AN INTEGRATION FRAMEWORK FOR AIRPORT AUTOMATION SYSTEMS

Ningjiang "Jay" Cheng, The MITRE Corporation, McLean, Virginia

1. Introduction

A large airport typically has dozens of automation systems that require automatic information interchange among them to ensure safe and efficient airport operations. However, those systems were often developed by different vendors and were not designed to be interoperable, which makes systems integration a very complicated matter. Ad hoc point-to-point integration often proves to be problematic. An integration framework is needed to avoid "spaghetti integration". This paper presents an integration framework for defining the integration information requirements and designing the systems integration architecture.

1.1 Background

The management and operations of a successful modern airport has always been a complex exercise. Today, pressure from new airline alliances and low cost operators, together with airport privatization and acquisition, is resulting in a far more competitive environment between airports. Operational efficiency needs to be taken to a more advantageous cost base without compromising growth and quality. This requires increased automation and integration of the airport systems, and a clearer, more consistent and timely view of the total airport business.

Most airports have established point to point system interfaces. When the number of systems to be integrated increases, the number of interfaces becomes unmanageable. This tactical approach to airport systems integration proves difficult to maintain, enhance, and extend. Information storage and access are fragmented across these isolated links, leading to slower decision making and less efficient teamwork in an increasingly complex airport environment. It also brings the risk that the common data that exist in various different applications may not be consistent. As competition and the need to maximize profits and efficiencies grow, airports have to find a better way to integrate these "fragmented" systems. This paper describes a framework that provides a better way to integrate airport systems. This integration framework guides the system integration not only for existing applications, but also for the design and procurement of a new airport's automation systems.

1.2 The Integration Framework

There are two basic aspects of the systems integration in a complex airport environment. One aspect is the approach for collecting and managing integration requirements. The other aspect is the conceptual integration system architecture used to guide the physical system architecture design and evolution. These two aspects are described in two models: an integration information model and an integration system architecture model.

The *integration information model* describes the airport business integration requirements by capturing integration information elements in terms of standard notations such as Unified Modeling Language (UML). Due to the complexity of a large airport, an organization and a process need to be established to manage systems integration requirements systematically.

The *integration system architecture model* provides a guideline for airport Information Technology (IT) architecture planning and integration system design and development. This model identifies important system components for the integration, among which are integration middleware and an Airport Operational Database (AODB). An AODB provides a centralized management of common airport operational data with high data quality and reliability. The integration middleware provides systems integration services and adds to the system with much greater extensibility, scalability and evolveability.

Together, these two models provide a high level "what" and "how" for the airport authority to smoothly plan, procure and manage its airport automation systems integration. They also provide a base for integrators to define their approach to requirement analysis and architecture design. During the maintenance phase of an integration project, this framework governs the smooth evolution of integrated airport automation systems.

2. Systems Integration Information Model

The operations of a large airport are driven by business information. Airport business units create information, transform information, distribute information, and take actions on received information. An integration information model is essential to describe information interchange requirements. This model consists of information elements that describe the inter-operations. An integration control board (ICB) is needed to manage the definition and maintenance of these information elements with a defined process. This model provides a framework for the systems integration requirements management.

2.1 General Description

The integration information model consists of the following information elements that are essential to fully describe the information interchange requirements of an airport:

- Airport business functions/units
- Airport business events
- Event data objects
- Airport business processes

These information elements are captured in both text and diagrams such as business function hierarchy diagram and business process flow diagram and could be viewed at different levels of detail by different users.

The model is developed and maintained by the integration requirements team. This team initially collects requirements and refines the list. It then follows the model throughout the development lifecycle, producing more details as time goes on. In the end, the model could be mapped into an implementation design that follows the architecture model, and becomes the basis to define the specifications for the message types and the integration Application Programming Interface (API).

The model is typically stored in a repository that provides easy and secured access to interested parties, provides views of different levels of detail, and also facilitates the maintenance of the integration information. A Computer Aided System Engineering (CASE) tool can be used to help maintain this repository. This information model will also become the major source for the metadata of the AODB.

This integration information model provides a clear path for the integration requirements specification that helps prevent "spaghetti integration" and ensures that the integration is maintainable and extensible.

2.2 Airport Business Functions/Business Units

This element describes the business area functions of the airport that need to cooperate with each other in some way to accomplish the overall airport operations mission. It also describes the business units that execute the business functions. For example, Baggage Handling is an airport business function, which is executed by the business unit that provides baggage-handling services, using an automated baggage handling system.

In modern airports, the majority of business area functions are computerized. There are usually computer automation systems in each business area function. These application systems produce and/or consume information about airport operations and resource allocations.

An airport operation can be divided into the following three high-level business areas: Airside Operations, Landside Operations and Terminal Operations. These can be further decomposed into more specific business area functions, such as Flight Information Display, Baggage Handling, Cargo Handling, Ramp Control, and Building Management. Those business area functions need to be analyzed to define their integration requirements in terms of what they need from other business area functions and what they should provide to other business area functions. The analysis results would be documented into the integration information model.

2.3 Airport Business Events

A business event is an occurrence or situation that is considered important to the overall operations of the airport. Typically, the event is generated in one business area function, and its result will impact other business area function(s). When information systems are not integrated, the end-user has to gather the event result and pass it to the operators of other interested business units. When integrated, participating systems are notified automatically of an event of interest, and the result of the event is available immediately to those systems. Triggering information sharing on occurrence of business events can deliver significant benefits to the airport: throughput and efficiency can be improved, manual operations and errors can be reduced, and overall responsiveness can be increased.

For example, an arrival delay is a business event that has impact on many other business area functions. This event would likely originate from Air Traffic Control (ATC), and has impact on business area functions such as Flight Information Display, Ramp Handling, Baggage Handling, and Cargo Operations.

Business events can be grouped based on their originating business area functions. The model describes the event, where and how the event is initiated, and what results should come out of this event.

2.3 Airport Event Data Objects

The event data object element describes the information transferred between business area functions when business events occur. A data object is a real or conceptual entity that holds relevant information for a business operation. The data object is described in plain language and further defined by its attributes. For example, data object FLIGHT SCHEDULE is further defined by its attributes FLIGHT NUMBER, SCHEDULED ARRIVAL TIME, SCHEDULED DEPARTURE TIME and AIRPORT ID. When an event occurs, the data object that needs to be passed is described by its identification attributes and the attributes that get new values. The related events should be identified also.

The data objects are typically transferred in messages and may be collected in the airport central database. The Extensible Markup Language (XML) provides a standard for formatting the data object in a message.

2.4 Airport Business Processes

An event activates a business process, which transforms the event and related data objects into new events or data objects. The model thus identifies the stimulating events and data objects involved. The processes described in the model are typically airport-wide and are for integration only. They are usually triggered by flight events or passenger events, such as flight arrival, flight delay, flight cancel and passenger check-in. An airport business function is often a process step in an airport business process. For example, the process triggered by flight delay event would involve a lot of business functions such as flight information display, baggage handling and ramp control. The process diagram gives a clear picture of how the affected airport business area functions inter-operate to handle the occurrence of an airport business event. Many processes involve more than one application, and many applications also have their own internal business processes, which typically perform lower-level functions within a business area.

When a process is triggered by a business event, the involved business systems get notification and data of the event delivered by the integration software components that reside on the database server or the integration application servers.

2.5 Integration Control Board (ICB)

An airport business unit usually represents a business area that is supported by one or more automation system(s). It is very important for all participating airport business units to reach consensus on their inter-operation requirements. A mechanism needs to be established to keep communications between participating organizations flowing and coordinated. The integration information model serves this purpose, by calling for a central organization to facilitate the requirements collection and analysis and to maintain the integration information elements defined in this model. This central organization also needs a formal process to approve the requirements specifications and changes. Without such an organization and process, it is hard to define integration requirements and even harder to maintain these requirements. We call this organization the Integration Control Board (ICB). The ICB usually consists of airport operations domain experts, the airport IT chief and operations manager and sometimes the project technical manager (or chief engineer) of the primary integration contractor.

The functions of the ICB include the following:

- Set up and supervise working groups
- Review and approve integration information requirements.
- Review and approve integration information requirement changes.
- Provide a mechanism for each authorized party (including airport business units, contractors and airport authority) to easily access the integration information model elements at the desired level of detail.
- Provide a mechanism (such as a integration forum) for all interested stakeholders to raise their questions, concerns, and requirements, and to exchange opinions.

The working groups are responsible for integration requirements analysis and specification, repository maintenance, and other daily tasks. Each working group can be formed with subcontractors and other stakeholders to work on subsets of the integration information model and issues in other project phases such as interface specification, interface testing plan and procedures, integration testing, and acceptance testing. Various collaboration capabilities such as e-mail, web pages, newsgroups, and teleconferencing facilitate operations of these working groups.

The ICB and its working groups provide an organizational assurance to prevent the “spaghetti integration” that may otherwise occur.

3. Systems Integration Architecture Model

The airport automation consists of its IT infrastructure and a large number of individual information subsystems that share flight information and other airport operations and resources data among them. It is a system of systems. To avoid chaos in the evolution of such a system, an architecture model becomes important. It provides technical guidance for airport IT architecture planning, and systems integration design and development.

3.1 Requirements of the Systems Integration Architecture

The systems integration architecture model presented is designed to achieve a number of goals: reliability, maintainability, evolveability, extensibility, scalability, and interoperability. These goals help reduce cost in system life cycle while increasing business operations efficiency.

3.1.1 Reliability

Reliability is a concern in all operational systems, but especially so in an airport environment. If a major subsystem (such as Baggage Handling System or Passenger Check-in System) fails, it has major impact on the operations of the airport. This becomes more of an issue in an integrated airport since so many different subsystems are interconnected, there are many more things that could go wrong. Thus, the architecture not only needs to provide reliable central systems (e.g., the central database and application servers), but also ensures that a failure in one subsystem does not interrupt others. For example, if a Flight Information Display fails, it should not stop the Passenger Check-in System.

A reliable system guarantees the success of airport business event publication and delivery since the airport operation is basically a sequence of events triggered one by another. If the sequence stops, this airport operation would stop. To prevent this, it not only requires a highly reliable integrated system, but also requires a contingency plan for manual operations be in place, tested, and practiced.

3.1.2 Maintainability

Maintainability refers to how easily an overall system can be kept operational. For example, if there is a bug in the software of one subsystem, it should be fixed quickly with minimum cost and impact to the business operations and other integrated subsystems. A requirement change should only impact a minimum number of modules. Ideally, only one module should be updated, or even better, only update the data objects needed. The use of Open Systems design principles, such as well-defined interfaces and well-structured design, improve the maintainability of systems.

3.1.3 Evolveability

Evolveability is defined as the ability of an integrated system to adapt to continuous changes in the business requirements and technologies. A hierarchical component-based Open System Architecture that provides well-defined interfaces between each system object in the system hierarchy accomplishes this goal.

The system components in each architecture layer provide independent functional service to the upper layer following international standards. When there is a change in the business needs or technology used in one layer, only related components of the layer need to be changed; all other integrated subsystems and components need not change. If one vendor goes out of business or no longer provides maintenance service, only those obsolete components need to be replaced with components from other vendors; as long as the defined interfaces are maintained, no other system components are affected. A rule-driven or data-driven software design will make the system easier to adapt to business changes.

If there is technology advancement in the IT infrastructure, the old system components may be wrapped or bridged to implement the interfaces defined by the new IT infrastructure or simply be replaced.

Documentation is as important as having a good design. Time and budget should not be the excuse for not updating documents. Keeping documents updated and accessible will make the system evolution much smoother.

3.1.4 Extensibility

Extensibility is defined as the ease of adding new functions to the system, either to support new business functions or just to improve the efficiency of existing business operations. As with evolveability, this concern applies both to individual integrated subsystems, and to the overall system IT infrastructure used for integration itself. For example, in a new airport, only eight subsystems might be integrated in the first phase of the project. Then, another 35 subsystems might be added in the follow-on two phases of the project. The initial architecture of the system integration becomes very important to facilitate easy integration in the follow-on phases. A hierarchical structure and component-based Open Systems approach provides for better extensibility.

3.1.5 Scalability

Scalability refers to the ability to add more systems or components to improve the system performance. For example, an event server facilitates event-based messaging between different application systems. When many more application systems are added to the group originally integrated, the existing event server might not be able to provide satisfactory event services. The bottleneck could be easily resolved by adding more event servers or a larger more powerful event server to the IT infrastructure if it has a multi-tiered distributed system architecture. Use of application servers and a component-based distributed architecture will help.

3.1.6 Interoperability

To achieve a high level of automation, hence high efficiency of business operations, it becomes increasingly important for components in a system to freely exchange data. This is achieved by promoting and following international standards. The IT infrastructure in an airport promotes this interoperability by providing necessary integration services to individual systems. With collaboration among leading vendors and airports, it is possible to standardize those integration services, so that vendors can have those integration interfaces built in their products. With a built-in standard interface, a commercial-off-the-shelf (COTS) product communicates with other systems with little or no customization.

The architecture should utilize as much as possible the centrally managed metadata that are originated from the integration information model. Data standardization becomes possible with integration information model and is vital to achieve interoperability.

3.2 Integration System Architecture Hierarchy Model

A layered architecture is called for to satisfy the six "-bility" requirements. This model consists of hierarchical layers, which in turn consist of independent yet interoperable modules or software components. Each layer provides services to its upper layers. The diagram shown in Figure 1 describes this hierarchical view.

Although the general rule is that the software on each layer should be built on top of its next layer, there are cases in which some software needs to access the services of the layers lower than the adjacent layer. The diagram shows that the Subsystems Layer accesses directly each of the lower layers down to the Network Communications layer; the AODB accesses directly each of the lower layers to Database Network Interface layer. The Network Communications layer is hidden by vendor's Database Network Interface layer, such as Oracle's Net8 (formerly called SQL Net). Each layer is described in the following sections from bottom to top.

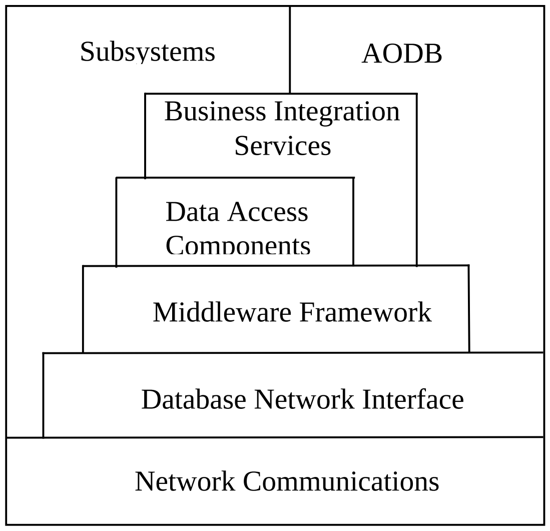

Figure 1. Systems Integration Architecture: A Hierarchical View

3.2.1 Network Communications Layer

This layer provides end-to-end network connectivity. One important characteristic of this layer is that it provides universal connectivity between any two devices on the network. Before the Internet Protocol (IP) allowed universal connectivity (routing), many subsystems were isolated from the rest of the network. Custom gateways were used to connect them, but at reduced flexibility and increased cost. Today, IP can easily provide universal connectivity at a low cost. Above the universal connectivity IP layer, the network provides other communication protocols. Among these are transport protocols such as TCP, UDP, IP Multicast, and session/application protocols such as FTP. The transport protocols allow for such added capabilities as reliable end-to-end data transfer, flow control, and efficient sending of data to multiple recipients. TCP/IP is the most frequently used standard protocol provided by this layer. Application systems access the network communications layer directly if they do not need specific middleware-layer services. Such communication is often used by pre-existing, stand-alone subsystems that do not have automatic data exchange with other systems. Additionally, most subsystems will use this type of direct connectivity internally to communicate with other components of the subsystem. When integration is needed, the application system uses functions of the middleware layer, which in turn uses the Network Communications Layer for its communications.

The Network Communications Layer shields the details of physical connectivity and other lower level protocols for all computing devices in the airport. Obviously, the communication and computer networks (including backbone network) in the airport are hiding behind this layer.

3.2.2. Database Network Interface Layer

This layer supports the client/server database architecture. In networked environments, client applications submit database requests to the server using SQL statements through the database network interface layer. Once received, the server processes the SQL statements and the results are returned to the client application via the database network interface.

The database network interface shields the details of different supporting network communication protocols. The database server and client application developers do not have to be concerned with the supporting network communications. If the underlying protocols change, the database administrator makes some changes on the network configuration file, while the applications require no modification and will continue to function. An example of the database network interface is Oracle's Net8. The services of the Net8 are usually provided through the Oracle Call Interface (OCI).

3.2.3 Middleware Framework Layer

This layer provides the common services required to support component-based software and facilitates information exchange between application subsystems in a distributed network environment. It shields the upper layer from any network concerns and acts as the "glue" that connects the different software components. The database access components and integration components are largely built on top of this middleware layer. There are a variety of middlewares available. This is largely due to the varied set of functions that such a layer is called upon to perform. Among the types of middleware available are:

- Database access middleware: software components, typically provided by database vendors, that provide access to the database from a variety of different operating systems and programming languages. The most popular ones are ODBC and JDBC. Almost all vendors also provide database access APIs in C/C++. Some also provide transaction management and load balancing like TP Monitor.
- Message-Oriented Middleware (MOM): a message-passing service that allows for asynchronous communication between sender and receiver. For example, if the receiver program is not running when the message is sent, the message will be queued until the receiver starts running. MOM capability can be used by publish-and-subscribe services or for direct point-to-point communication.
- Software component framework middleware: software frameworks allow components to run in an orderly manner and to be ported to different platforms and physical process structures. With a component-based development tool, components can be assembled into an application that runs on top of the component framework.

There are three primary standards for the component framework middleware [1]. Distributed Computing Environment (DCE) is the oldest of the three technologies, and it is based on the idea of a "distributed operating system" that allows applications to treat the network as a single computing resource. Distributed Component Object Model (DCOM) is a more recent object-oriented middleware technology that was originally derived from DCE and still shares many features with it. DCOM is provided by Microsoft and is specifically oriented towards Microsoft operating systems such as Windows NT. Common Object Request Broker Architecture (CORBA) is an object-oriented middleware technology that is strongly supported by the vendor communities as an alternative to DCOM. There are vendor proprietary middleware standards also, but most middleware products are now either starting to or already have interfaces or “bridges” to CORBA and DCOM.

3.2.4 Data Access Component Layer

A component in the architecture is a software module that performs a well-defined business function(s) or system function(s), and is replaceable without any impact to other components. That is, it has well-defined interfaces and can thus be replaced with another component implementation that maintains the same interfaces. Often, these components are designed so that they can be reused at more than one physical location in the system. The application builder would select and “glue” the components together to form a final system that could be used directly by the end users. These components could be integrated through vendor-provided API’s.

As the component-based application development approach becomes mainstream, data access functions have been extracted out of the mixed application software modules. They became components that are called by business components needing data from a database or files, thereby shielding data access details from the developers of the business components. With the support of the middleware framework, business components and data access components can be deployed anywhere within the framework, and can be replaced by another vendor’s components with the same external interfaces. These data access components usually map directly to the data model of the AODB. They sometimes also include aggregated data objects or other derived data elements based on business requirements. Sometimes, these components also implement complex integrity rules and security functions.

3.2.5 Business Integration Service Layer

The business integration service layer provides software components with APIs specially required by the integration. They are supported directly by the data access components and/or the middleware framework. They provide high level services such as event services, security services, directory services, logging services and transaction services with the benefit of shielding the details of the complex middleware APIs from each individual applications. Components of this layer also contain the overall business rules of the airport operations. For example, when a passenger checks in, the business rules of the airport dictate that a certain number of operations be performed. By implementing these business rules in these reusable components, one not only successfully reuses code, but also implements business rules in a consistent way across airport systems. This is very important to ensure that the integrated airport operates correctly and efficiently.

Although the integration service components could be deployed on the same physical machine together with AODB, it is desirable that they be deployed on application server(s) for greater scalability, performance and reliability.

The major difference between this integration architecture and other system architectures lies in the business integration service layer that is implemented as integration middleware components and provides business integration services via API's. Although there are industry standards for most of the lower layers, there is no recognized standard for these integration services yet. The typical products on the market that fall into this layer are the integration broker suites, which provide a valuable, central organizing “backbone” for enterprise application portfolios [2].

3.2.6 AODB Layer

This layer provides centralized data management services for airport operational and historical data. It provides a central data repository and tools to access and maintain the database. A centrally managed database either physically or virtually provides the benefits of high data quality, reduced resource consumption and more efficient system and business operations. Architecturally, the airport central operational database and the data warehouse belong to this layer.

The AODB Layer provides two types of services: the data service for airport operational and historical data, and the metadata service that defines and helps the use of the data service.

There are typically at least two physical databases installed in an integrated airport: an operational database and a data warehouse. The operational databases are typically used for on-line transaction processing and hold only the data that is currently needed or will be needed in the near future. The data warehouse typically holds all the historical data for the operations and supports decision making and analysis functions. Some differences between the data warehouse and operational database are:

- The quantity of data stored: only current and near future data are stored in the operational database while all historical data are stored in the data warehouse
- The level of reliability: since the operational database is used by all integrated applications, it must have higher reliability (e.g. by using a redundant hot stand-by) than the data warehouse
- The usage pattern: the operational database has a lot of small transaction processing occurring continuously and concurrently, whereas the data warehouse has fewer, more processing-intensive queries

The metadata defines the format and constraints on data stored in the central database. For example, in a relational database management system (RDBMS), the metadata includes, among other information:

- The tables that will be stored in the database
- The attributes (also called columns or fields) in each table and the type of data that may be stored in each attribute
- Physical constraints on the database such as the maximum amount of data that can be stored or the maximum number of characters that can be stored in a given attribute
- Security constraints, usually taking the form of access control lists of which users can have access to which table, and what kind of access (create, read, update, etc.) a given user has to a given table
- Integrity and referential constraints on the database, such as a constraint that a given attribute cannot be empty, or that the only legal values of one attribute in one table are found in another table

The metadata could also include what information will be shared between the integrated subsystems and how that information might be shared. The metadata typically has information in a number of different categories of airport data:

- Dynamic/operational data that is updated constantly such as the status of actual flights for the day and the status of airport systems
- Schedule data, such as future scheduled flights
- Resource data, such as stands/gates, check-in counters and carousels
- Reference data, such IATA aircraft types, IATA airline codes and IATA airport codes

3.2.7 Subsystems Layer

The subsystems layer is at the top of the architecture hierarchy. The subsystems are the airport applications that support the day-to-day functions of the airport operations. These systems are independent systems, although they often use the same flight information and airport resource information. They can get that information from different channels, which may result in inconsistent data between different systems. With AODB providing central airport information dissemination and quality control, data inconsistency across the airport is minimized. Each subsystem can even resynchronize its local data with the AODB on specified time intervals. Airport subsystems frequently correspond to business functions and may include Flight Information Display Systems, Baggage Handling System, Gate Management System, Building Management System, and Security Management System.

In addition to specific subsystems that are integrated in an airport, system management and control functions are vital. It is a challenge for big enterprises to successfully manage increasingly complex distributed application systems that carry "mission critical" information and support core business processes. An integrated system management function is essential for keeping these distributed systems working properly. The more complex and integrated the airport becomes, the more important system management is to the smooth operations of the airport.

The system management function sits “on top” of all the other software and hardware, monitoring their operations status and allowing for remote control. In this layered architecture, the system management system could be treated just as another subsystem, but it needs to be able to access system objects in every layer.

It is interesting to observe that the AODB sits at the same level as other subsystems. This symmetrical architecture increases system flexibility. The decision on how to manage the source information is made in the Business Integration Services Layer. Ideally, the business integration services are configurable with an integration service management tool that could be considered another subsystem in the architecture.

4. Summary

Large-scale airport systems integration is a very complex and difficult task. Without a well-planned approach and a framework for guidance, if the project itself does not fail, its follow-on maintenance could be unsupportable.

To understand and follow a good framework is the first and the most important step in approaching large-scale systems integration. This framework includes two models that provide guidance on managing the integration requirements and designing the integration system architecture respectively.

The integration information model provides guidance on managing the integration requirements. There are different ways to implement this model depending on the schedule and budget. When resources are limited, the interface requirements document, sometimes called the airport business integration guide, suffices to record the requirements information. However, an ideal implementation would establish a central automated repository for the collection and distribution of the integration requirements information. A formal notation, such as UML, should be encouraged in the development of the integration model.

The integration architecture model provides guidance on designing the integration system architecture. It is a multi-layer model with subsystems and AODB on top, middleware components below, and the database network interface and network communications on the bottom. The major difference of this architecture model from other system architectures lies in the integration middleware that could be implemented as integration components and provide business integration services via API's. The integration service API's should be standardized and used by all the vendors to improve productivity and interoperability.

Together, these two models provide a high level "what" and "how" for the airport authority to smoothly plan, procure and manage its airport information and communication systems. They also provide a base for integration contractors to define their approach to requirements analysis and architecture design.

Acknowledgements

The author gratefully acknowledges Catherine N. Bolczak and Ronald G. Rhoades of the MITRE Corporation for their constructive comments, and An-ping Hou and Gary Bisaga for their constructive inputs.

References

[1] International Systems Group, Inc., 1997, *Middleware – The Essential Component for Enterprise Client/Server Applications*, Part Number: CE-Z7970-93

[2] R. Schulte, R. Altman, 1 September 2000, *Application Integration Middleware Market,* Strategic Analysis Report, GartnerGroup, R-11-5113, pp.23