

Table of Contents

FOREWORD	6
PREFACE	7
About this Document	7
Purpose	7
Intended Audience	7
Acknowledgments	8
OVERVIEW	9
An Interoperability Framework for Open Air Traffic Control Architecture	9
The Enterprise Model	9
The Architecture	12
Operational ATC Enterprise	12
Relationship to ongoing Technology Adoption Processes (C4I, ...)	14
Air Transport Requirements	15
Air Traffic Control Requirements	15
REFERENCES	16
AIR TRANSPORT SERVICES AND FACILITIES	18
AIR TRAFFIC CONTROL SERVICES AND FACILITIES	18
NAS ARCHITECTURE	18
OBJECT MODEL DEVELOPMENT FROM THE NAS ARCHITECTURE FUNCTIONAL DECOMPOSITION	18
OBJECT MODEL DEVELOPMENT FROM THE NAS ARCHITECTURE INFORMATION ARCHITECTURE	18
ATC COMMUNICATIONS SERVICES AND FACILITIES	18
Overview of Communication requirements	20
Communication protocols	20
Services required	20
Object Oriented Architecture	20
Objects	20
Services Common to all objects	20

Problems to be solved	20
Communication protocols	21
Surveillance Radar	21
Interfacility Communications	22
Communications protocols for peripheral and external domains	23
Physical Interfaces	24
Services Required	25
Object Oriented Communication Architecture	29
Objects	29
Sensor Object	31
Device Object	32
Port Objects	33
Common Services required of all objects	34
Problems to be solved	35
ATC DISPLAY SERVICES AND FACILITIES	35
ATC TRACKING SERVICES AND FACILITIES	46
ATC REALTIME DATABASE	46
ATC SCHEDULING AND SEQUENCING	46
ATC WEATHER SERVICES AND FACILITIES	46
NAS INFORMATION MANAGEMENT SYSTEM	47
Client Reservation System	47

Table of Figures

<i>Figure 1 Strategic to Tactical Levels Operational ATC portion of Air Transport Architecture</i>	<i>9</i>
<i>Figure 2 Strategic to Tactical Levels Weather portion of ATC section of the Air Transport Architecture</i>	<i>10</i>
<i>Figure 3 Strategic to Tactical Levels Maintenance portion of ATC section of the Air Transport Architecture</i>	<i>11</i>
<i>Figure 4 Use of CORBA in FAA William J. Hughes Technical Center Integration and Interoperability Facility</i>	<i>12</i>
<i>Figure 5 Current En Route Architecture</i>	<i>27</i>
<i>Figure 6 Future En Route Architecture</i>	<i>28</i>
<i>Figure 7 Object Oriented Communications Architecture</i>	<i>30</i>

Foreword

This architecture is intended to address those areas of air transport which are of interest to any attendee at meetings of the Transport_Air working group.

The interests of the present attendees are concentrated in the air traffic control portion of air transport. This includes an interest in free scheduling, free routing and eventually free maneuvering. Free scheduling might present an interface to the reservation activity, in the sense of strategic planning for scheduling based upon statistics of reservations, and possibly in the area of small business aircraft whose schedules are adaptive to passenger demand on a relatively short time scale.

The current state of air traffic control system architecture is that several heterogeneous platforms are employed, interoperating, but that the communications between the platforms is not based upon widely available general purpose communications software standards. The life-cycle maintenance costs of the current software might be reduced if commercial off the shelf software could be employed that would meet the needs of air traffic control systems.

The ability to “plug-and-play” with products from a variety of vendors is limited by the present implementation of the air traffic control algorithms. A vision of a new air traffic control system in which it is straightforward to upgrade the system incrementally, by the replacement of relatively small software components, available from multiple vendors, guides the development of this open architecture.

Preface

About this Document

This document is divided into several sections, permitting concentration on different areas within air transport.

One section of this document presents a high level specification of requirements for an air traffic control system. It identifies those interfaces, component interfaces, necessary to interoperate with other components comprising the air traffic control system. This section of the document proposes, either by reference to other documents or by material contained herein, high level requirements specifications relating to:

- air traffic control operations
- air traffic control infrastructure maintenance
- air traffic control related weather measuring and predicting systems

Another section describes a client reservation system, and its interface with the Airline Operations Control Centers, which provide one interface between the reservation system and the air traffic control system.

Purpose

The purpose of the open architecture document, as it evolves, is to develop a consensus among the partners: suppliers, users, maintainers, administrators, interested in the air transport business about the computer automation architecture to support air transport. Interface syntax and semantics identify products which can be used as components with which to construct the air transport system.

As the evolution rate of the document slows, the purpose of the document is to describe the interface syntax and semantics of components (which would be products) which are useful in the construction of the air transport system.

Intended Audience

The intended audience is the suppliers and users of air transport, including the regulatory agencies involved with air traffic control.

Acknowledgments

The preparation of this document reflects the contributions of many people, some of whom have attended meetings of the Transport_Air Working Group, and others of whom are contributors to the field whose work has been recognized herein.

Overview

An Interoperability Framework for Open Air Traffic Control Architecture

The Enterprise Model

An enterprise model for the operational air traffic control system is shown in Figure 1.

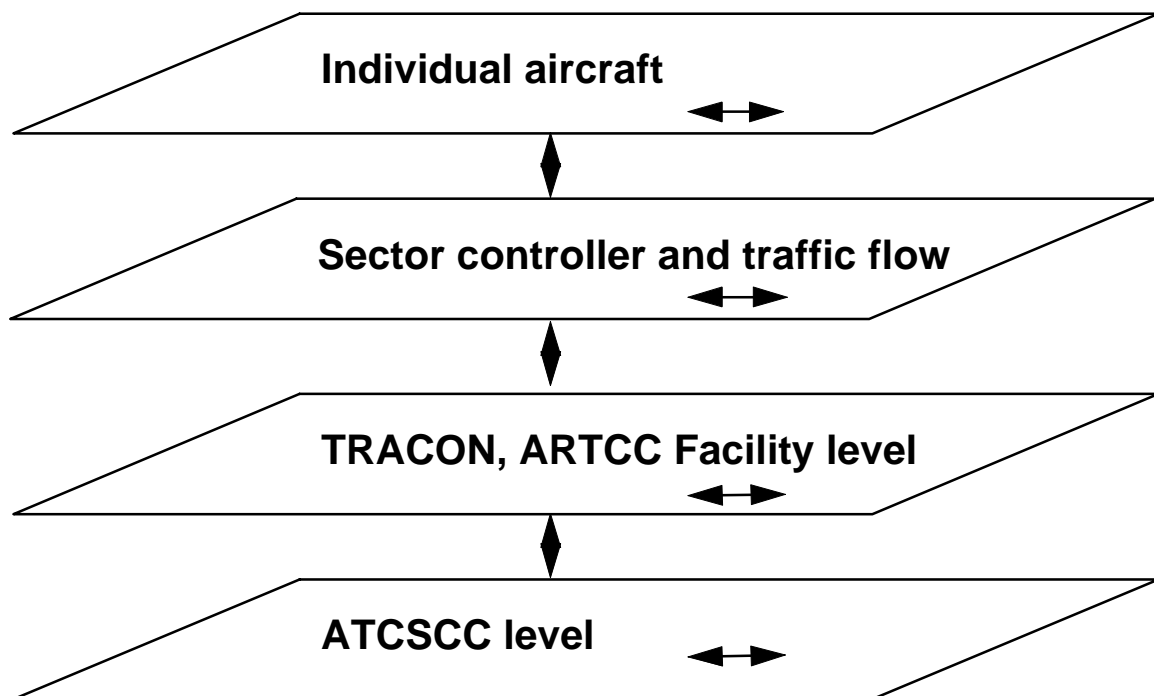


Figure 1 Strategic to Tactical Levels Operational ATC portion of Air Transport Architecture

An enterprise model for the weather related to the air traffic control system is shown in Figure 2.

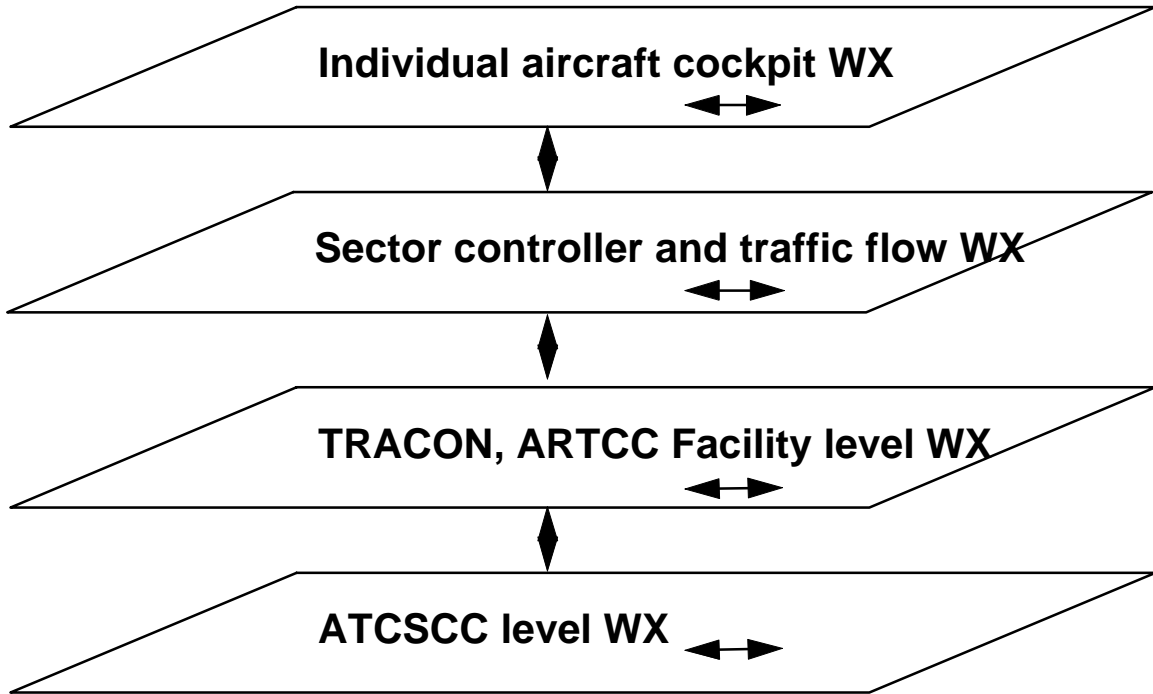


Figure 2 Strategic to Tactical Levels Weather portion of ATC section of the Air Transport Architecture

An enterprise model for the weather related to the air traffic control system is shown in Figure 3.

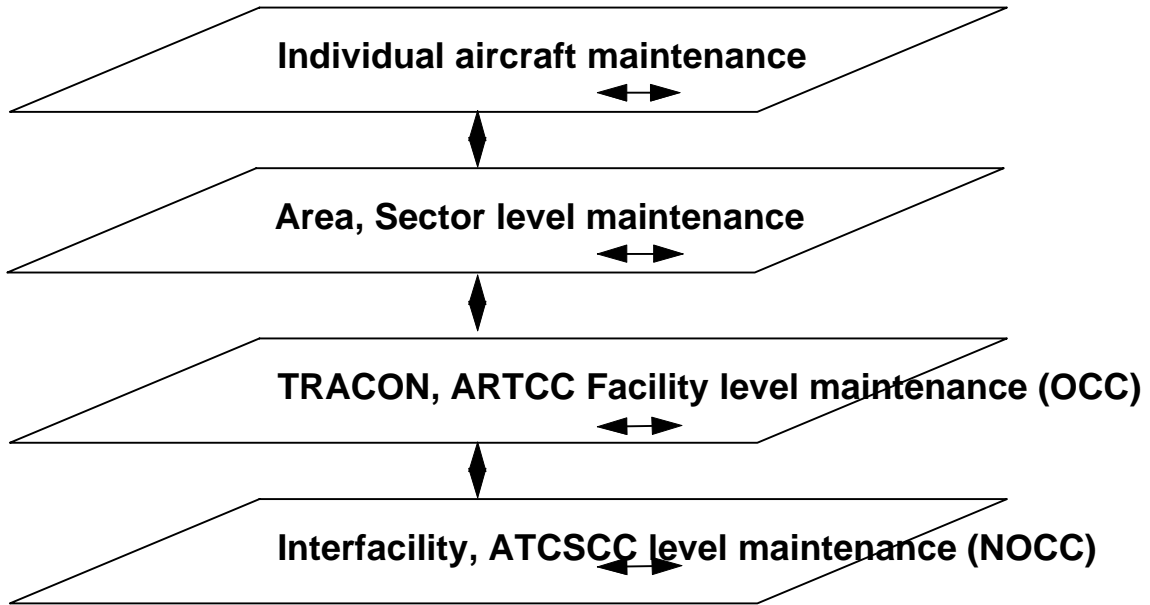


Figure 3 Strategic to Tactical Levels Maintenance portion of ATC section of the Air Transport Architecture

The scope of the joint enterprises includes simulation, for example, as shown in Figure 4.

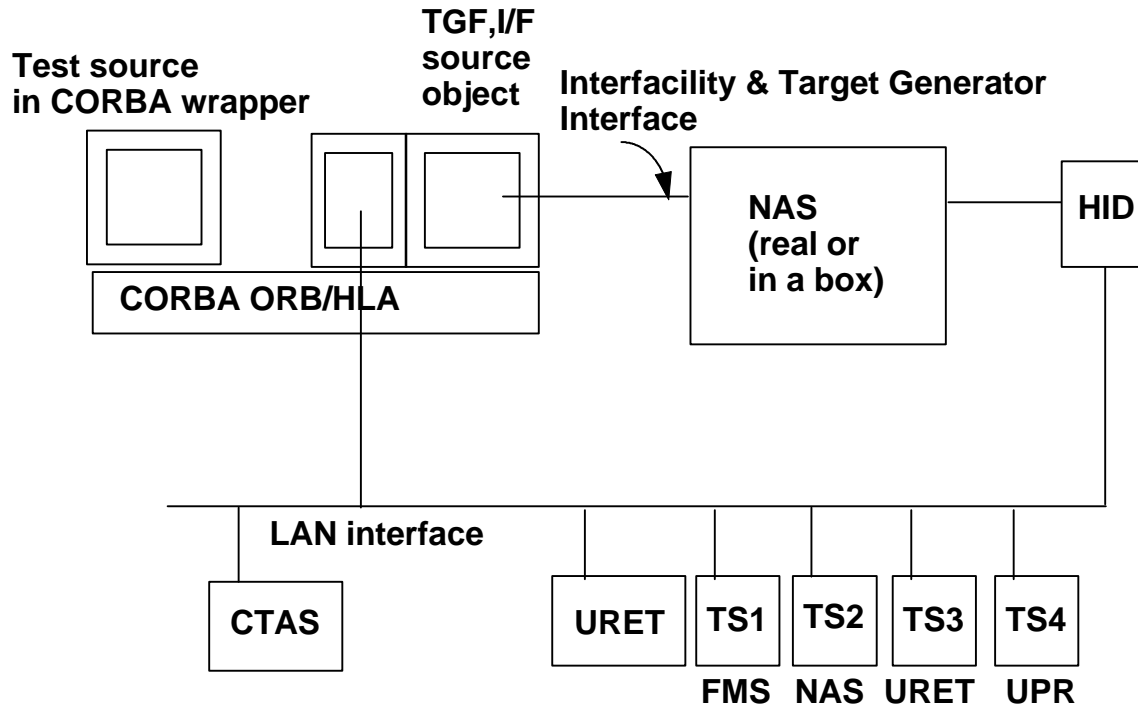


Figure 4 Use of CORBA in FAA William J. Hughes Technical Center Integration and Interoperability Facility

The Architecture

The work undertaken in developing this architecture is driven by the interest of enabling the Federal Aviation Administration (FAA) to utilize the products of many vendors in the construction and upgrading of air traffic control systems. Many existing standards are adopted herein, and some new ones are proposed.

Operational ATC Enterprise

Communications Services

The work by Camber/DSC has been received with interest by the ORB Object Services (ORBOS) group. Possibly the Object Services will be augmented to use the architecture, which would then be made available through ORBOS to application domains including air transport.

Recording Services

Display Services / Display Facilities

There is a user interface CORBA facility.

ISA Systems and Orthogon System have interoperating products used by Eurocontrol, built upon X-windows that provide ATC related windowing/display system.

(Untracked) Aircraft State Facilities

The representation of an aircraft in the computers supporting Air Traffic controllers could be an object, created with a factory object using the Object Lifecycle Service. This implies that the implementer of an aircraft object support the Lifecycle operations move, copy and remove. Use of move might accompany a handoff, use of remove might accompany remove track and remove strip, and use of copy might accompany the processing of a stereo flight, especially if such a flight is metered across a fix as individual aircraft.

System Resources Corporation has begun to model aircraft as objects, in a way consistent with the NAS Architecture.

Tracking Facilities

Intention Facilities

Aircraft Model Facilities

Trajectory Facilities

Airspace Facilities

The relationship CORBA service provides a means of representing a containment relationship, with container and contained roles which might be useful in representing subdivisions of airspace.

Sectorization Facility

Special Use Airspace Facility

This facility might be able to take advantage of work being done in the CORBAgis working group.

Terrain Facility

This facility might be able to take advantage of work being done in the CORBAgis working group.

Conflict Detection Facilities

Conflict Resolution Facilities

Scheduling Facilities

Sequencing Facilities

Spacing Facilities

Handoff Facilities

There is an information management CORBAfacility which might be generalizable to be relevant to the exchange of information between ATC facilities in handoffs. The CORBA relationship service might also be useful here.

ATC-related Weather Enterprise

Nowcast Facilities

Forecast Facilities

ATC Infrastructure Enterprise

Utilization Statistics Facilities

The task management CORBAfacility which was under

Relationship to ongoing Technology Adoption Processes (C4I, ...)

Component Definition Language Specification

Air Transport Requirements

Air Traffic Control Requirements

References

Source	Document
FAA, AUA-200	FAA's En Route Architecture Study Findings, May 16, 1997
FAA, ASD-110	NAS Architecture and revisions http://www.nasi.hq.faa.gov/nasiHTML/nas-architecture/v2/4_ALL.html
FAA, ASD-100	FAA: http://asd.orlab.faa.gov/files/menu.ht
MITRE/CAASD	http://www.mitre.org/centers/caasd/Papers/WP/90W542/index.html
Eurocontrol	http://www.eurocontrol.be
RTCA	SC-169, SC-162, SC-147, SC-182 Operational Concepts and Data Elements Required to Improve Air Traffic Management (ATM) - Aeronautical Operational Control (AOC) (Ground-Ground) Information Exchange to Facilitate Collaborative Decision-Making
Camber/DSC	communications architecture
System Resources Corporation	conversion from NAS Architecture functional model to object oriented model
Lockheed Martin Air Traffic Management	En Route Infrastructure Design Document, March 3, 1997
MIT/Lincoln Laboratory	Evans and Ducot, "The Integrated Terminal Weather System (ITWS)", MIT Lincoln Laboratory Journal, Fall 1994, Vol. 7, No. 2.
MIT/Lincoln Laboratory	Cole and Wilson, "The Integrated Terminal Weather System Terminal Winds Product", MIT Lincoln Laboratory Journal, Fall 1994, Vol. 7, No. 2.
Mississippi State University, 1994	Roy, Philip, Carter, "A Comparison of elements used in structure and object-oriented models."
Mississippi State University, 1994	Dorabshaw, Carter, Philip, "A procedure and criteria for the evaluation of case tools to support a software development environment using and IDEF-to-OOD paradigm."
Mississippi State University, 1994	Philip, Carter, "A comparative case study of system analysis using multiple structured and object-oriented models."

--	--

Air Transport Services and Facilities

Air Traffic Control Services and Facilities

Introduction

The Air Traffic Control Services and Facilities are described in more detail below. They are, in some cases, descriptions of completed work or work in progress.

NAS Architecture

Object Model Development from the NAS Architecture Functional Decomposition

Contributed by

Paula Nouragas, FAA Research and Development Human Factors Laboratory at the William J. Hughes Technical Center, and
John Richards, System Resources Corporation

Object Model Development from the NAS Architecture Information Architecture

Contributed by

Felix Rausch, FAA ASD-110

ATC Communications Services and Facilities

Contributed by Camber/Digital Systems Corporation

This work has been received with interest by ORBOS.

Ideal World

Provide services transparent to the underlying infrastructure with infinite extensibility over wide range of protocols.

Allow the plug and play of commercial communications subsystems to be utilized without software changes.

Maintain compatibility with legacy systems while building for the future.

Communication objects are instantiated based on a service that they can provide. Communication objects provide a common, transparent way to receive and transmit information from many different types of interfaces.

Overview of Communication requirements

Communication protocols

- Radar
- Interfacility
- Communication to peripherals
- External Systems and Domains

Services required

- Send/receive data
- Broadcast/connection
- Data recording
- Status and statistics
- Fault tolerance redundancy
- Extensibility

Object Oriented Architecture

Objects

- Sensor Objects
- Device Objects
- Port Objects

Services Common to all objects

- Configuration
- Status and statistics
- Message transfer

Problems to be solved

- Need to utilize COTS Communication controllers
- Need message encapsulation and delineation

Communication protocols

Surveillance Radar

Synchronous data transfer 2400 - 19,200 baud

Radar - Characterized by continuous small packets and varying protocols and message formats.

Variety of message types

- Beacon
- Weather
- Search

Radar Communication Protocols

- CD-2 12 bits + parity, 13 bit sync, 4, 7 fields per message
- ASR3 ADCCP
- ASR4 ADCCP
- ASR9 Enhance CD2 4, 7, 32 fields per message
- Asterix HDLC (Currently four versions)

Utilizes EIA 422, 485, 232, EIA 530

Interfacility Communications

Utilized for communication with other Air Traffic Facilities to exchange flight plans and facility hand-offs of planes from one center to another.

Communication format

- Currently proprietary synchronous
 - 17 bit sync, 8 bits + parity, LRC, SOM, EOM
- HDLC - Planned for use in the future to replace Interfacility

Communications protocols for peripheral and external domains

- FDIO - Flight Data Input/Output
 - Modified ADCCP used to send remote Flight Strips.
 - Protocol was implemented wrong and NS and NR fields are reversed.
- CTS - Coded time source Asynchronous interface to time source.
- X.25 - Utilized to receive weather data, and communicate with NEXRAD and NADIN.
- GPI/GPO Utilized for custom parallel interface

Physical Interfaces

- Synchronous and Asynchronous
- EIA - RS 485
- EIA - RS 530
- EIA - RS 232
- Custom

Currently most interfaces are running at 2400 baud synchronous. Would like new interfaces to run at 9600 baud or higher.

Services Required

The following capabilities are required of the interfaces.

Data Transfer

- Basic Send/Receive of data messages
- Data Filtering
- Data reformatting and conversion
 - Radar conversion, ASCII to EBCDIC, etc.

Protocol handling

- Automatic switch - over to backup ports if an adapted error rate is exceeded.
- Adjustable data rates
- Flow Control

Port Control

Line Control - The port should be able to disable itself to allow modem sharing devices to work.

- Baud Rate
- CRC
- EOM characters
- link level protocol
- time-outs
- Physical Media

Performance and statistics

Each port object must be capable of reporting the current statistics.

- Total number of messages
- Total number of bytes
- Line Utilization
- Number of Errors

Data recording

- All data need to be recorded with time stamp.
- Simulation - recorded data should be capable of being played back.
- Adaptation should control what gets recorded and where.

Current EnRoute Architecture

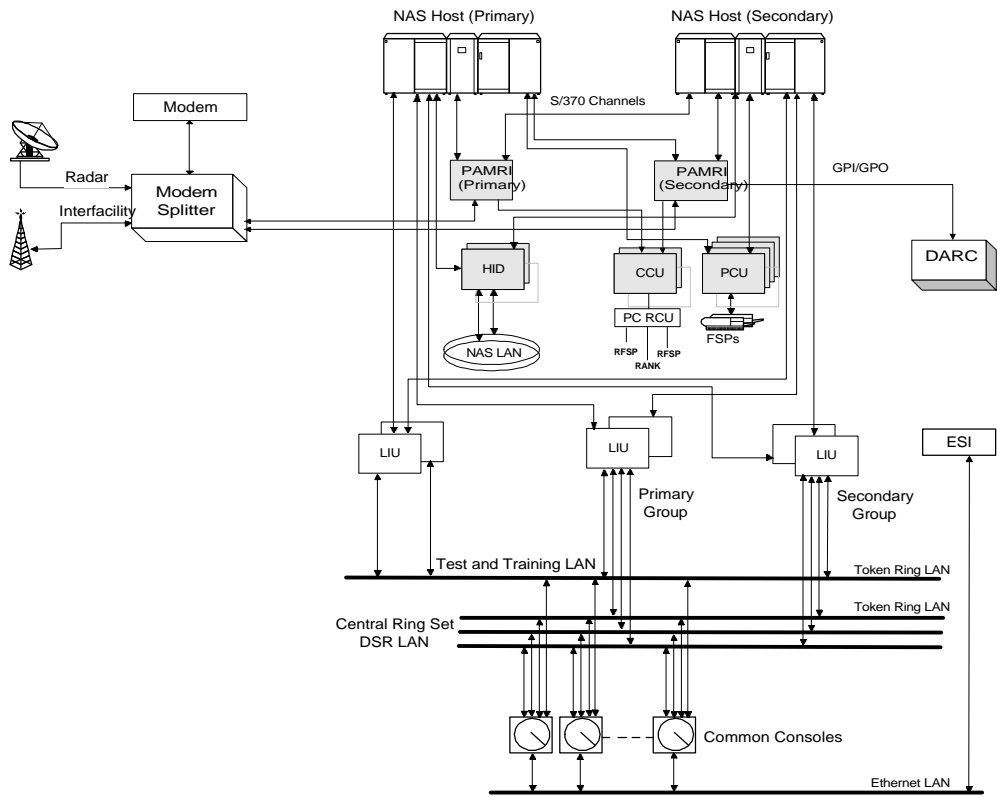


Figure 5 Current En Route Architecture

Future EnRoute Architecture

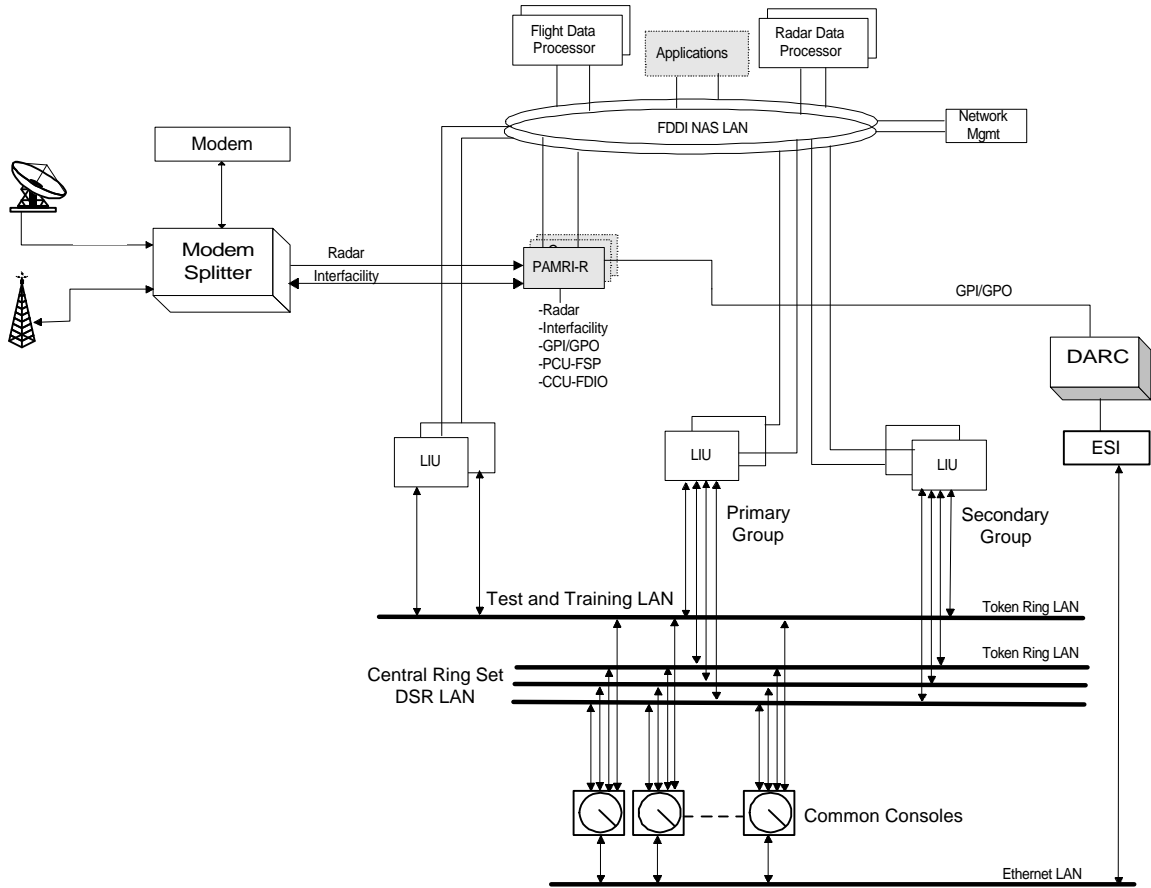


Figure 6 Future En Route Architecture

Object Oriented Communication Architecture

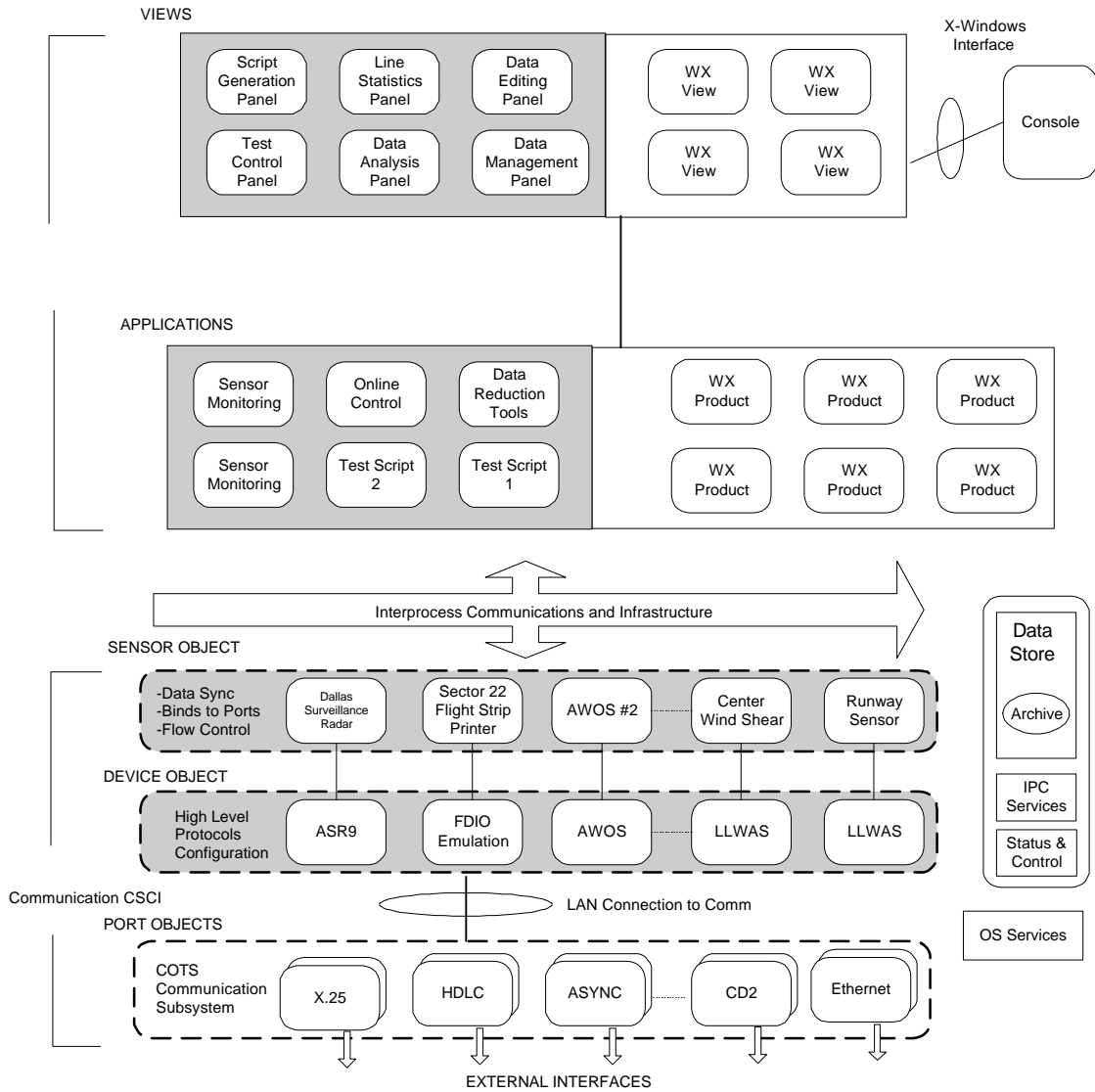
Objects

Sensor Objects - Application services for sensors or communication services

Device Objects - Provides Network transparency and device protocol specific handling

Port Objects - Represents physical or logical communication channels and ports.

Object Oriented Communication Subsystem Example



Communication Architecture

Figure 7 Object Oriented Communications Architecture

Sensor Object

The sensor object provides a consistent common interface to the application.

The application instantiates objects to provide services that it requires.

Capabilities and Services

- Data Transfer
- Data Conversion
- Flow Control
- Filtering
- Message Encapsulation
- Support for transport connections
- Message Notification

Device Object

Provides a consistent interface between the Sensor and the Ports

May be considered as a Network Layer providing transparent redundancy, backup paths, and message routing.

Capabilities and Services

- Responsible for higher level protocol (example may poll for data, handle acknowledgments)
- May bind to one or multiple Port objects
- May bind to one or multiple sensor objects
- May provide proxy capabilities for some port objects
- Provides data recording interfaces
- Provides statistics and performance monitoring capabilities.
- Message delineation

Port Objects

This object encapsulates all of the physical and link level control required by the interface.

There exist a single port object for each physical or logical communication line.

Capabilities and Services

- Data transfer
- Configuration
- Hardware control capabilities
- Link level protocol
- Status and error monitoring

Common Services required of all objects

- Configuration
- Status and statistics
- Data recording capabilities including time-stamping
- Flow Control

Problems to be solved

The current system must support a wide variety of interfaces and must be capable of being expanded with additional ports and types of interfaces.

The communication capabilities must be available to separate diverse applications without impact on existing applications or functionality.

The two main issues involving the current architecture are:

1. Common access to information for Applications. (Message encapsulation) of diverse data streams.

2. Ability to utilize COTS communication controllers to:
 - Allow the utilization of the latest protocols and technology
 - Easily increase the number of ports and resources
 - Reduce development time
 - Reduce maintenance and logistical cost.

ATC Display Services and Facilities

Contributed by ISA/Orthogon System / Deutsche Flugsicherung GmbH
Copyright 1997 by DFS Deutsche Flugsicherung GmbH

1. Purpose and Scope

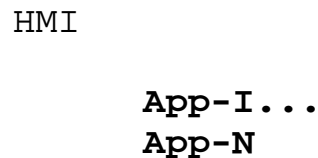
The objective of this paper is to explain the principles and the features of HMI-Application-interface. This paper does not provide a complete and detailed definition of the interface, but the information contained in this document is fine-grained enough to help estimate the effort in software development, i.e., the adaptation of existing ATC software to match the ITS technical requirements.

The reader should be aware of the intention to use this interface for applications like weather information, flight data coordination, flight control etc. It is not intended to make use of that interface in a time critical radar data processing component (But due to the fact that displaying of radar data is based on a unidirectional communication nearly all advantages summarized in chapter 3 are also true for that kind of application - i.e. this restriction of the use of the interface does not lead to relevant disadvantages.)

Since the definition of the Interface is not yet in a final state, parts of the described elements may be subod to change.

2. The basic architecture

The DFS has the need to define some architectural structures which provide the capability of being integrated for each of the tower components. One of these structures is gtvn by the 'separation of HMI and application',



Fig, 2-1: Separation of HMI-software from ATC applications

The term 'application' stands for software which handles the ATC work like 'organizing weather information' or 'controlling state transition of flight plans' etc. HMI (human machine interface) is the software which is responsible for the presentation of information, the look & feel, the controlling of user dialogs, etc.

The 'HMI-Application-Interface' is somewhat in between those two kinds of software parts (which are running in different address spaces, i.e., HMI and application are separate processes). It is based on CORBA. Such an interface is obviously not another kind of graphical library or another collection of smart widgets. An analysis of the roles

of the HMI-software and the applications shows that the HMI-Application-Interface has two main purposes:

1. providing access to an application object to achieve a manipulation of that object derived from a user input via HMI
2. propagate attribute values and value changes of the application object to the HMI component where rules determining the kind of presentation to the user

(The term application object is used regardless of whether the implementation language is object oriented or not. A typical application object would be a flight plan, a cluster of runway lights or a wind object.)

Concerning the first aspect mentioned above the application object is a server object. The remote HMI has access to the methods of that object via a CORBA, i.e., this object has a CORBA interface specified in IDL. From that point of view the application object is exclusively responsible for providing clearly defined and robust methods. It does neither know where the input comes from nor how the input is done. The invocation of a method of an application object is a result of processing an input in the HMI component (this input can be a push of a button, a click on a menu bar or some key strokes). If someone would change the behaviour of the input procedure or would change the complete look & feel of the user interface the implementation of the application object would not be affected.

Fig- 2-2, An application object acts as a server offering the HMI component some 'levers' (methods) for manipulating the object inside

In the second case the HMI acts as an observer on the application object. Whenever an attribute of an application object changes its value the HMI does its work: driven by rules defined in the HMI component, information is displayed or the color of a graphical object changes or something else.

```
attribute 1  
attribute 2
```

```
attribute N
```

Fig- 2.3: The HMI component acts as an observer on the attributes of an application object

There are already some HMI components on the market which make use of that architecture. But the communication between application and HMI is still vendor specific. The idea is to make use of the advantages of CORBA at that point.

3. The advantages

With that architecture combined with the technical features of an industrial standard like CORBA the following advantages can be achieved:

- capability of integration of different tower components under a common user interface
- a change of the user interface - i.e. changing the layout or changing the kind of widgets etc. - does not lead to changes in the application code (The DFS has the experience that most of change requests are pure HMI requirements - so that is an important economical requirement)
- maintenance of HMI and application can be done separately. e.g. maintenance can be done by different teams - typically a

HMI expert has to have other skills than an expert for a weather information system.

- independent single source of the application objects is given. No appropriate counterpart object in the HMI component has to be programmed for adapting the HMI component.
- development of HMI and application can be done separately. Since IDL scripts are the only common part of the two components (HMI and application) the application programmer does not need any library from the vendor of the HMI component. The development environment can be completely different. No needs for upgrades in the application if HMI component changes.
- independence of the application from the HMI product. No HMI features are visible at the interface. In addition that supports 'clear semantics and 'clean contexts' in development of software: an source of an application will contain code that handles application specific procedures only because the developer of an application has no access to presentation objects and no possibility to create presentation objects anyway.
- In most of the cases of changes less tests and re-tests. Changes are more restricted to a component (mostly HMI) that means other critical sequences of code will not be touched. A mixed implementation (when HMI-procedures are part of an application software) would bear much more risks.
- hardware independence
- independence from the operational system (HMI and application may run under different OS).
- * independence from the programming language (C, C++, smalltalk, etc.)

4. The technical details

Chapter 2 is a description of principles of an architecture that separates HMI and applications. But there are still some questions that are not yet answered in that chapter, e.g., How does the HMI know about the existence of an application object and how does it get the ID of an application object? How to hide the number of displays or the number of HMI components from an application?

4.1. Providing access to an application object

Remote access to an application object means the object needs a CORBA interface. The interface of the application object has to be specified in CORBA-IDL - the 'interface Definition Language'. Derived from an IDL-script, an IDL compiler generates a server skeleton source code which is the further basis for developing the full object. (Note that the ORB product and its IDL compiler can be different from that ORB the vendor of the HMI component uses for development).

Once an application is completed and installed a remote HMI component is able to invoke methods of the application objects if

- the HMI component knows the object ID
- the HMI component knows the interface

The way the HMI gets the object ID is described in section 4.3. The interface definition can be retrieved from the Interface Repository of an involved ORB (The interface definition is only relevant for software which will make use of the CORBA Dynamic Invocation Interface - the 'DII'. But since HMI components will handle CORBA calls in an asynchronous mode, use of that kind of interface is a typical implementation). Provided that an HMI component makes use of the DII and processes its HMI-rules interpretative no re-compilation of the HMI component is necessary to introduce a new kind of application object (use of DII allows method access by specifying a string representing the method name). All other kinds of implementation (using CORBA or not) of an HMI component will require a re-

compilation when a new kind of application object - that means new interfaces - will be used.

4.2. Indication of attribute changes

In chapter 2 the HMI component is introduced as an observer of attributes of the application objects. The ideal case would be to let the application object be the totally passive part in that observation business. From the view of an application object this would implicitly mean to have no extra lines of code for notifying attribute value changes to the remote HMI component. In reality with standard tools (i.e. standard preprocessors, standard compilers, standard language mapping) this is not possible. Therefore, it is obvious that each 'set attribute routine' of an application object needs additional code for invoking an object which is responsible for propagating the attribute value to the HMI component.

The box below shows in extracts the IDL definition of such an 'event distributor object (ED) that notifies attribute changes to the HMI.

(The difference between the '*...UpdateAttribute*' and the '*...NotifyAttribute*' routines is the runtime mode. The Update routines are running in asynchronous mode.)

An ED-object is located in the HMI component and is accessible from the application object. One should keep in mind that each application object has its own 'personal' ED-object in the HMI component.

The ED is an uniform thing. Regardless of the application class definition or the kind of inner structure of the application object the ED interface looks always the same. But the number of available data types for attributes is restricted to a subset of basic data types and its appropriate array types.

```
interface HA-ED {
    exception HA-ErrorHandle {
        long      ErrCode;    // ID number of an error
        string    ErrMsg;    // text message, error description
    };

    // methods for notifying value changes:
```

```

        oneway    void HA_i_UpdateAttribute(in string Name,
            in long Value) raises(HA_ErrorHandle);

            void HA_i_NotifyAttribute(in string Name,
            in long value) raises(HA_ErrorHandle);

        // etc. routines for some other types (float, string,
        and appropriate arrays of that types)

        oneway    void HA_UpdateAttribute(in string Name,
            in any Value) raises(HA_ErrorHandle);

            void HA_NotifyAttribute(in string Name,
            in any Value) raises(HA_ErrorHandle);
};

        // end of definition HA-ED

```

4.3. Object lifecycle and identification of objects

Due to the fact that an HMI component shall observe application objects it has to know about the existence of these objects and it must know each particular object-ID. Therefore each process of creation and deletion of an application object has to be notified additionally to the HMI component.

The notification of the object's creation to HMI can be done in the object's constructor, accordingly the notification of its intended deletion to HMI can be handled in its destructor. With the notification of its creation process the application object delivers its object ID to HMI. As a result the HMI delivers the object ID of an ED object which then is exclusively assigned to the application object. With the notification of the application object's deletion the assigned ED in the HMI component can be deleted too.

So, an ED exists as long as its assigned application object exists. Handling of creation and deletion of ED objects is naturally done outside the EDs. That means the HMI component has to have a single object that receives the create- and delete notifications and controls the creation, assignment and deletion of the EDs.

The following box shows the most important parts of an IDL definition of such an object.

```

interface    HA_Admin {
//...
    // notifying creation and deletion;
    short    HA_Create(in Object ApplObjRef,
                       out HA-ED EObjRef,
                       in string ApplTag,
                       in string ObjTag) raises(

    short    HA_Delete(in HA_ED) EObjRef) raises( ...

    short    HA_mDelete (in string ApplTag, in string
                       ObjTag)...

};    // end of definition HA_Admin

```

The input parameter 'ApplObjRef' in the HA_Create method is the object ID of the application object. The 'EObjRef' is the ID of the assigned ED object which is then available to the caller (application object) with the return of the routine.

The additional parameters 'ApplTag' and 'ObjTag', are tags which can be used by the application to organize its objects in sets. In specifying tags the application process is able to initiate the deletion of a whole set of objects by invoking the HA_mDelete method. That is needed in bad error situations where an application dies and its application objects are destroyed without being able to notify HMI. Invoking the HA_mDelete method in the start up phase of the application process the still existing 'zombie ED objects' - i.e. EDs where the assigned application objects are lost - will be cleared; that eliminates the need to restart the whole HMI.

Both kinds of tags are strings. The application is free to assign any string to the tags. An 'application tag' is mandatory, an 'object tag' is optional. An object matches the given tags if the application tag and the object tag are identical - an empty object tag is treated like a wild card.

4.4. The processes at runtime

Fig. 4.1 shows the processes between the interface entities of application and HMI component.

Fig. 4.1: invocations and events at the interface, process of HA_mDelete not shown

1. Creation of an application object leads to invocation of HA_Create of the sole HA_Admin object
2. Due to 1 creation of an HA_ED; when finished HA_Create returns to the caller. After 1 and 2 the HA_ED knows the object ID of the Appl_Obj and *vice versa*.
3. Due to an attribute value setting Appl_Obj invokes an HA_UpdateAttribute or an HA_NotifyAttribute routine of the ED.
4. HMI determines the defined presentation dependent on the indicated value and dependent on HMI rules.
5. A user input leads to an invocation of a method of the Appl_Obj.
6. Destructor of Appl_Obj invokes HA_Delete routine
7. HA_ED is deleted

4.5. Development of applications

Fig- 4.2 shows the steps in developing an application. The IDL script contains the interface definition of an application object. An IDL compiler generates the application code frame and loads the interface repository of an ORB.

Dependent on the features of the HMI component the new interface is learned dynamically or the HMI is rebuilt automatically (i.e., without adding code manually).

executable application

Fig. 4.2 development process of an application

5. **Summary of the technical characteristics**

At this point one can see how the implementation of an application object, for which use by HMI is intended, differs from a 'regular' object. First, it has to have a CORBA interface for providing access to its relevant methods. Second, it has to include program code for notifying lifecycle events and attribute changes to the HMI component.

Each potential candidate of application object which eventually may be used by the HMI component must be prepared for that; the decision for manipulation or presentation of an application object is exclusively made in the HMI component.

The presentation performed by the HMI component is driven by observing single attribute values of an application object. The number of possible data types for attributes is restricted.

The manipulation of an application object is possible by invoking its methods (if a CORBA interface is defined for that particular method). Here the HMI component is not restricted to methods which manipulate single attributes. Every defined

method can be invoked if the HMI is able to provide all the parameters the method needs for invocation.

The single source principle is supported. An application object has to be defined only once. Use of a new kind of application object interface is derived from the standardized Interface Repository of the ORB. The interface to the HMI component is the fix part of the HMI-Application-Interface.

The number of displays or connected input devices is not visible to the application and is completely handled in the HMI component.

ATC Tracking Services and Facilities

Contributed by Lockheed Martin / Owego

ATC Realtime Database

Contributed by Lockheed Martin / Air Traffic Management

ATC Scheduling and Sequencing

Contributed by Greg Wong, NASA Ames Research Center

ATC Weather Services and Facilities

Contributed by Seth Troxel, MIT / Lincoln Laboratory

NAS Information Management System

Contributed by

Client Reservation System

Contributed by