SENG 443
Project 2

Component Architecture
Air Traffic Control
(ATC)

Authors:

Hanif Mohamed
Aarti Punj
Nitin Puri
Cledelyn Santos
Kamal Singh

University of Calgary
Department of Computer Science
Faculty of Science
Calgary, Alberta, Canada
December 2002

Version: 1.0

nitin@cpsc.ucalgary.ca

TABLE OF CONTENTS

TABLE OF FIGURES

1. EXECUTIVE SUMMARY

This document provides the Component Architecture of the Air Traffic Control (ATC) system. There are four major sections in the document to provide a clear and detailed technical description of the Component Architecture of the ATC. These four sections are the Conceptual View, Module View, Execution View, and Code View.

The Conceptual View section outlines how the ATC is related to the overall system. Within the Global Analysis subsection, we have included a factor table, which lists product, organizational, and technological factors. In the Conceptual Configuration subsection, we have illustrated how the Overall System Architecture and ATC are related.

The Module View section examines our modules and subsystems contained within the ATC component. Within our Global Analysis subsection, we have used a factor table, which lists the product, organizational, and technological factors. In the Module Configuration subsection, we have described the implementation of design patterns, such as the Bridge Pattern, Broker Pattern, Factory Method Pattern, and Proxy Pattern, within our modules.

The Execution View section examines execution of the modules and subsystems in terms of system performance in regards to hardware/software requirements. Within our Global Analysis subsection, we have used a factor table, which lists the product, organizational, and technological factors. Within our Central Design Tasks subsection, we have also examined the execution of the modules and subsystems in terms of Execution View, Communication Paths, Configuration, and Resource Allocation, in their own respective subsections.

The Code View section describes how the different modules and subsystems will be hierarchically stored. Within our Global Analysis subsection, we have used a factor table, which lists the product, organizational, and technological factors. In the Directory Hierarchy subsection, we have described how the files and folders should be setup in for optimal hardware and software performance. Furthermore, we have described how the how the Intermediate Build Package and Executable Software Package will be used in the development of the ATC, in their respective subsections.

In order to provide a clearer understanding of terms used in this document, a Glossary is included at the end of this document.

2. CONCEPTUAL VIEW

This section will outline how the ATC is related to the overall system. Within the Global Analysis subsection, we have included a factor table, which lists product, organizational, and technological factors. In the Conceptual Configuration subsection, we have illustrated how the Overall System Architecture and ATC are related.

2.1 Global Analysis

The purpose of this section is to discuss the different factors that affect the Conceptual View of the ATC architecture of the system and defines strategies needed in order to accommodate these in the design. A real-time, event driven architecture design is used.

2.1.1 Factor Table

The following factor table discusses the Product, Organizational and Technological factors that make up the Global Analysis of the Conceptual View.

Product Factor	Flexibility and Changeability	Impact
P1: Functional Features		
P1.1: High Concurrency		
The system must monitor many entities and flights.	The number of planes and entities monitored are changeable. The system must remain highly concurrent (non-negotiable)	There is a moderate impact on the real-time performance of the system.
P1.2: Very Dynamic		
The monitoring of the aircrafts leaving and entering the airspace will change constantly.	The number of planes and entities monitored are changeable. The system must remain highly dynamic (non-negotiable).	The more planes handled, the more strain on available resources. The system must be able to handle this. There is a moderate impact on the real-time performance of the system.
P1.3: Asynchronous, Event-Driven System		
Messages between entities should be sent as events occur. The exception to this method of handling is during a conflict situation. Conflict messages should be prioritized and handled immediately.	This is not negotiable.	Lack of proper event handling could result in loss of planes and lives. There is a moderate impact on the real-time performance of the system.
P1.3: Configurable		
The system has to incorporate commercially developed software components and hardware and upgrade easily.	Types of compatible software and hardware may be negotiable.	This has a moderate effect on real-time performance, and user interface. There is an effect on modifiability.

P2: User Interface		
P2.1: Ease Of Use		
The user should be able to switch between views easily, in a windowed environment.	This is flexible. The types of views and style of windows is negotiable.	There is a moderate impact on the user's ability to handle events efficiency. This affects real-time performance.
P2.2: Transparency		
The windows representing events should be transparent to avoid potentially crucial data from obscured.	This is not flexible.	Lost or obscured data could result in loss of planes and lives. There is an impact on security/safety.
P3: Performance		
P3.1 Acquisition Performance		
Events must be handled in real-time. We must have continuous monitoring of planes.	This is non-negotiable.	Delays in the handling of events could result in loss of planes and lives. There is an impact on real-time performance.
P3.2 Recovery Time		
Recovery time must be extremely fast. The backup system is responsible for overtaking the primary communication network during recovery if the Host fails.	This is non-negotiable.	Delays in recovery could result in the loss of important and crucial data. There is an impact on availability.
P4: Dependability		
P4.1: Ultra-High Availability		
The system must be constantly available. I.e. there should be less than 5 minutes per year downtime.	This is non-negotiable. The system is prohibited from being inoperable.	Downtime could result in loss of important and crucial data, resulting in loss of planes and lives. There is an impact on availability.
P4.1: Reliability		
Messages passed between entities must be handled without any loss of data.	This is non-negotiable.	Loss of important messages could result in loss of planes and lives. There is an impact on reliability.
P4.2: Safety		
Safety provisions include transparent messaging to ensure important data is not ignored, as well as network related security to prevent any un-authorized entry into the system.	This is non-negotiable.	Data could be obscured, and safety of the entire ATC System could be at stake if the network security is compromised. (e.g. Terrorist attacks) There is an impact on safety.
P5: Failure Detection, Reporting and Recovery		
P5.1: Error Classification		
The system must be able to quickly and efficiently classify errors in detail so they may be handled quickly.	This is non-negotiable.	Large numbers of unclassified errors may result in unacceptable downtime of the system. There is an impact on reliability and availability.
P5.2: Error Logging		

Detailed error reports should be generated for all occurrences of errors to facilitate classification and handling.	This is somewhat negotiable. The detailing of the reports is variable.	Inefficient logging of errors can make it difficult to classify errors. The time and ability to effectively handle recurring errors is affected by the proper recording of errors. There is an impact on reliability.
P5.3: Diagnostics		
The system must be able to identify potentially crucial errors early.	This is somewhat negotiable. It is impossible to predict all errors, however the system should be able to predict to the fullest extent possible.	The proper diagnosis will help to avoid system crashes and virus type attacks. There is an impact on reliability and availability.
P5.4: Recovery		
Errors and bugs should be quickly quarantined to avoid further system damage. Recovery must be complete.	This is not negotiable.	This affects the functionality of the system and may result in unacceptable system downtime. There is an impact on availability.

P6: Service		
P6.1: Software Installation and Upgrade		
Openness of the system is important to allow for future software and hardware upgrades to be incorporated into the system.	This is somewhat negotiable. Basic hardware and software components must remain, but can be enhanced.	This affects acquisition performance and the quality of the user interface. There is an impact on modifiability.
P6.2: Software Testing		
Extensive testing must be performed throughout the system lifecycle.	This is not negotiable. Testing has to occur frequently to identify problems as early as possible.	Lack of testing may affect system availability resulting in the loss of crucial data. There is an impact on reliability.
P6.3: Maintenance of System		
Extensive system support services should be available to ensure the system exhibits ultrahigh availability.	This is somewhat negotiable. The type of system support may be variable.	This may affect system availability reliability.

Technological Factor	**Flexibility and Changeability**	**Impact**
T1: General-Purpose Hardware		
T1.1: Processors		
The fastest distributed multi-processors available should be used to ensure high concurrency. The recommended processor is the IBM RS/6000 series processor.	This is non-negotiable	This has an effect on real-time performance.
T1.2: Network		
Completely secure and fast network connections should be used. The network used must be able to handle large communication loads without any loss of data.	This is non-negotiable.	This has an effect on safety and real-time performance.
T1.3: Memory/Disk		

Large amount of memory should be available to store logs of messages for later playback.	This is negotiable. Much of the data is transferred to CD format and stored in libraries.	This has an effect on the quality and detail of message/error logging. This affects safety.
T2: Domain-Specific Hardware		
T2.1: Specialized Hardware		
16-40 radar devices per facility are needed to track the actual flight movements are needed. Specialized consoles able to track aircraft are needed.	This is non-negotiable.	This has an effect on the quality of aircraft monitoring. This affects real-time performance.
T2.2: Specialized Network		
The network for the system should be able to control from 400-2440 separate aircraft tracks simultaneously.	This is negotiable. The network may require expansion to support heavier air traffic.	This has an effect on real-time performance.
T3: Software Technology		
T3.1: Operating System		
The operating system used for the system must be extremely secure. Unix is the OS of choice. It has far less occurrences of crashing than Windows.	This is somewhat negotiable. Other secure operating systems such as Linux or may be considered.	This has an effect on system availability.
T3.2: Implementation Language		
Real-Time Concurrent Ada-95 is the language chosen for the system. It is a highly powerful; secure language that can be easily documented. Ada 95 facilitates high concurrency (a requirement) more comprehensively than any other language.	This is somewhat negotiable. Other real-time languages such as Erlang and Real-Time Java.	This affects concurrency and system availability.
T3.3: Design Patterns		
The patterns used should be supportive of real-time application, such as Factory and Bridge.	This is negotiable. Other patterns may also be used to support the real-time based patterns.	This affects the overall system design, and real-time performance.
T4: Architecture Technology		
T4.1: Architecture Styles		
The system exhibits an independent components style and has both event systems and communicating processes. In addition the system should also have some layering as it aids in modifiability.	This is somewhat negotiable. Other styles complementing event driven (real-time) systems may be used.	This will have an effect on the communication of messages/events and real-time performance of the system. This also affects modifiability.
T4.2: Architecture Patterns and Frameworks		
The architecture patterns to be used include the Proxy and Broker patterns.	This is negotiable. Other patterns may also be used to support the real-time based patterns.	This affects the architectural design, and real-time performance.
T5: Standards		

T5.1: Database		
The ATC Host is a large database containing information such as flight plans, and actual flights. This database must be highly expandable and reliable.	This is somewhat negotiable as the type of database can be chosen to fulfill requirements.	This may have an effect on the reliability of the system and the storage of important information.
T5.2: Communication		
An efficient communication network must be in place due to the emphasis on high-speed message passing between entities. The network speed should remain constant and should not vary in response to increased message passing.	The type of communication network is negotiable as long as it fulfills the requirements specified for this system.	This may have an effect on the real-time performance.
T5.3: Algorithms and Techniques		
Efficient algorithms and techniques that are highly object oriented (i.e. supportive of data encapsulation) should be used to ensure high speed.	This is negotiable.	This will have an effect on system performance and availability.

Organizational Factor	**Flexibility and Changeability**	**Impact**
O1: Management		
O1.1: Build vs. Buy		
The system architecture must be built, whereas components such as processors and monitors can be bought. The system should be built to allow for modifiability.	This is negotiable and flexible. More components can be built or added if necessary.	This affects the release schedule and reliability and modifiability.
O1.2: Schedule vs. Functionality		
Functionality of the system must be completed at deployment.	This is not negotiable. A partially complete system is not acceptable.	This will affect the overall performance as well as safety.
T2: Staff Skill, Interests, Strengths, and Weaknesses		
O2.1: Application Domain		
The staff should have extensive knowledge in real-time languages and operating systems. They should write quality, secure, highly testable code.	This is non-negotiable. For a system as mission-critical as this, staff with extensive knowledge is essential.	This affects system availability and safety.
O2.2: Software Design		
A proper software engineering approach must be used in the system design. The waterfall life cycle model may be applied.	This is somewhat negotiable as long as the prototyping model is not used. This is because partially functional features cannot be used or properly tested in this type of environment.	This affects the development/release schedule and the testing process. This affects reliability.
O3: Process and Development Environment		
O3.1: Configuration Management and Tools		

There must be configuration management to allow for modifiability of the system. Stubs may be used for later expansion of the system, and extensive documentation is important to help future developers.	This is somewhat negotiable because the type of configuration management can be chosen.	This may affect the development schedule and modifiability.
O3.2: Testing Process and Tools		
Integration, Black-box, White-box, Bottom-up and Top-down testing of each module and the entire system must be done.	This is non-negotiable. Extensive testing must be done to ensure extremely high uptime.	This affects system availability and safety and performance.
O4: Development Schedule		
O4.1: Delivery of Features		
All features required for the system to operate fully must be delivered upon deployment. Software packages, such as packages used to enhance views, can be released on later dates.	This is non-negotiable. The system must be fully functional if used in an Air Traffic environment. A partially functioning system is not acceptable.	This affects safety.
O5: Development Budget		
O5.1: Head Count		
Lower numbers of highly trained staff should be used. Smaller teams would ensure there is no possibility of a lack of quality as all work can be checked. There should be extensive management of these team members. There should also be a broad range of staff skills, i.e. from system design to networking.	This is somewhat negotiable. Further team members can be added at the beginning, but should not be added late in the project as it can result in lack of communication in the team.	This affects the quality of the system and could possibly affect safety and performance.
O5.2: Cost of Development Tools		
The budget should be extremely high as this is a mission-critical system. Only the best tools and staff available should be used to ensure the system will not fail.	This is non-negotiable. Less expensive components should not be used at the expense of system quality.	This affects safety and performance. Lack of quality system components could result in the loss of planes and lives.

Reliability
Issue: Data retention is essential for the ATC system. Any loss of data, such as the loss of a flight plan, would be unacceptable and therefore the system must be reliable. Backup systems must be in place to ensure dependability and allow for quick recovery time. The network must also be reliable in terms of speed of transfer for real-time messages. **Influencing Factors from the Factor Table:** **Product Factors – 4.1, 5.1, 5.2, 5.3, 6.2, 6.3** Correct data must be provided at all times to communicating entities. Minimizing the possibility of errors or system crashing is crucial. **Technological Factors – 1.3, 5.1**

The system must be able to efficiently record and store data for later use.

Organizational Factors – 1.1, 2.2

Proper engineering approach should be used to ensure a reliable system.

Solution: Design a strong, robust system that is easy to test and uses highly reliable components. Extensive testing is done frequently. Error logging and classification is essential to lower the time needed to recover in the event of system failure.

Strategy: A layered design should be used for the system architecture.

Strategy: Employ network management for extensive monitoring and control information.

Strategy: A reduced backup capability should be in place for the ATC host database.

Strategy: A policy for the handling and logging of errors should be in place.

Security/Safety
Issue: Security of the data contained in message passing is essential in Air Traffic Control systems. Any unauthorized access to information regarding flights may result in malicious attacks. Safety issues include the obscuring or distorting of data. **Influencing Factors from the Factor Table:** **Product Factors – 2.2, 4.2** Preventive features of the system are needed to ensure crucial data is not obscured or lost. **Technological Factors – 1.2** The network must be secure against unauthorized access, and also be able to handle large amounts of data. **Organizational Factors – 1.2, 3.2, 4.1, 5.1, 5.2** Staff skills in real-time systems are essential. The system provided upon deployment must fully functional.
Solution: Design a robust system that protects that data passed through a network through methods such as encryption. Use alarms and GUI transparency to make important messages appear obvious. *Strategy:* Data passed along the network should be heavily encrypted to avoid unauthorized viewing/use of data. *Strategy:* Provide widowing GUI facilities, with transparent messages to avoid the loss or obscurity of important data. The air traffic controller should not be able to miss important messages. Conflict alert messages should also be prioritized, and thus made visually identifiable, over regular messages. *Strategy:* Provide recording capability for later playback, to ensure availability of information when needed.

Real-Time Performance
Issue: Real-time performance is extremely important to ensure flights are handled promptly. Any lag in flight handling, especially in an emergency situation, could result in

disaster if not handled immediately. To obtain real-time performance, the system must be operating very efficiently. This requires the system to handle large numbers of aircraft (i.e. 400-2440 planes) simultaneously. Networks must be able to carry the communication loads and the software must be able to develop computations quickly and precisely.

Influencing Factors from the Factor Table:

Product Factors – 1.1, 1.2, 1.3, 2.1, 3.1

The system must be able to provide real-time data and handle asynchronous events efficiently.

Technological Factors – 1.1, 1.2, 2.1, 2.2, 3.2, 3.3, 4.1, 4.2,5.2

Real-time capability of the operating system (Unix), hardware (IBM RS/6000 series) and the overall design of the system, as well as the implementing language (Ada 95) are essential.

Organizational Factors – 2.1

The staff skills in real-time systems are essential.

Solution: Use networks and components capable of handling large loads of messages. Design the system to handle any type of error effectively without the possibility of complete system failure.

Strategy: Client-Server architecture should be used to promote modularity and cohesion, as well as reduce coupling.

Strategy: A distributed multiprocessor should be used to improve communication between entities.

Strategy: Design and Architecture patterns supportive of event-driven systems should be used when designing the system.

Modifiability

Issue: The system must exhibit openness so that it may be easily upgradeable and expandable. If the system is not modifiable it will quickly become obsolete and will not be able to incorporate commercially developed software components (which are required). The rate at which technology has been advancing, as well as the growth of the Air Traffic Control system must be handled.

Influencing Factors from the Factor Table:

Product Factors – 1.3, 6.1

The system must be able to upgrade and expand easily.

Technological Factors – 4.1

Call and Return architectural style, specifically layering, should be adhered to.

Organizational Factors – 1.1, 3.1

Configuration management and building the system are important to allow for modifiability of the system.

Solution: Use a layered approach to designing the system. Provide utilities designed to make hardware and software upgrades faster.

Strategy: The open-system concept should be used to facilitate upgrading and expansion.

Strategy: Dependencies between modules should be minimized, and coupling should be reduced wherever possible to aid in quick upgrading. A change made in one area shouldn't necessarily require changes throughout the rest of the system.

Availability
Issue: As Air Traffic Control systems are mission-critical, it could mean disaster if the system was down for an extended period of time. Therefore, the system must exhibit ultra-high availability, which means less than 5 minutes per year should be unavailable. **Influencing Factors from the Factor Table:** **Product Factors – 4.1, 6.2, 6.3** Ultrahigh availability is required due to mission critical nature of the ATC system and the system components should support this. **Technological Factors – none** **Organizational Factors – 2.1, 3.2** Proper testing and knowledgeable staff are required to ensure the system has extremely low downtime if any.
Solution: Provide for extensive system checks throughout the lifecycle of the ATC. Keep logs of errors in a debugging utility to make recovery more efficient. *Strategy:* A policy for testing throughout the life cycle of the system should be in place to ensure ultrahigh availability. *Strategy:* Secure operating systems (Unix) should be used to lower the probability of crashing and failure.

2.2 Conceptual Configuration

2.2.1 Overall System Architecture

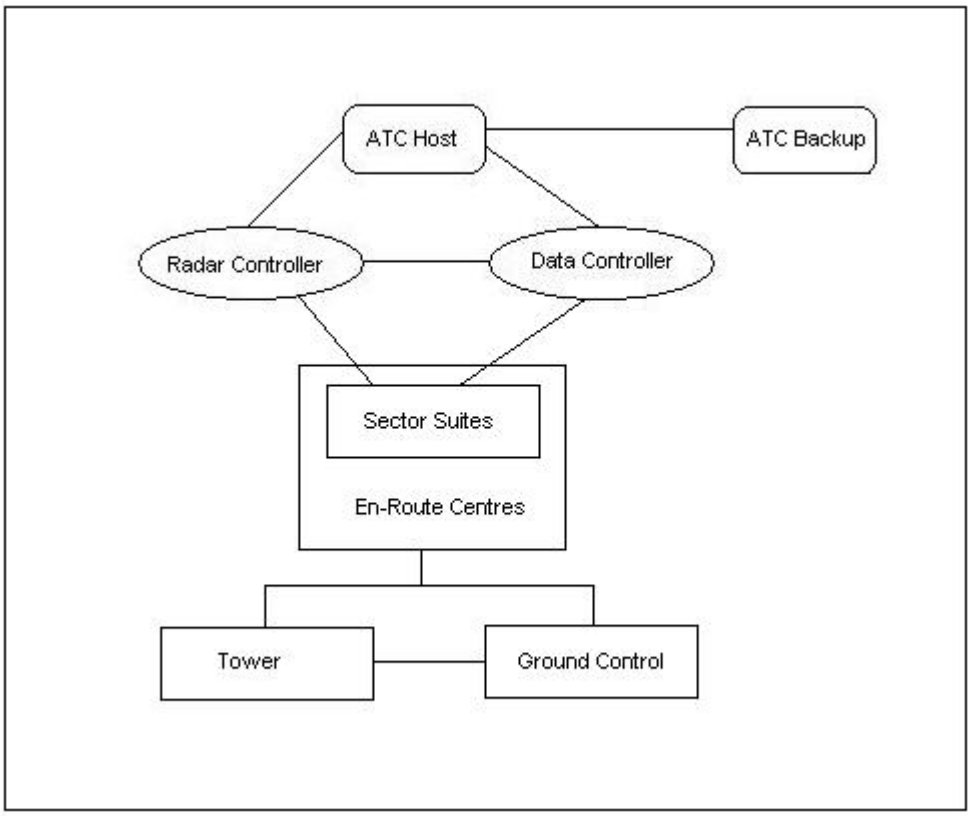

Figure 1: Overall Conceptual View

The overall ATC system is divided into a number of entities. These entities interact with each other in order to assure safety in the airport and accuracy as well as timeliness of data. These entities are assigned responsibilities that are critical to the operation of the system.

The Ground Control guides an aircraft on the ground while the Tower monitor the aircraft's position within the terminal control area. The Ground Control and the Tower can communicate messages to each other.

The En-Route Centers are responsible for monitoring sections of air space. Within each ERC reside Sector Suites. Sector Suites contain the Data and Radar Controllers, who communicate data/information to the ATC Host. They can also communicate with each other (through verbal or non-verbal forms of communication). Our Host is connected to a backup system (details on this will be explained later in the document), which is used in case the Host fails.

2.2.2 Data Flow Components and Connectors

The flow of data between the Radar Controller, the Data Controller, and the Host can best be illustrated by use of the following diagram:

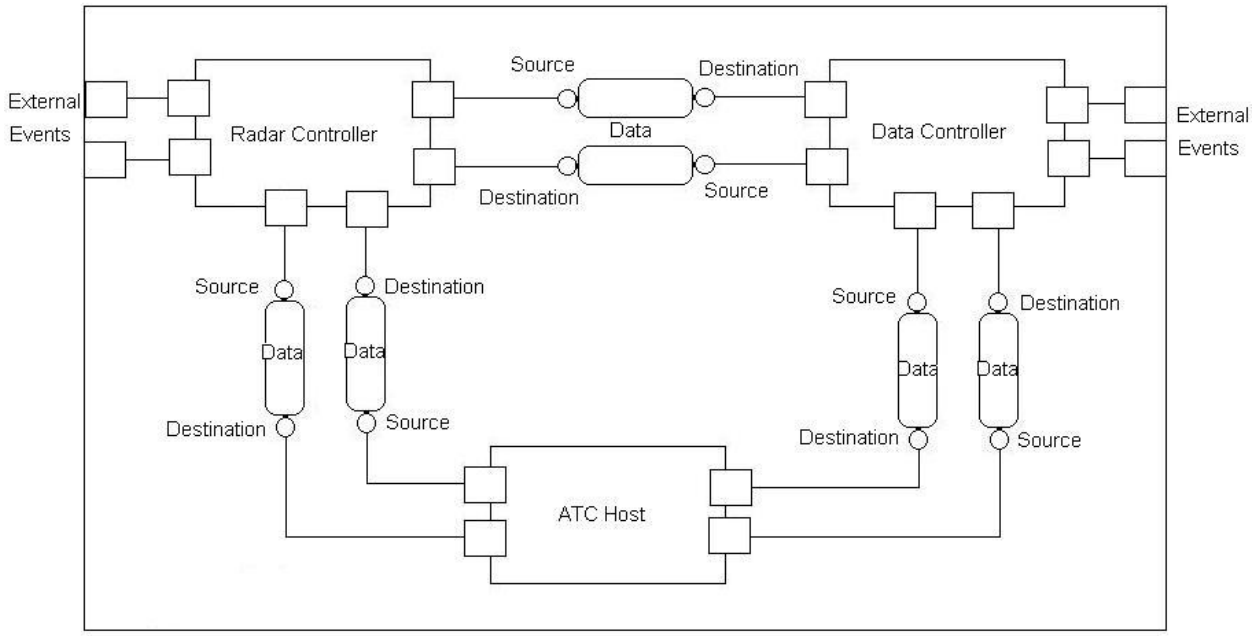

Figure 2: Data Flow Components and Connectors View

The Radar and Data Controllers receive data as input from external sources (such as the plane transponder, or the Data Controller can receive data from the Radar Controller, etc). The Controllers act as a source and destination of data to and from the ATC Host (and the Host acts as a source/destination of data as well). As discussed previously, they can also send messages to and from each other. A more detailed explanation of this interaction is depicted in the following sequence diagram:

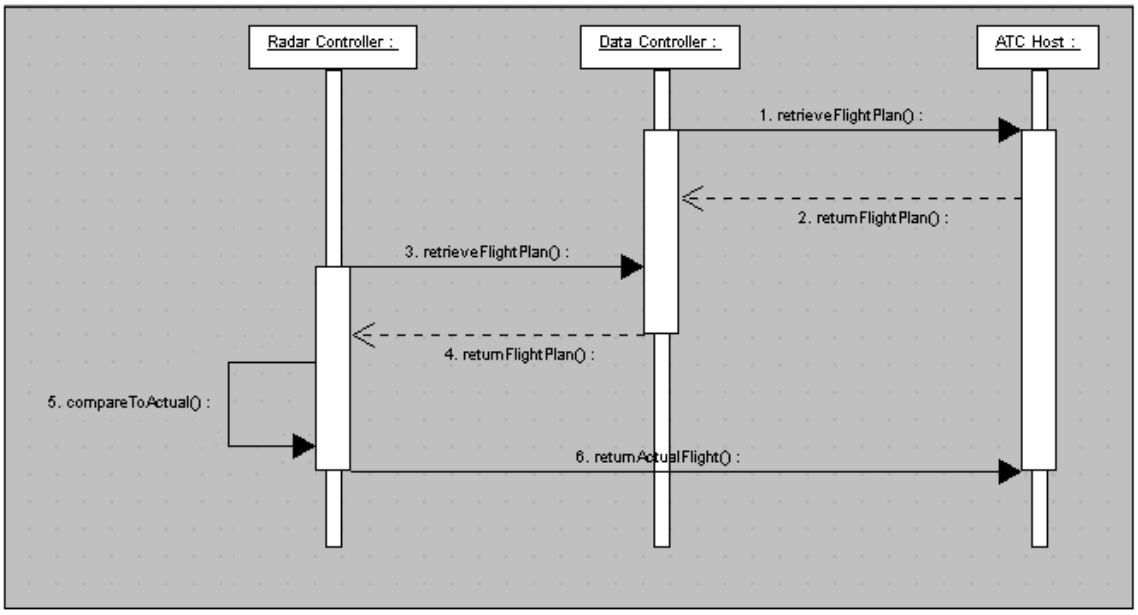

Figure 3: Interaction Sequence Diagram

The sequence diagram above shows the behavioural relationship between the ATC Host and the separate controllers within a sector suite. The ATC Host contains all of the updated flight plans for its service area. The Data Controller is responsible for obtaining the flight plan data in order to determine where the plane is supposed to be and where it will be going. The Data Controller obtains the flight plans from the ATC Host and will send this information to the Radar Controller. The Radar Controller determines the actual flight information through radar data retrieval, and compares this to the flight plan it receives from the Data Controller. If there are any anomalies from the flight plan, the Radar Controller will communicate with the aircraft to determine the reason for the anomaly. The Radar Controller will then send a message to the ATC Host so that this information is updated.

This interaction occurs in every sector suite within an En-Route Center airspace. This information is required in order to guide aircraft through its sector suite.

2.2.3 Main Components and Connectors

There are two main components in the ATC System that interact with the Host. The DataControllerPresentation component is divided into two components: DataModel and DataDisplay. The DataModel component takes the FlightPlans and updates them, which in turn updates the DataDisplay component allowing the updated display of flight plans.

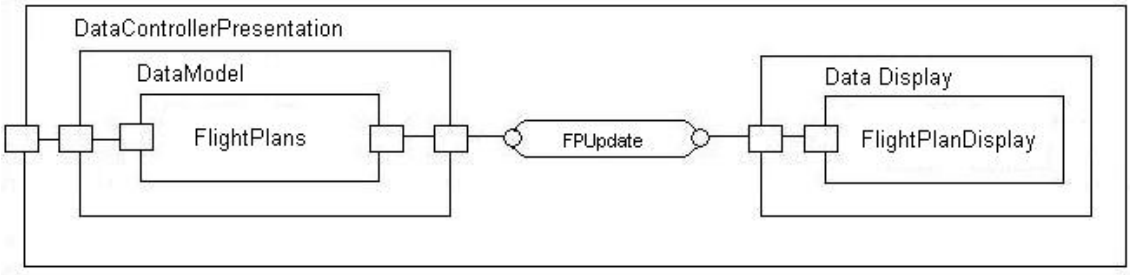

Figure 4: DataControllerPresentation in the ATC System

The RadarControllerPresentation component is also divided into two main components: RadarModel and RadarDisplay. The RadarModel component has three subgroups: RadarSurveillanceData, AlarmStatus and AircraftCommunication. Each of these gets updated which in turn updates the RadarDisplay components. For instance, any changes in the RadarSurveillance Data gets updated and passed onto the LocationDisplay (which contains the speed and the altitude of the aircraft). When an Alarm Status is updated, the Alarm Display in the consoles also gets updated. Any change in the Aircraft Communication gets relayed onto the Communication Display panel.

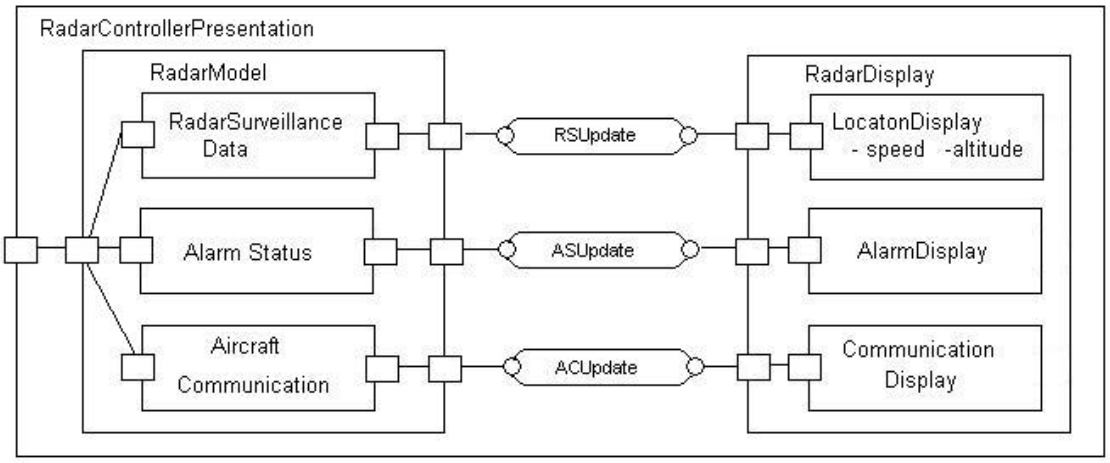

Figure 5: RadarControllerPresentation in the ATC System

2.3 Backup System

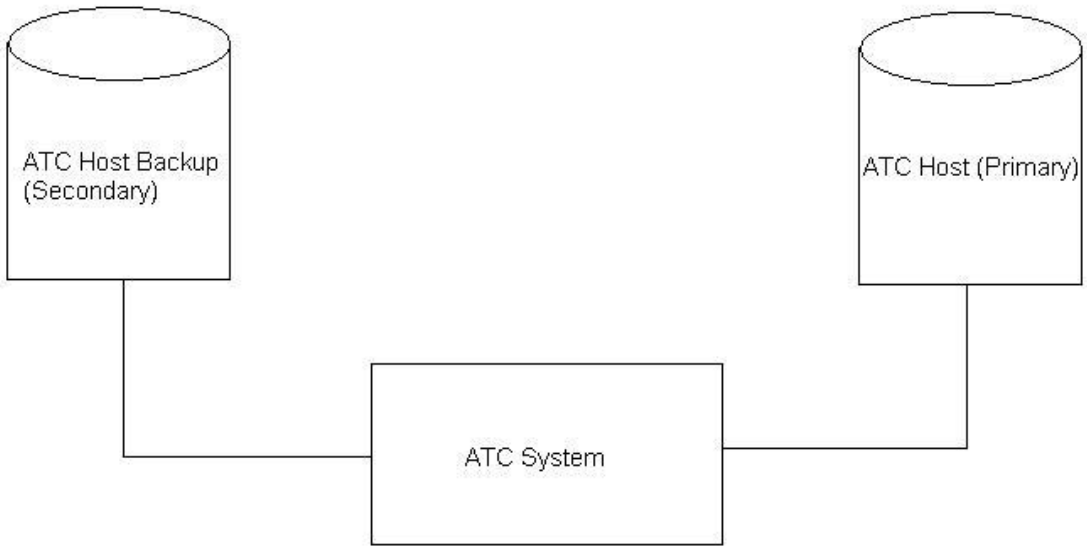

Figure 6: ATC Backup System

The backup method the ATC system will employ consists of a single backup that is parallel to the ATC host (only one backup is used as the use of additional backups will likely only end up wasting space), which is the primary method of storage. This basically means that the ATC system will update both the ATC Host and the backup at the same time, as opposed to the other method of backing up information in which the ATC Host updates the backup after the ATC system updates the ATC Host.

The drawbacks of using the method of backing up that is shown above, is that it is slower than the other method of backup. This is because every time the system backs itself up, it must update two separate storages as opposed to just one. However, when considering a real-time system in which many lives are at stake, this is a small price to pay when we consider what could happen if the ATC system were to use the other backup method, in which the ATC Host updates the backup. Imagine if the system were to crash at a time right before the ATC Host was able to update the backup. The consequences of such an event could be disastrous, as the backup would not know the current state of the system, and this could result in the loss of many lives. In our opinion, it is much better to have a system that is a little slower, but is less likely to result in the loss of lives.

It is also important to note that the backup for the system will not store all of the information that is stored in the ATC Host. Instead, it will store only vital information necessary to the functioning of the system. The backup is not designed to be able to handle the running of the system for extended periods of time, because if it were, it would just be another ATC Host, and this would end up wasting space, and be rather costly.

3. MODULE VIEW

This section examines our modules and subsystems contained within the ATC component. Within our Global Analysis subsection, we have used a factor table, which lists the product, organizational, and technological factors. In the Module Configuration subsection, we have described the implementation of design patterns, such as the Bridge Pattern, Broker Pattern, Factory Method Pattern, and Proxy Pattern, with our modules.

3.1 Global Analysis

The purpose of this section is to discuss the different factors that affect the Module View of the ATC architecture of the system and defines strategies needed in order to accommodate these in the design. The concepts of software reuse and modularity, in conjunction with industry standard design patterns, are used.

3.1.1 Factor Table

The following factor table discusses the Product, Organizational and Technological factors that make up the Global Analysis of the Module View.

Product Factors	**Flexibility and Changeability**	**Impact**
P1: Functional Features		
P1.1: Module Dependencies		
Using highly object-oriented design, and adhering to information hiding and encapsulation principles can minimize module dependencies. Each module should have a specialized task independent from other modules.	This is somewhat negotiable, however interdependencies are discouraged.	The speed of the message passing is affected. (The more information needs to be passed, the slower the response).
P1.2: Reuse of Models and Subsystems		
Designing highly maintainable code can maximize reuse of models and subsystems. If the code cannot be easily changed to suit a new purpose, it is not likely to be reused. Libraries may also be created to store commonly used functions and classes.	The amount of reusability is negotiable. Some specialized functionality may not be made easily reusable.	There is a moderate affect on meeting the development schedule.
Organizational Factors	**Flexibility and Changeability**	**Impact**
O1: Staff Skills		
The staff should have extensive training or experience with components and connectors, and object oriented technology.	To avoid inconsistency and inefficiency in design, staff should remain with the project.	There is a moderate impact on the quality of the system. The loss of expertise could result in delays and overall performance of the system.
O2: Management		
O2.1 Build vs. Buy		

There is a preference to acquire software packages including GUI facilities such as windowing rather than building from scratch.	This is negotiable; it may not be possible to implement some functions using generic packages.	There is an impact on meeting the development schedule.
O3: Process and Development Environment		
O3.1: Testing Process and Tools		
Sandwich Integration is preferred to properly test modules thoroughly.	This is negotiable; different testing methodologies can be implemented as required.	If testing is not done properly, this will result in delays and possible defects upon release resulting in possible loss of planes and lives.
Technological Factors	**Flexibility and Changeability**	**Impact**
T1: Changeability		
The module components should be easily updated to accommodate the need for modifiability and upgradeable software, hardware, and operating systems.	This is not negotiable since the system will quickly become obsolete if it cannot be easily adaptable to new environments.	The longevity of the system's life cycle is affected.

3.2 Module Configuration

3.2.1 Design Patterns Interactions

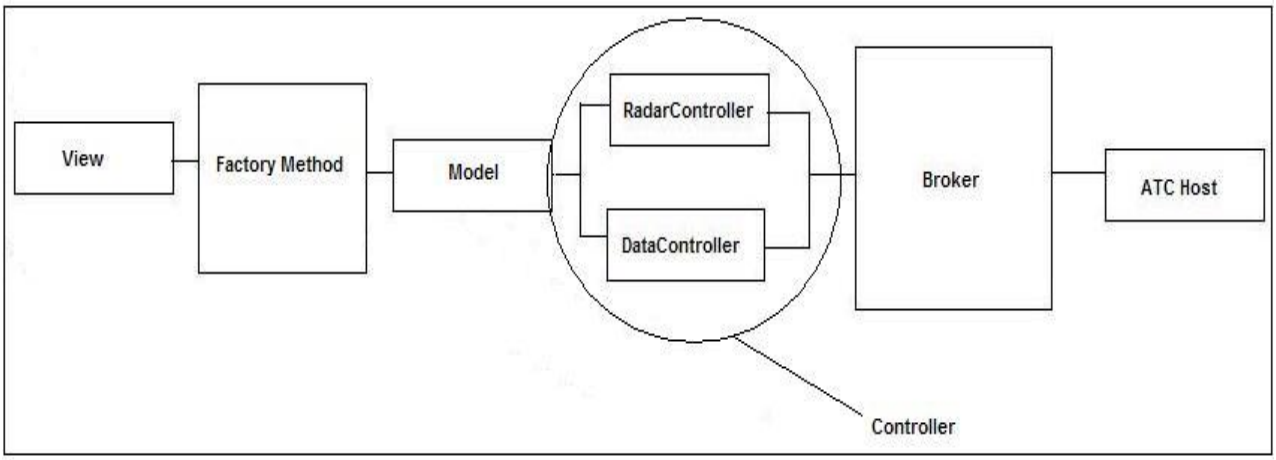

Figure 7: Design Patterns Interactions

The figure above illustrates the way in which the design patterns for the ATC system interact with one another. The ATC host acts as the server in the Broker Pattern, and communicates via a broker with the acting clients Radar Controller and Data Controller. These Clients in turn form the Controller aspect of the model-view-controller (MVC). The controller sends information to the model, which in turn initiates the creation of the views by utilizing a factory to make them.

3.2.2 Overall Module View

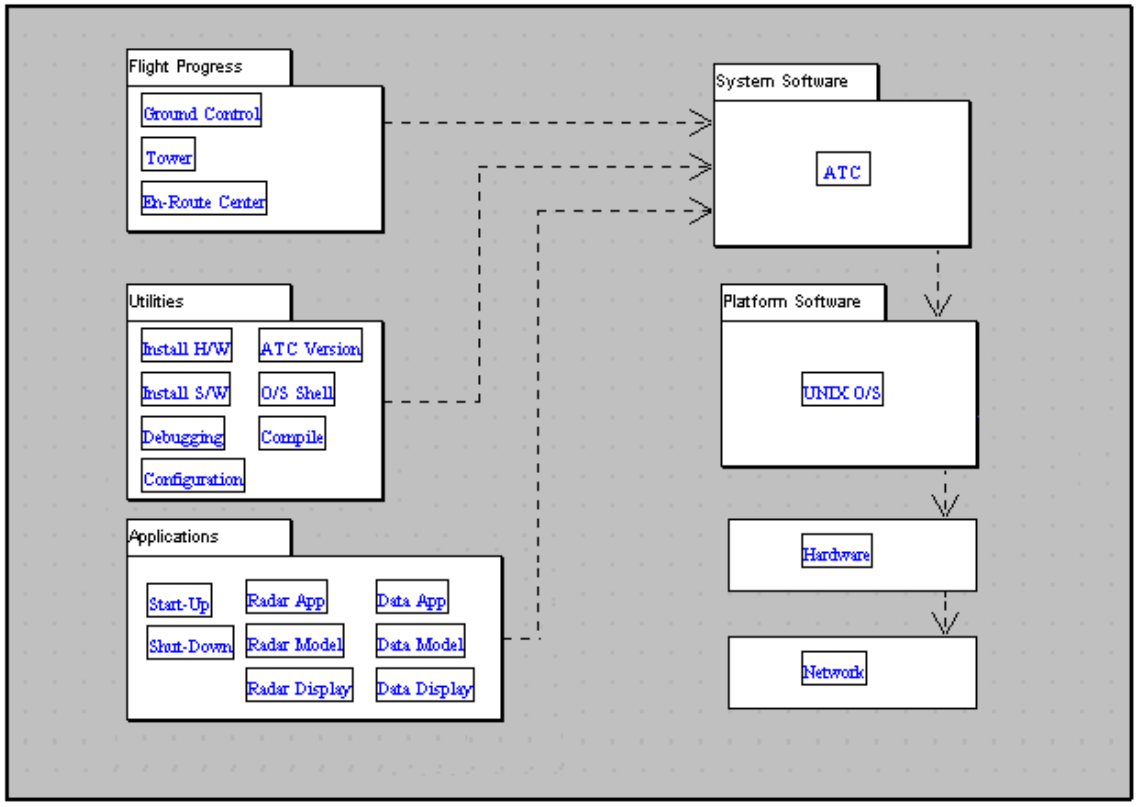

Figure 8: Overall Module View

The figure above depicts the overall module view of the Air Traffic Control System. As mentioned in the Global Analysis, one of the issues identified was reliability. The strategy to ensure reliability of the system was to *create a layered design* in addition to having an event driven system. This also addresses the issue of Real-Time Performance as the ATC is mission-critical software and events must be handled asynchronously for the safety of the planes and people. An advantage of the layered design approach allows us to minimize dependencies between modules, as core functionality is contained within the specific module. Fewer dependencies will make it easier to configure the system. As a result of layering, a change to one module, or even an error in one module, won't necessarily affect the whole system.

The layers of the ATC system include Flight Progress, Utilities, Applications, System Software, Platform Software, Hardware, and Network. These various layers contain modules essential for the functionality of the entire system.

The Flight Progress Layer contains modules relevant to the tracking of the aircraft as it travels from take-off to landing. The monitoring of aircraft begins with ground control and towers. The ground control keeps track of the plane while it is still on the ground, and the tower assumes responsibility within a certain radius of the airport. Once

the plane takes off, the data controller and radar controller within a sector suite of an en-route center interact with the ATC host to track the aircraft.

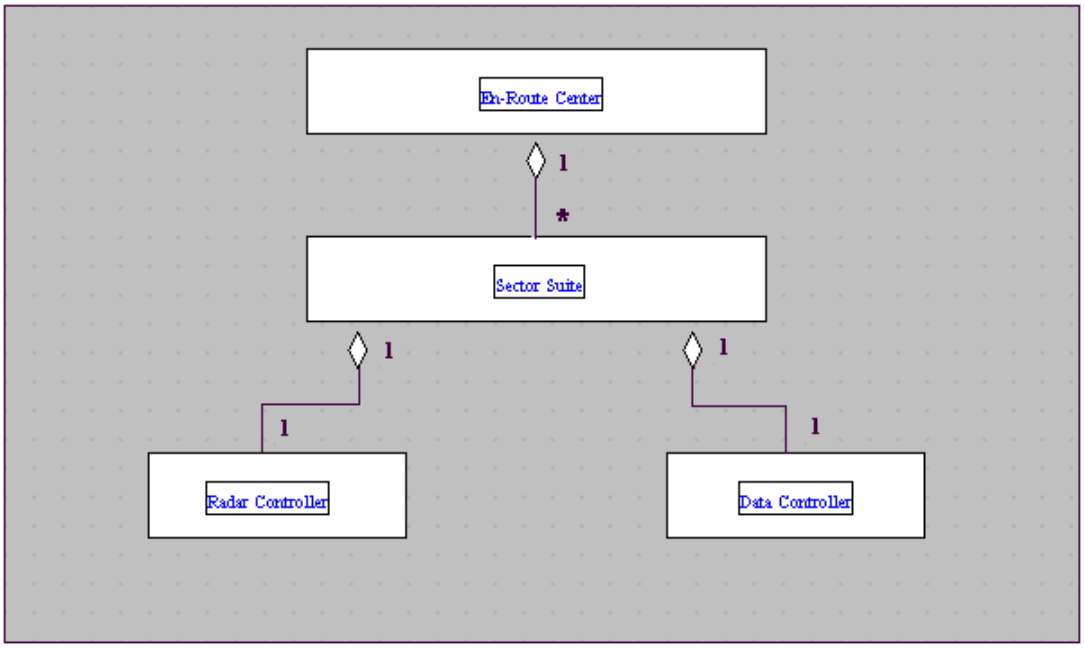

Figure 9: Decomposition

As depicted in the above diagram, each en-route center in a country consists of many sector suites. Each sector suite is associated with both a radar controller and a data controller. As the plane moves across the boundaries of the sector suite, the radar/data controllers of a neighbouring sector suite track the progress.

The Utilities Layer consists of all the modules that allow the system to allow for the issues of modifiability and reliability. These modules include the Install Software and Install Hardware to allow for configuration changes and upgrades. The debugging facility takes into consideration the *error policy* strategy as identified in the Global Analysis section. This module allows the system to ensure that errors are appropriately classified and logged in order to reduce the possibility of system failure, thereby reducing overall downtime.

The Applications Layer consists of all the modules related to views and displays on control consoles within a control center. The Start-up and Shut-down are provided in case of total system failure where the system may need to be completely rebooted. The RadarApp and DataApp represent the client functionality of the system. These include the previously defined conceptual elements Radar Model, Radar Display, Data Model, and Data Display.

The remaining layers, which include the System Software, Platform Software, Hardware, and Network, complete the functionality of the ATC system. The Network

connects all existing hardware together. The hardware is required to run the Unix operating system. Finally, the operating system is required to operate the Air Traffic Control software.

3.2.3 Broker Pattern

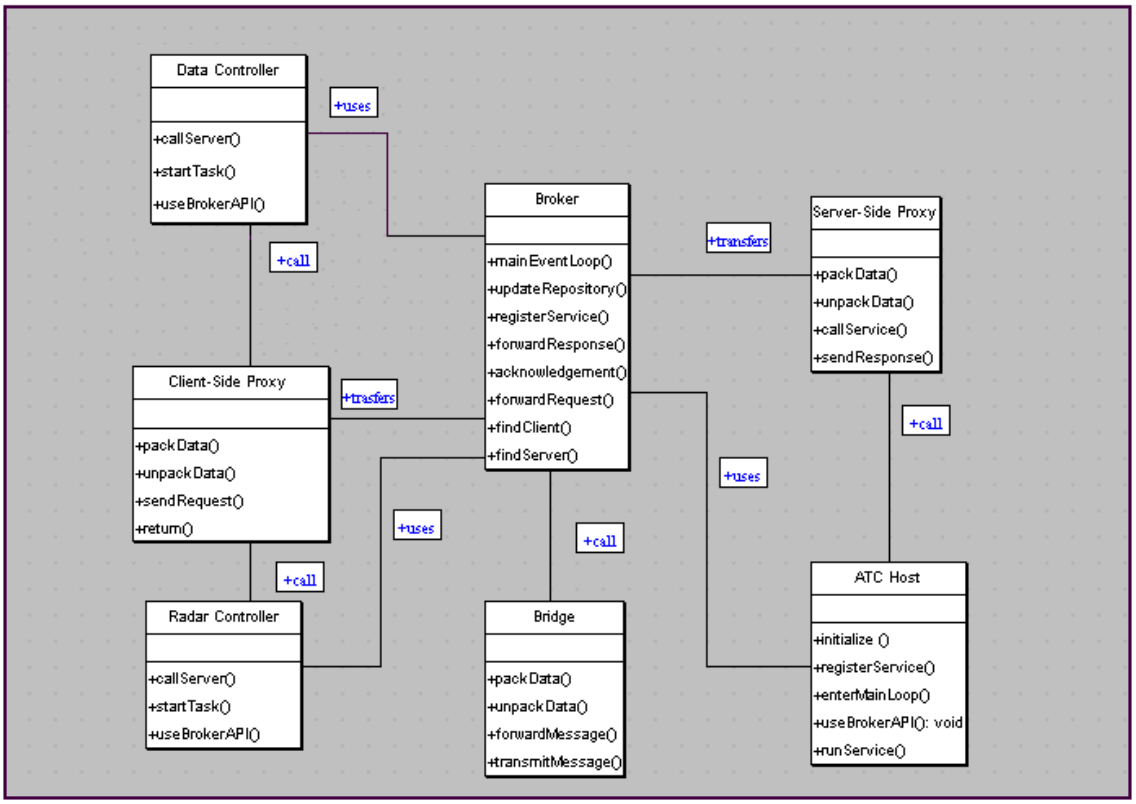

Figure 10: Broker Pattern

A distributed system with a network, such as the Air Traffic Control System, requires some sort of message passing mechanism to effectively communicate between its separate co-operating components. This mechanism should allow for location transparency, changeability, and extensibility in a real time system. The location transparency is needed so that different components may be concerned with just their individual functionalities. Each component is not concerned with other component functionality. The messages will be received in a central location and directed to the appropriate component in the system. Changeability and extensibility are required to allow for system flexibility in order to be upgradeable and meet changing needs.

The Broker Pattern is a message passing, real time, architectural design pattern that fulfills all of the above-mentioned needs. The location transparency is accomplished by the client and server proxies as depicted in the diagram above, while changeability and extensibility can be incorporated through the addition of clients and servers. This addresses the issue of modifiability as identified in the Global Analysis.

The Data Controller and Radar Controller are represented as clients in the diagram. These clients will send a request or a message by calling the Client-Side Proxy, and this request is then transferred to the Broker. The Broker is a central intermediary that receives all messages from the separate clients, and is then responsible for transferring these messages to the Server-Side Proxy. The message will then be communicated to the ATC Host.

The Broker Pattern being used for the Air Traffic Control system is a direct communication pattern, which is a variation of the original Broker Pattern. Once a link is established between two communicating entities, they are able to communicate directly as opposed to indirectly which would have occurred in the original Broker Pattern. The advantage to using this type of communication is the quick and effective passing of messages. This is crucial to a real time system, such as the ATC system, as messages must be handled promptly and efficiently.

A drawback of using the Broker Pattern is it makes use of proxies for the initial communication between clients and the server. This can cause the communication network to be somewhat slower, because of the indirect message passing. This, however, is only required until a link is made, at which point direct communication will take place. Another disadvantage of using the Broker Pattern is that it may cause testing of the system to become very complicated.

Although there are some disadvantages to using the system, the Broker Pattern was chosen to represent the architecture of the ATC system because it provides an effective communication mechanism whereby messages may be passed efficiently. This pattern provides *client-server architecture* as required to address the issue of real time performance. In addition, using the Broker Pattern also fulfills the strategy of using *design and architecture patterns supportive of event driven systems*.

The handling of error messages addresses the issues of reliability and availability of the system. As mentioned in the Global Analysis, the *error policy* was a strategy to effectively diagnose and classify errors so that they could be handled appropriately resulting in a reliable ATC system. By increasing the reliability, this would also decrease overall system downtime by reducing the possibility of a system crash, and thus the issue of availability is addressed.

3.2.4 Priority Queues

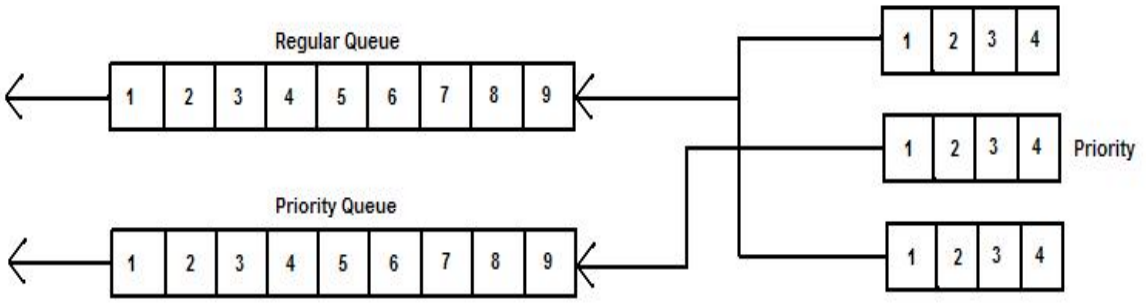

Figure 11: Priority Queues

An important requirement of the system is to be able to prioritize conflict alerts according to their importance. For example, an impending plane crash would need to be handled immediately as it could result in the loss of lives. This message would have to be easily identifiable by the Radar Controller so that preventive measures may be taken to avoid a collision. The prioritizing of messages can be accomplished by the use of separate queues containing normal messages and conflict alert messages.

Normal messages would be stored in a normal queue and would be handled on first come first serve basis. Conflict alert messages, on the other hand, would be stored in a priority queue and would be given priority over any messages stored in the normal queue.

3.2.5 Factory Pattern

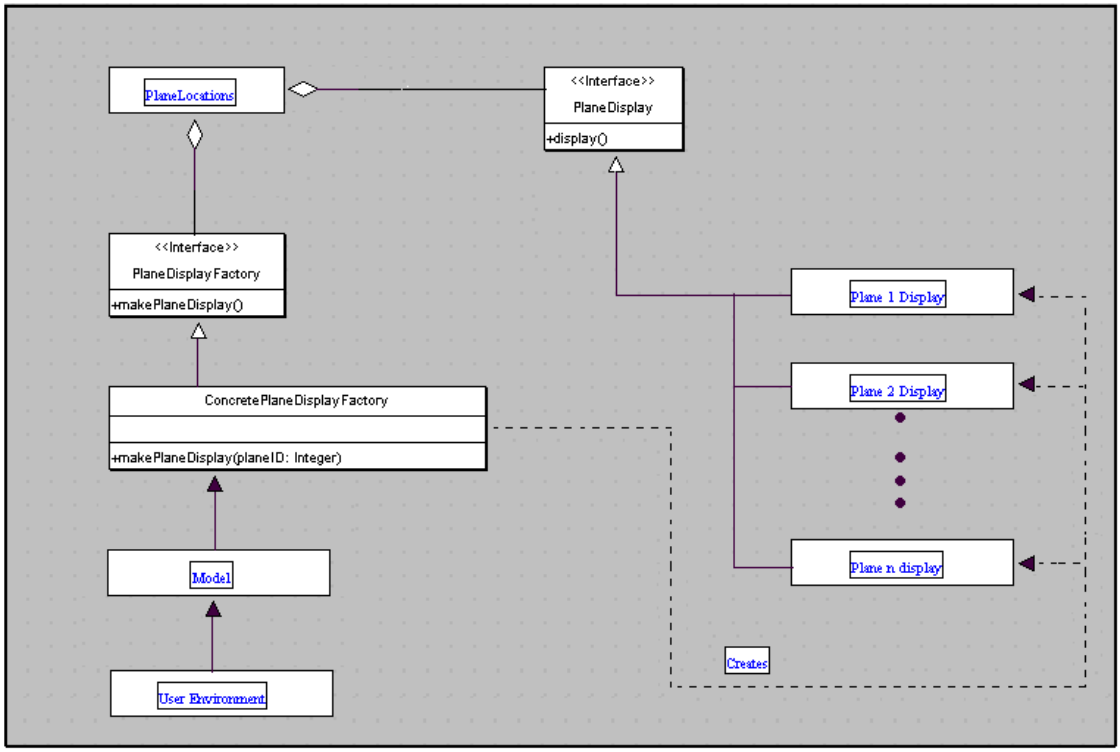

Figure 12: Factory Pattern

The Factory Pattern is used in the MVC component of the ATC system to create the console displays for the various aircraft being monitored. The User Environment represents the Data Controller or the Radar Controller, which would request a particular display that they require. This request is then processed in the model, and the model will then send a message to the ConcretePlanDisplayFactory to create the various displays, and the requested display is then shown on the console.

The Factory Pattern was chosen because the displays are created at the same time so they are accessible when needed. This is more efficient and flexible than creating separate displays one at a time, as more than one display may be needed on a control console. This addresses the issue of *real time performance* as it creates the displays faster, thereby increasing real time performance.

3.2.6 Classes

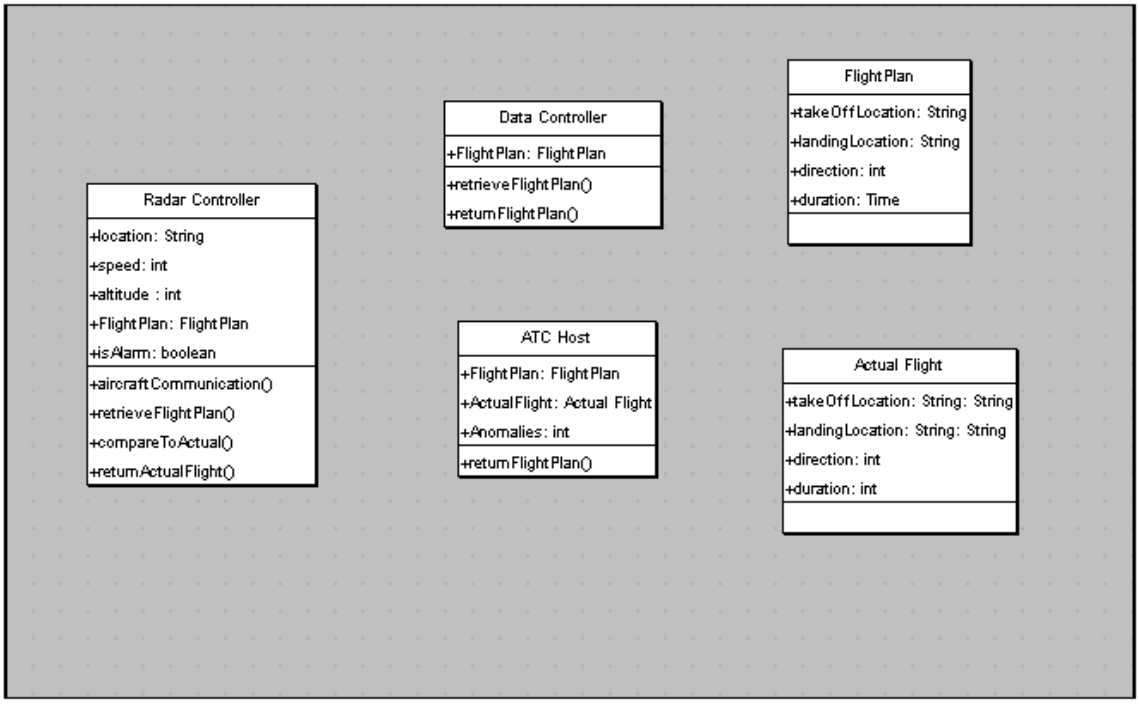

Figure 13: Classes

The classes required in the ATC system include the Radar Controller, Data Controller, ATC Host, Flight Plan, and Actual Flight. The Radar Controller is responsible for managing, and monitoring the tactical situation of the aircraft within a sector suite of the en-route center. This is accomplished through acquiring radar surveillance data such as the location, speed, and altitude. As well, the Radar Controller must maintain communication with the aircraft to determine any reasons for anomalies. Any problems that occur must then be communicated to the ATC Host by passing the Actual Flight object to the ATC Host.

The ATC Host contains all updated information about the progress of flights. It maintains communication with both controllers in all of the en-route centers by providing flight plans to the Data Controllers, and receiving the actual flight progress along with any anomalies from the Radar Controller.

The Data Controller is responsible for providing the Radar Controller with the flight plans for aircraft within a sector suite, or about to enter its sector suite. The flight plans are retrieved constantly from the ATC Host, and then passed on to the Radar Controller. The Actual Flight and Flight Plan classes contain all of the information about flights. The Data Controller retrieves Flight Plan objects from the ATC Host, which is what the plane is supposed to do. The Radar Controller sends Actual Flight objects to the ATC Host, which contain updated flight information and any anomalies that may be present.

4. EXECUTION VIEW

This section examines execution of the modules and subsystems in terms of system performance in regards to hardware/software requirements. Within our Global Analysis subsection, we have used a factor table, which lists the product, organizational, and technological factors. Within our Central Design Tasks subsection, we have also examined the execution of the modules and subsystems in terms of execution view, communication paths, and resource allocation.

4.1 Global Analysis

The purpose of this section is to discuss the different factors that affect the Execution View of the ATC architecture of the system and defines strategies needed in order to accommodate these in the design. Real-time events and hardware/software specifications are examined within this section.

4.1.1 Factor Table

The following factor table discusses the Product, Organizational and Technological factors that make up the Global Analysis of the Execution View.

Product Factors	**Flexibility and Changeability**	**Impact**
P1: Performance		
P1.1: Recovery Time		
In the event of a crash, the system should be able to recover to the state prior to the crash. There must be a back up for the system, which provides back-up processing and communication facilities.	This is not negotiable. There should never be an occurrence where system cannot be fully recovered within an extremely small time frame.	The reliability and availability of the system performance is affected.
P1.3: Acquisition Performance		
High-volume real-time events such as messages must be handled in an asynchronous manner.	This is not negotiable. When there are high volumes of events generated, any lag in handling events/messages is unacceptable.	This affects the real-time performance and safety.
P2: Failure Detection, Reporting and Delivery		
P2.1: Diagnostics		
Run-time errors should be handled through exception handling and never result in a crash or unexpected response from the system.	This is somewhat negotiable.	This affects the availability of the system.
Organizational Factors	**Flexibility and Changeability**	**Impact**
O1: Process and Development Environment		
O1.1: Testing Process and Tools		

All types of integration, acceptance, and performance testing should be done in order to ensure a reliable system.	The amount of testing is negotiable, but should be very high.	The system availability and safety is affected.
Technological Factors	**Flexibility and Changeability**	**Impact**
T1: Standards		
T1.1: Communication		
Communication of information between modules, and communication of information to the entities must both be done at optimal speed.	This is not at all negotiable. Slow communication between modules will result in improper functionality of the system.	This affects safety and real-time performance.
T1.2: Hardware Platforms		
The CPU is an IBM RS/6000 series processor for the console.	This is somewhat negotiable. Similar CPUs may be used.	This affects system reliability.

4.2 Central Design Tasks

4.2.1 Execution View

The architecture of the Execution View of the ATC focuses mainly on real-time performance, reliability, accuracy and modifiability. Each of these goals had to be met and were discussed in the overall conceptual global analysis. Afterwards different strategies were discussed in the global analysis in order to aid the mapping of these elements in the Module View to the Execution View.

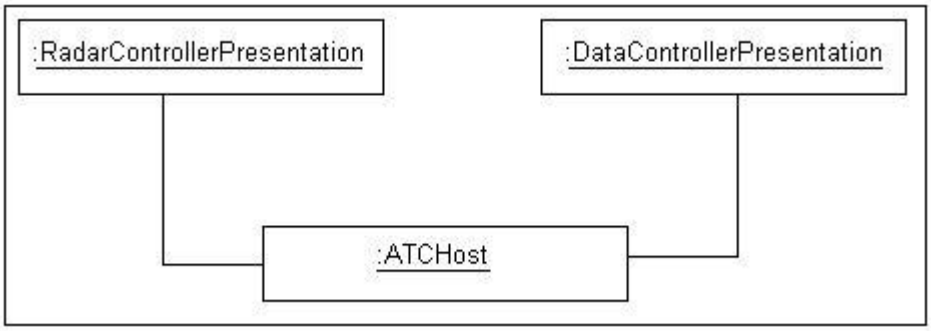

Figure 14: ATC Execution Architecture

The figure shown above is an overview of the ATC execution architecture view. The Radar and Data Controller Presentations and the ATC Host were mapped to their own processes. The RadarControllerPresentation sends commands/data requests to the host, which processes these commands, and gets back to the RadarControllerPresentation, as the DataControllerPresentation will do the same. Data is passed to and from the Host through the use of the Broker Pattern design pattern. A complete mapping of the RadarControllerPresentation and the DataControllerPresentation can be found in the Conceptual View.

4.2.2 Communication Paths

The main communication mechanism path in the ATC system is the broker. The broker is responsible for transmitting messages, by using client-server proxy relationships, from the Host to the Radar and Data Controllers and vice-versa. Incoming data come from the keyboard, or voice recognition system, if applicable; trigger the broker, our main communication path, to communicate the data over to the runtime entities. The keyboard uses Unix as the platform element, which in turn executes the runtime entities. The figure shown below illustrates the flow of communication:

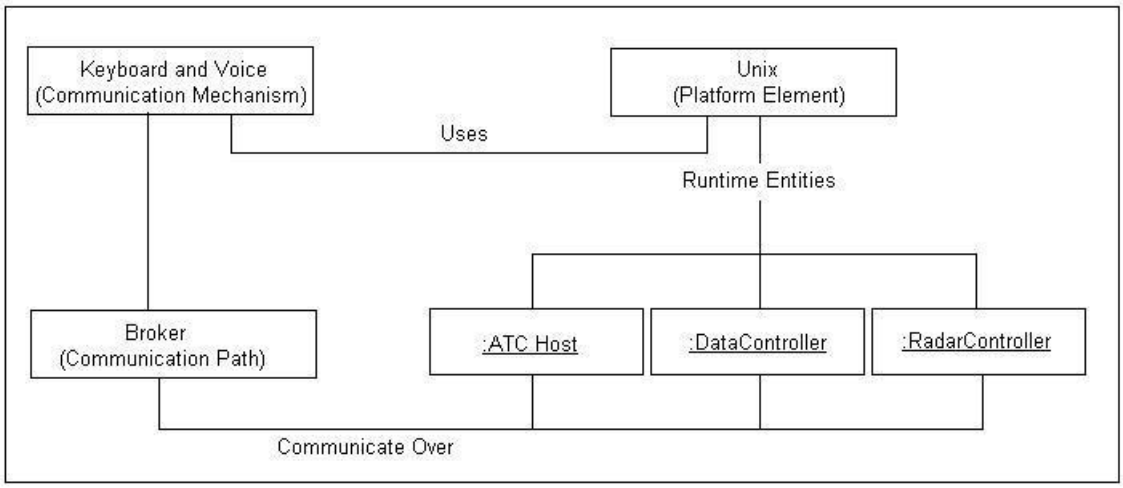

Figure 15: Communication Path Meta-Model

4.2.3 Configuration

The display of real-time data was crucial to the operation of the ATC System. Delay should be as minimal as possible. Hence, initial GUI process groupings for the two controllers were created. The Host acted as a Server and, along with the LogServer and PlatformManager, was attached to the network. The figure below shows how the processes, servers, and data are all interconnected:

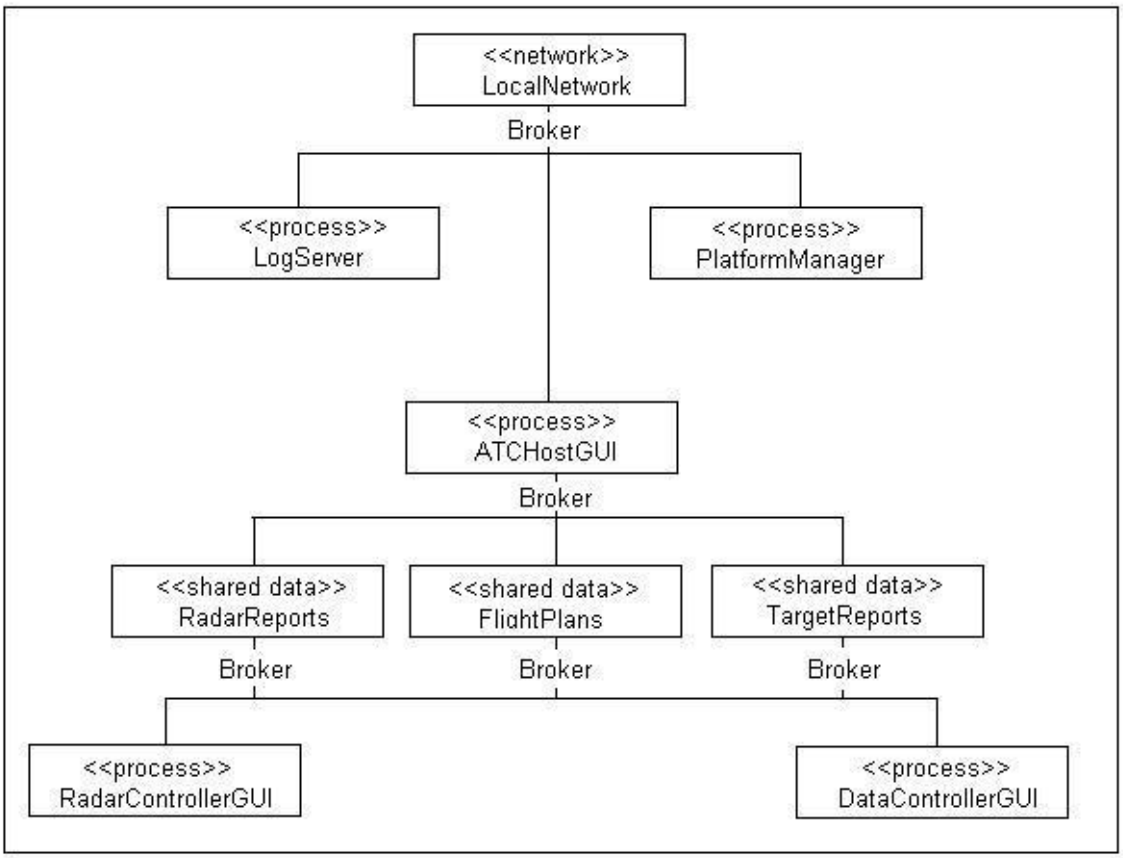

Figure 16: Execution View of Processes

The LogServer contained the Logging operating system services. The PlatformManager contained the Startup and Shutdown operations of the system. The Host, as discussed earlier, acts as the main database type server for the entire system. All three are connected to a local network and communication of messages is handled via a broker, by use of the Broker Pattern. Now the Host communicates to the RadarControllerGUI and the DataControllerGUI also via a broker. Radar reports, flight plans, and target reports are data that are shared between the Host and the two Controller GUIs. The Controllers, in turn, can also communicate and share data to the Host, via a broker.

4.2.4 Resource Allocation

Resources that are allocated during the execution view include main memory, CPU usage, processor speed, disk, hardware (radar equipment) and the Unix Operating system. Memory and CPU usage is high since data is constantly being passed back and forth in different stations. Lots of memory is required since huge amounts of data need to be stored in the system for reference information and later playback. The speed of the processor is also allocated to implement real-time data and asynchronous events and displays and to ensure high concurrency. A number of strategies were identified in order to achieve the best performance in resources. Most of these are discussed in the global

analysis, in the Conceptual and Module views. Radar equipment hardware is needed to track actual flight movements. It was suggested that module dependencies be minimized by using object-oriented design, and implementing information hiding and encapsulation principles, as discussed in the global analysis of the Module View. Reuse of modules and subsystems were highly emphasized to promote efficiency and reduce cost and time. Since the architecture strived to minimize module dependencies, resource usage and tried to allocate resources as efficiently as possible, the issues of configurability and performance were met.

5. CODE VIEW

This section will also describe how the different modules and subsystems will be hierarchically stored. Within our Global Analysis subsection, we have used a factor table, which lists the product, organizational, and technological factors. In the Programming Languages section, we have explained why we have chosen Ada – 95 as our source code programming language.

In the Directory Hierarchy section, we have described how the files and folders should be setup in for optimal hardware and software performance. Furthermore, we have described how the ATC will be setup within the root directory, along with the five major sub-directories, and how they are further broken down for modularity.

In the Intermediate Build Package section, we have described how the intermediate object and library files are produced, during the rebuilding of the executable files. Since these files need a physical location to both be created, stored, and if necessary, be upgraded and/or reinstalled, we have provided details for this.

Finally, in the Executable Software Package section, we have described how the ATC will have the ability to recompile both individual files and folders, and if required, the entire ATC system. Every previous build of the ATC system will also be archived in the host computer system, and will include both backwards compatibility and versioning.

5.1 Global Analysis

The purpose of this section is to discuss the different factors that affect the Code View of the ATC architecture of the system and defines strategies needed in order to accommodate these in the design. Concepts such as reuse of code, development of software platform and environment, are also examined.

5.1.1 Factor Table

The following factor table discusses the Product, Organizational and Technological factors that make up the Global Analysis of the Code View.

Product Factor	Flexibility and Changeability	Impact
P1: Functional Features		
P1.1: Reuse of Code		
Reuse of code and subsystems can be maximized by designing highly maintainable code, as if the code cannot be easily changed to suit a new purpose, it is not likely to be reused. Libraries may also be created to store commonly used functions and classes.	The amount of reusability is negotiable. Some specialized functionality may not be made easily reusable.	There is a moderate affect on the meeting of the schedule.

Organizational Factors	Flexibility and Changeability	Impact
O1: Staff Skills		
O1.1: Software Design		
The staff should have extensive knowledge of the languages, platforms, and patterns used to build the system.	This is not negotiable. Staff must be highly trained in real-time systems to ensure quality.	The quality of code produced, and the effective use of language features.
O2: Management		
O2.1: Environment		
An aesthetically pleasing environment, with adequate incentives for programmers and management will produce a better quality system.	This area is negotiable. Employee relations are extremely important in a successful project.	There is a moderate effect on the meeting of the deadline.
O3: Process and Development Environment		
O3.1: Development Platform		
The development platform chosen should support the language chosen. Unix is the platform of choice.	This is negotiable. Similar platforms such as Linux may be used.	There is a moderate effect on the development process and the efficiency of the system.
O3.2: Development Environment		
The environment should be Unix, as system security is important.	This is negotiable. Similar environments such as Linux may be considered.	There is a moderate effect on the development process.
O4: Development Schedule		
O4.1: Delivery of Features		
The development platform, environment, and schedule must all be followed closely in order to meet the release schedule.	The delivery of non-essential features is negotiable.	There is a moderate effect on the development process.
O4.2: Release Schedule		
A fully functional system must be released upon deployment. Non-essential components can be released at a later date.	This is not negotiable.	This affects system reliability.
Technological Factors	**Flexibility and Changeability**	**Impact**
T1: Software Technology		
T1.1: Configuration Management		

There must be configuration management to allow for modifiability of the system. Stubs may be used for later expansion of the system, and extensive documentation is important to help future developers.	This is somewhat negotiable.	This affects modifiability.
T1.2: Target Environment		
The target environment is Unix. Unix specific operations, such typical key commands, must be considered when designing the code for the system.	This is not flexible.	This affects the Air Traffic Controller's ability to efficiently use the

5.2 Programming Language

The programming language we will use to design and develop the ATC will be Ada-95. The reason we are using Ada-95, instead of another programming languages such as C++ or Java 2, is because the Ada-95 language enhances portability, maintainability, flexibility, and provides interoperability by standardization, in order to develop large, reliable applications. As an example, Ada - 95 is used for mission critical software packages, and used by organizations such as NASA for the launching and tracking of satellites within the Earth's orbit, and for deep space exploration programs. Additionally, Ada – 95 has already been established as the programming language for current European Air Traffic Control (ATC) systems. Hence, our conclusion from these examples is that Ada –95 is indeed a viable programming language to use for the design and development of our ATC system.

Furthermore, Ada-95 provides efficient real-time concurrent programming, with improved facilities for programming in large environments, and with an increased ability to interface with source code written in other languages. Consequently, interfacing with other programming languages by better standardizing the interface mechanism, is an advantage by using Ada-95. Another advantage of using Ada-95, are the special features, such as hierarchical libraries and partitions, to assist in the development of very large and distributed software components and systems.

5.3 Directory Hierarchy

Since the directory hierarchy is divided into five major folders, each folder is shown is shown in a separate figure.

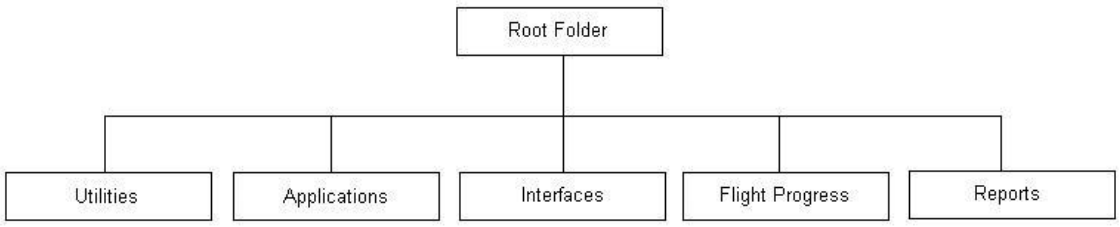

Figure 17: Directory Hierarchy

5.3.1 Utilities Folder

The Utilities folder will contain all the files and folders required for the ATC installation and utility maintenance. The Utilities Folder is the 1st folder that will be accessed, for the reason being that the ATC must be perform tasks, as listed by the sub-folder. This means that on the first instance of the ATC setup, or any other time the ATC is down, the system must:

- Install the hardware and software
- Configure the hardware and software
- Compile all of the ATC system source code
- Version the current build of the ATC system

The compilation of the ATC source code, and intermediate build package, along with the ATC final software package and versioning, are explained in greater detail in sections **Intermediate Build Package** and **Final Software Package**, respectively.

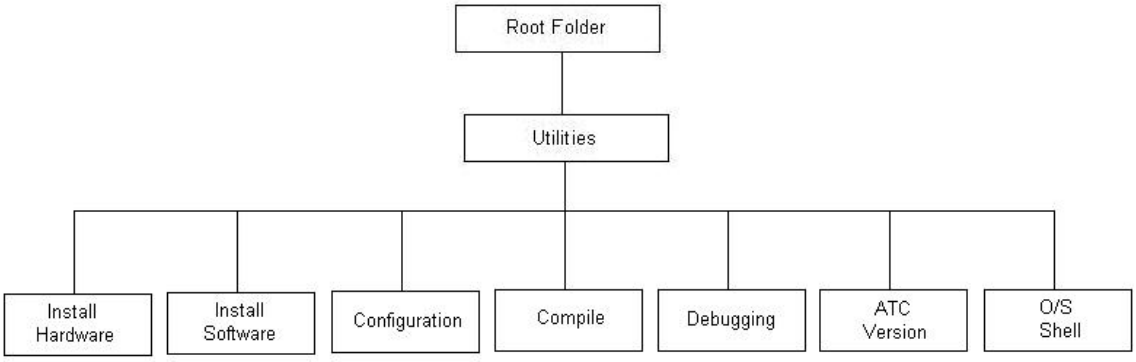

Figure 18: Utilities Folder

Within the Utilities folder, there are seven sub-folders, as shown in the figure above. The Install Hardware and Install Software folders will contain all of the files required to install hardware and software, respectively. The Configuration folder will contain all of the necessary files required for the hardware and software configuration.

We have also included both a Compile and Debugging folder, with the files required to accommodate real-time system upgrades and/or installations, of either hardware or software, respectively. Furthermore, since the ATC is a mission critical system, being able to compile and debug the ATC source code, in real-time, is a requirement.

Furthermore, we have also included an ATC Version folder, since every complete ATC source code build will be required to be versioned. Finally, we have also included an O/S Shell folder, in order to directly access the operating system, should it be required.

5.3.2 Applications Folder

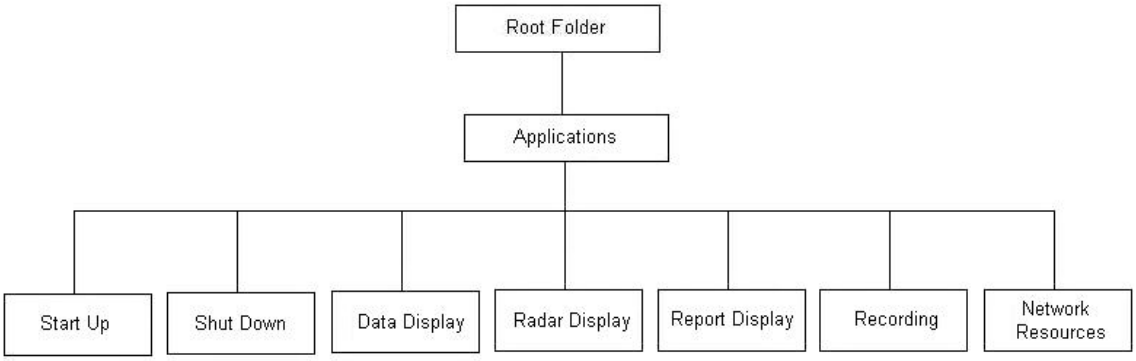

Figure 19: Applications Folder

The Applications folder will contain all the files and folder required for the applications that will be used in the ATC. Within the Applications folder, there are seven sub-folders, as shown in the figure above.

The Start Up and Shut Down folders will contain all of the files required to start up, and shut down the ATC, respectively. Both procedures are tasks within themselves, as the ATC cannot simply start up or shut down without performing their respective system tasks.

The Data Display, Radar Display, and Report Display folders will contain all of the files required to retrieve and display the ATC data, radar, and reports, respectively. The Recording folder will contain all of the files required for the continuous recording of the ATC transmissions, along with the files required for storage, retrieval, and playback. The Network Resources folder will contain all of the files required for maintaining and

monitoring all of the network resources, along with the dynamic reallocation of network resources as administratively required, in real-time.

5.3.3 Interfaces Folder

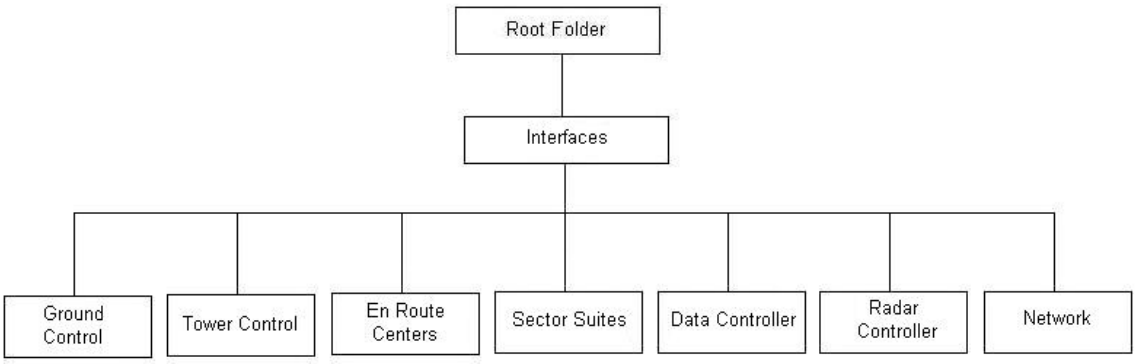

Figure 20: Interfaces Folder

The Interfaces folder will contain all of the files and folders for the ATC entities to interface with one another. Within the Interfaces folder, there are seven sub-folders, as shown in the figure above.

The interface entities consist of:

- Ground Control
- Tower Control
- En Route Centers
- Sector Suites
- Data Controller
- Rader Controller
- Network

Each interface entity's files are contained within the folder bearing the same name. Furthermore, the Interface folder itself will contain the files required to actually interface all of the interface entities, as required in real-time.

5.3.4 Flight Progress Folder

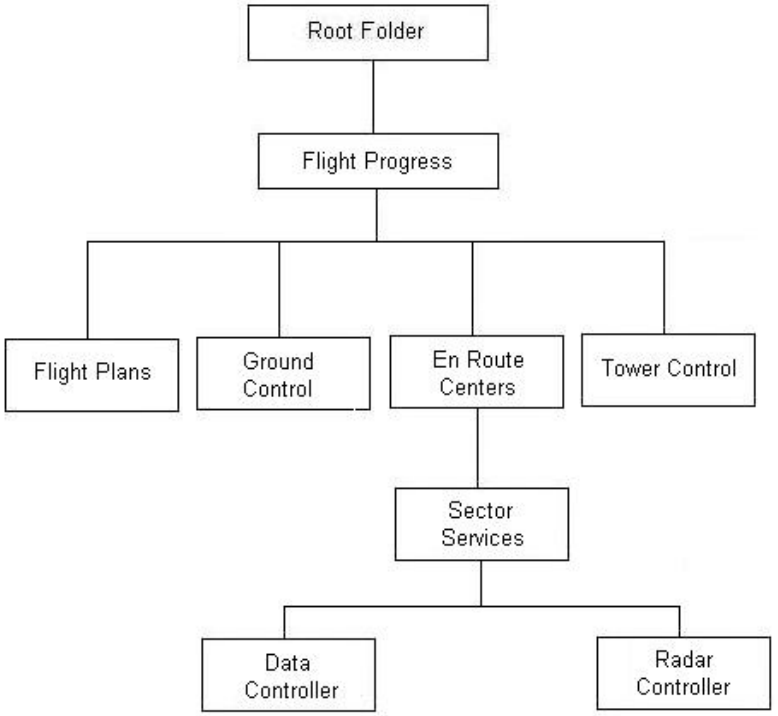

Figure 21: Flight Progress Folder

The Flight Progress folder will contain all the files and folders required for monitoring the progress of any flight. Within the Flight Progress folder, there are four sub-folders, as shown in the figure above.

The Flight Plans folder will contain the files required to access and retrieve the flight plans for each flight, from the flight plans database. The flight plan database itself will contain the following information pertaining to each flight:

- Airline
- Flight Number
- Type of Aircraft
- Point of Origin
- Point of Destination
- Time of Departure
- Time of Arrival
- Flight Time
- Flight Path

In other words, every flight that is been monitored has all of the following attributes associated to it. Furthermore, the last two attributes, Flight Time and Flight Path, are the most important attributes for the ATC in terms of managing air traffic and for generating the all of the required reports.

The Ground Control and Tower Control folders will contain the files required for the ground control and tower control to monitor the progress of any flight, respectively. The En Route Centers folder also contains all of the files required for the en route centers to monitor the progress of any flight, along with one sub-folder, Sector Suites. The Sector Suite sub-folder contains the files for the sector suites, along with two additional sub-folders, Data Controller and Radar Controller. These two sub-folders contain the files required for monitoring the progress of any flight for the data controllers and radar controllers, as per the ATC specifications, respectively.

The reason why we have placed the Data Controller and Radar Controller folders as sub-folders of the Sector Suite folder is because they are dependent on the Sector Suite folder itself, as per the ATC specifications. Consequently, this directly benefits the run-time performance for the respective folders and files.

5.3.5 Reports Folder

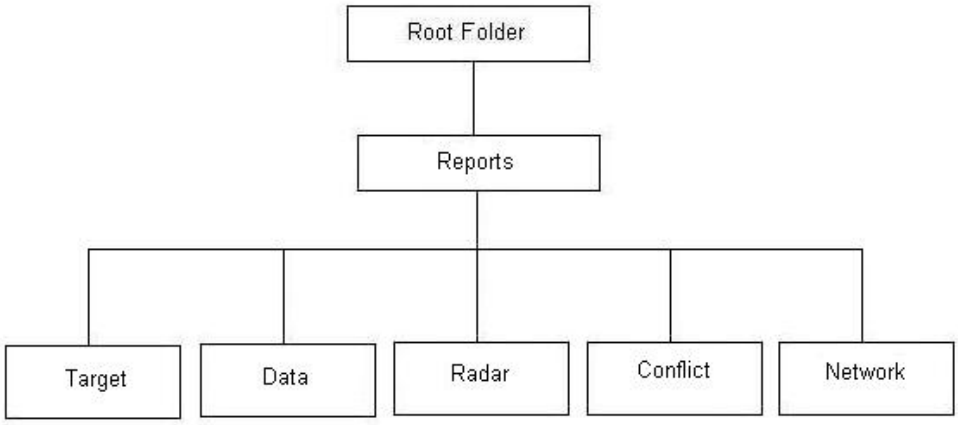

Figure 22: Reports Folder

The Reports folder will contain all the files and folder required for the reports that will be created, retrieved, and displayed, in the ATC. Within the Reports folder, there are four sub-folders, as shown in the figure above.

The report entities consist of:

- Target
- Data
- Radar
- Conflict
- Network

Each report entity's files are contained within the folder bearing the same name. Furthermore, the Report folder itself will contain the files required to actually retrieve and display all of the report entities, as required in real-time.

By separating the source code in a directory hierarchy, the representation of the source code and their relevance to the ATC is evident. The reason for this is because the source code file names will be named after the file component it represents, and also because it will be located within a directory related to the file component.

Furthermore, another advantage of using our directory hierarchy is for upgrading and/or re-installation of files. By re-linking and re-compiling only the required files necessary, this will allow the remaining ATC programs to execute normally, as they would not be affected. Additionally, if more than one directory, or sub-directory, required an upgrade and/or reinstallation of files, this could accomplished as parallel processing job, since directory, or sub-directory, would be able to link and compile their local files, at the same time. By using parallel processing, this would minimize the build time required for the necessary files to be linked, compiled, and executed, in order for the ATC to be functional with the most current program build.

However, the only disadvantage with our directory hierarchy is that tracing dependencies between directories, and sub-directories, requires a minimal amount of physical searching within the system. The direct result of this is that access time could take slight longer. In contrast, containing all of the source code in one large directory is an inefficient storage method, as the directory would become too large to maintain, and the linear searches would take much longer. By traversing through our directory hierarchy, the ATC will be able to trace the dependencies more efficiently, with a minimal time trade-off.

5.4 Intermediate Build Package

During the rebuilding of the executable files, intermediate object and library files are produced. These files need to a physical location to both be created, stored, and if necessary, be upgraded and/or reinstalled. Consequently, every directory, and sub-directory, will contain the following folder setup:

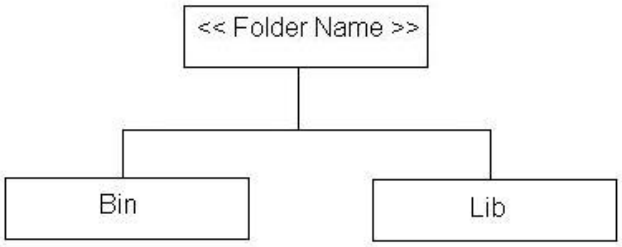

Figure 23: Intermediate Build Folders

The two sub-directories, Bin and Lib, will store all of the object and library files, which are created at time of compilation, or required for upgrades and/or reinstallation, respectively. The organization of all library files into one directory means the linking of executables would be more efficient, in terms of storage and run-time performance, since all the library files are located in directory. Consequently, another benefit of this is that now all of the executables could be run with different object files and/or library files, without directly having to recompile the entire folder contain the necessary files for the re-linking and re-compiling. An additional benefit is that the build time could be reduced even further, if only updated object files and/or library files, were required. It would be assumed that the object files and/or library files would have already been tested on a different, stand alone system, running the same executable program, and that they were verified to work correctly with the existing files on the ATC.

By describing how the parallel processing of re-linking and re-compiling files would occur, this would clearly increase build performance, by decreasing the development time. The only disadvantage would be an increase in the amount of storage space required. However, the trade-off for this is minimal, and is worth the overall benefit the ATC will gain. Furthermore, since these files would be part of the intermediate build package, it would not be a requirement that all of the object and/or library files. While most object files could be deleted, certain object files would have to remain in their folders, as they would be drivers for certain executable. However, they could be upgraded and/or installed, if for example, new drivers were to be implemented in the ATC. Also, the library files should not be deleted, unless they are being upgraded and/or reinstalled with new library files for the ATC.

5.5 Executable Software Package

The ATC will also have the ability to recompile both individual files and folders, and if required, the entire ATC system. A complete rebuild of the ATC system, will be archived in the host computer system as shown in the diagram below:

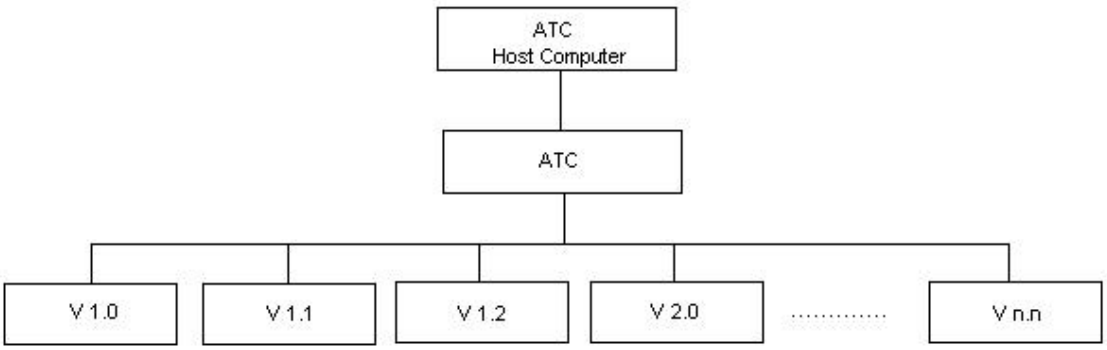

Figure 24: ATC Archive

Storing each version in its own directory would allow multiple versions to be run in parallel, if required. This means the ATC system could fallback to an older version, if the current one was being completely upgraded and/or reinstalled with new folders and files. The direct result of this would be no downtime for the ATC system, which would optimal for this mission critical system. As previously mentioned, the only disadvantage would be an increase in the amount of storage space required, for the archived ATC versions. However, the trade-off for this would still be minimal, as this would benefit the ATC system overall.

Every build of the ATC will also have backwards compatibility, in order to facilitate the transition to an earlier version, if required. The physical storage space used by all previous versions of the ATC could be reduced, by sharing folders and files between versions. However, this is too costly and inefficient, in terms of both the run-time and real-time performance to access folders and files on previous versions. Another disadvantage in sharing folders and files between versions is the additional maintenance, which would be required. If folders and files from previous versions were used concurrently with folders and files from the current version, this could potentially cause both run-time and real-time performance issues, due to compatibility issues, along with any common folder and file names. Thus, in order to avoid any of these possible scenarios, all previous versions of the ATC will be archived in its entirety on the host computer system, and on backup systems.

In addition to the backwards compatibility, every build of the ATC will also be version controlled, with the use of CVS, control versioning software. Before every new build of the ATC is mounted on the host computer system for operational use, it must be verified for proper versioning. This means that if previous versions were to be deleted,

this must be done thru the CVS, otherwise future versions of the ATC would not be versioned properly, and would not be permitted to operate on the host computer system.

6. CONCLUSION

As shown by this document, the Component Architecture of the Air Traffic Control (ATC) system requires detailed designs and descriptions in order to provide a clear and concise technical description of the Component Architecture of the ATC. We have used standard software engineering procedures such as Global Analysis and factor tables for each of our four major sections. Once again, these four sections are the Conceptual View, Module View, Execution View, and Code View. Furthermore, we have implemented industry standard design patterns, such as the Bridge Pattern, Broker Pattern, Factory Method Pattern, and Proxy Pattern, within our modules. Additionally, we have examined the execution of the modules and subsystems in terms of system performance in regards to hardware/software requirements, and we have also described how the files and folders should be setup in for optimal hardware/software performance

We are confident that our analysis, design, and development of the Component Architecture of the Air Traffic Control (ATC) system, is applicable in the actual design and development of this software application.

7. GLOSSARY

Ada – 95:

The programming language to be used to design and develop the ATC.

ATC:

Abbreviation for Air Traffic Control.

Bridge Pattern:

The Bridge Pattern decouples an abstraction from its implementation so that the two can vary independently."

(Source: http://c2.com/cgi/wiki?BridgePattern)

Broker Pattern:

The Broker architectural pattern can be used to structure distributed software system with decoupled components that interact by remote service invocations. A broker component is responsible for coordinating communication, such as forwarding requests, as well as for transmitting results and exceptions.

(Source: Buschmann, F., R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. Pattern-Oriented Software Architecture: A System Of Patterns. West Sussex, England: John Wiley & Sons Ltd., 1996)

Code View:

View showing transitive relationship between runtime entities to deployment entities, and deployment entities to source code entities

Conceptual View:

View showing conceptually how the ATC component is related to the Overall System Architecture.

CVS:

Abbreviation for control versioning software; vendor software required to verify and validate all versions of each subsequent ATC executable build.

Execution View:

View showing relationship between runtime modules and target hardware/software platform.

Factory Method Pattern:

Creational pattern that defines an interface for creating an object, but allows subclasses to determine which class to instantiate.

Factor Table:

Table consisting of product factors, organizational factors, and technological factors; along with their flexibility, changeability, and impact.

Global Analysis:

Broad overview of system input, advantages/disadvantages, and output.

LOC:

Abbreviation for lines of code.

Module View:

View showing modules and subsystems for the ATC component.

Model-View Controller:

Pattern breaking down into three views, Model, View, and Controller; which deal with Processing, Output, and Input respectively.

Overall System Architecture:

The overall architecture of the ATC system is divided into a number of components, or subsystems, among which the Model-View-Controller is a part of.

Proxy Pattern:

The Proxy pattern provides a surrogate or placeholder for another object to control access to it.

(Source: SENG 443 – Lecture Notes)

UNIX:

Operating system to be used on ATC host computer system.