International Journal of Computer Applications and Technology
journal homepage: www.ijcat.com

Design Analysis of Autonomous Air Traffic Flight Control System

D. Vasumathi
Department of Computer Science and Engineering,
JNTU Hyderabad,
India.

P. Rajarajeswari
Department of Computer Science and Engineering,
Madanapalle Institute of Technology and Science,
Madanapalle, India

A. Ramamohan Reddy
Department of Computer Science and Engineering,
S.V.University, Tirupathi, India

Abstract: Software architectural design, also known as top-level design, describes the software top-level structure and organization and identifies the various components. The concept of an automated air traffic flight control system which controls airplanes requires a high degree of operational integrity and availability. One possible solution to alleviate air travel congestion could be the automation of air traffic control and allowing it to have direct control over airplane flight paths. Such a system would, in theory, reduce the workload of the flight crew and the air traffic controllers, as well as increase traffic flow. This paper presents several analyses of such a conceptual system from a "net-centric" perspective. First, the system's operation is described from the context of a flight, to provide a basis for the discussion of various system models and views. Spiral development model stages as well as related events which occur during system design give an idea of how the system would be developed incrementally. Formal methods can be used to improve software security but can be costly and also have limitations of scale, training, and applicability. To compensate for the limitations of scale, formal methods can be applied to selected parts or properties of a software project, in contrast to applying them to the entire system. The concept of object-oriented development (OOD) has gradually matured from being presented. The OOD can still be regarded as one of the mainstream development models. UML includes a standardized graphical notation used to create an abstract model of a system, referred to as a UML model. We describe AATFCS system with UML modeling techniques. AADL is an extensible and allows us to introduce new properties; we can define a set of properties specific to the data state variable. In this paper we present the AADL language for AATFCS system.

Keywords: Software Architecture, Autonomous Air traffic Flight control system, Spiral development, UML modeling analysis, Architecture analysis design language.

1. INTRODUCTION

The architectural design allocates requirements to components identified in the design phase. Architecture describes components at an abstract level, leaving their implementation details unspecified. Some components may be modeled, prototyped, or elaborated at lower levels of abstraction. Top-level design activities include the design of interfaces among components in the architecture and can also include database design.

Formal methods are the incorporation of mathematically based techniques for the specification, development, and verification of software. The OOD can still be regarded as one of the mainstream development models. Obviously we have approaches to describe software architecture according to such concept. As we know, in software engineering, the famous Unified Modeling Language (UML) (Booch, 2005) is a non-proprietary specification language based on the concept of OOD for object modeling. The UML is an effort to create a standard, generic, graphical modeling language for software systems, as a general-purpose modeling language, UML includes a standardized graphical notation used to create an abstract model of a system, referred to as a UML model. A software designer can describe the system architecture employing UML and kinds of models.

Air traffic congestion is rapidly becoming one of the major commercial transportation challenges at the start of the 21st century as more people take to the skies for their travel needs. "Forecasts indicate a significant increase in demand, ranging from a factor of two to three by 2025…. In short, U.S. competitiveness depends upon an air transportation system that can significantly expand capacity and flexibility, in the presence of weather and other uncertainties, while maintaining safety and protecting the environment"[1].

We describe Autonomous Air traffic flight control System in section 2. In section 3 we provide AATFCS System Architecture Modeling and Analysis. We present the design of architecture for an autonomous air traffic control system using Uml modeling techniques in section 4.In section 5 we provide AADL for an AATFCS system. We presented conclusions in section 6.

2. Autonomous Air Traffic Flight Control System Description

Although the system description of the AATFCS System description provide the information is included here in order to give clarity and context for the analyses of this system.

2.1 AATFCS System Overview

The Automated Air Traffic flight Control System consists of two primary system element types: ground stations and airplanes. The two types are connected via an air-to-ground wireless network and are in constant communication with the other nodes in the network. Each system element type also communicates with other network members of its own type: ground stations within the vicinity of an airport are linked to each other and airplanes communicate with other airplanes within range. Ground stations have additional interfaces with secondary system elements such as external data sources. Airplanes possess their own internal networks which connect on-board subsystems to flight control computers. Each element and its architecture and interfaces are described in further detail in this section. A top-level diagram of the system is shown in Figure 1.

The pilot is assumed to take control at this point for Free Flight during cruise for the reasons previously mentioned in the Background section. However, the pilot could decide to allow the automatic model to continue computing the flight vector and fly the plane based on the last valid commands received and its current position, with updates provided by any "waypoint" ground stations it connects to and authenticates with en route. The airplane does not attempt to connect with another airplane in an ad-hoc air-to-air network until it reaches its destination. As the airplane enters the airspace of the destination airport, it once again connects to and authenticates with the local air-to-ground and air-to-air networks.

The system performs the same actions as during take-off, though in reverse. The pilot, if in command, relinquishes control of the airplane after the data from the local networks has been validated. The airplane then automatically slots itself for approach and landing, in accordance with the ground station's instructions. After landing, the airplane taxies off the runway and transitions back to pilot control before reaching the gate.

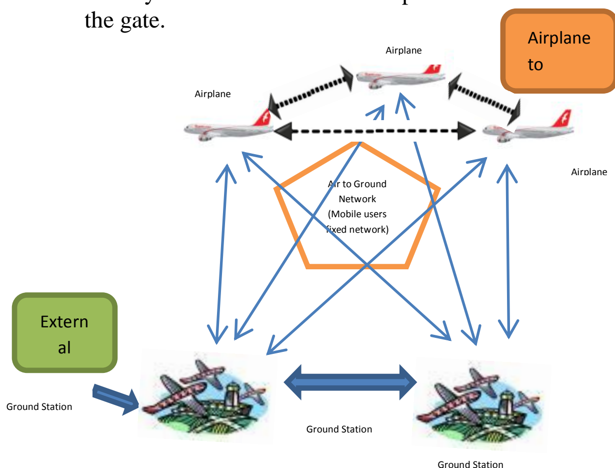

Fig. 1 – Automated Air Traffic Flight Control System

3. AATFCS System Architecture Modeling Analysis

Architectural modeling is an important enabler for the understanding and comprehension of a complex system because it can provide unambiguous representations or views of the system's architecture and behavior. One definition of a model is "a virtual or physical representation of an entity for purposes of presenting, studying and analyzing its characteristics such as appearance, behavior or performance for a prescribed set of operating environment conditions and scenarios." Any system can be modeled from numerous points of view which are essentially projections of the system onto one or more operational domains. It should be noted, however, that although models may adequately represent the system for the purpose of further design and implementation, they are still a finite set of projections which limit their ability to exhaustively describe the system and its behavior because of the heuristic which states that "a model is not reality." It is just as important to be aware of the models'.

Irrespective of their limitations, it is important to develop system models early in the development phase of a program in order for stakeholders – people with an interest in the development or outcome of the design – to develop a common understanding of what the system will look like and how it will operate. Without this, errors from misinterpreting or misunderstanding the system's characteristics creep into the design and create nontrivial problems (often very big problems) in terms of schedule and cost when the errors are discovered and need to be fixed. In fact, poor communications has been cited as the number.

A picture is worth a thousand words' is a classic heuristic and a good set of system models can be worth their development cost by preventing errors which, if undiscovered, can propagate to later design, implementation and verification phases.

3.1 Spiral Development Model Stages

"The spiral model is a software development process combining elements of both design and prototyping-in-stages, in an effort to combine advantages of top-down and bottom-up concepts."[11]This approach allows for the iterative risk assessment of the design at various stages along the development path and "promotes quality assurance through prototyping at each stage in systems development." [12]Each loop of the spiral represents a single iteration and each quadrant represents one of four stages of design: determining objectives, alternatives and Constraints; identifying and resolving risks; development testing and planning the next. As the spiral progresses outward from the origin, each successive loop builds on the previous iteration and provides incremental functionality and risk reduction prior to the next loop. The horizontal axis is labeled 'review' to indicate the point in the spiral where a review is required before proceeding into the next loop and the vertical axis is labeled 'cumulative cost' to show the accumulating cost per

loop.

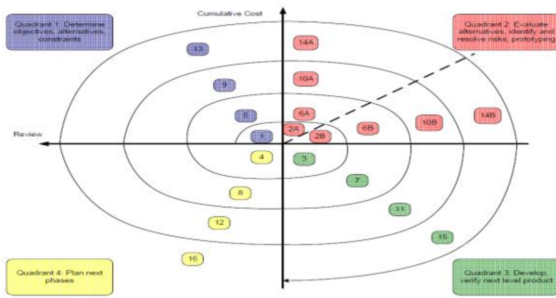

Fig. 2 – AATFCS Spiral Development Model

Figure 2 shows the spiral development model for the AATFCS. The spiral does not start at the origin but instead starts already established in quadrant 1. This is to indicate that the first task to be done in the development of the system is an initial review and determination of objectives, alternatives and constraints at the very top level. The dashed radial line in quadrant2 is the dividing line between risk (left) and prototype development (right) for a given loop. This shows that the risks must be assessed and a "phase gate" type of evaluation must be passed inured to allow development to continue for that iteration or phase. If the evaluation does not meet its pre-determined criteria, development can be halted or terminated.

4. Design of architecture for an autonomous air traffic flight control system

System architecture is a set of design decisions. These decisions are technical and commercial in nature. To meet the functional and nonfunctional requirements of the above said ATFC system it is necessary to model the complete AATFC system by the use of UML. Different types of diagrams are Request departure clearanceDepartGrant departure clearance designed and described below in brief: UML is perhaps the most well-known commercial industry modeling language today. The unified Modeling Language is a method by which one can "describe a complex system rigorously and unambiguously…such that the integrated system design can be tested and verified to meet requirements before generating any code or designing any hardware," for the reasons mentioned previously. System modeling takes place in task 3 of the spiral development model, which is early enough to provide assurance that the mission requirements, the overall system architecture, and the subsequent hierarchical decomposition are communicated among and understood by the program stakeholders.

The UML system architecture diagrams presented in this section are described from the use case perspective of an airplane's approach and landing. However, in order to model the system correctly, a brief discussion to provide understanding of the mission-level operations for this use case shall first be presented.

4.1 Airplane Approach and Landing Description

Some of the high-level description of an airplane's fault-free approach and landing has been previously mentioned in the discussion of the AATFCS system's operation. Additional details which describe the order of events can also be used to help establish the context for modeling the system properly using UML. It should be noted that understanding the system's fault response, which can be based on a system-level FMECA, is also required to generate a more complete model of the system.

Use Case Diagram

The use case diagram "provides a tool for organizing system requirements in order to understand interactions between:

- "Actors" that make a request, and
- "Activities" made in response by the system

The AATFCS use case diagram in Figure 3 shows the "fault-free arrival" use case for the AATFCS.

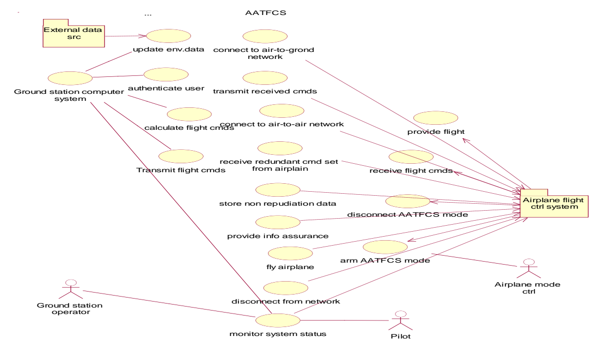

Fig.3 – AATFCS Fault-Free Airplane Arrival Use Case Diagram

The actors and activities in the use case diagram back to the steps in the airplane landing sequence. Note that the Ground Station Operator actor, External Data Source external system and the Update Environmental Data and Provide Information Assurance.

Class Diagram

UML class diagrams "show the static structure of the system at an abstract level" [15].In object oriented programming; classes are abstract representations of software objects, which are, in turn, instantiations of the class. The class diagram in Figure 4 shows representations of the two primary object templates in the AATFCS, the AATFCS subsystem and the network. The highest, most abstracted level of the class hierarchy for net-centric systems would generally depict only a generic object type in the system and possible connection methods (i.e., the network(s)). Each class in the diagram has three fields; from top to bottom, they are the name of the class, the attributes associated with the class, and the methods associated with the class. Additionally, the connections in the hierarchy depict aggregation, inheritance and multiplicity. A closed (filled) diamond indicates that the parent class is comprised of N child level items, with N being a specific or unspecific number or range of numbers such as 1, 1..3 (1 to 3), N, or 1..* (1 or more). The example shows that the AATFCS is comprised of 1 to 5 network types (air-to-air, air-to-ground, ground station network, AATFCS data bus network, or actuation data bus network).

There are three types of lower-level classes called ground control system, SWIM system and airplane connected to the subsystem class. These inherit the attributes and methods listed in the respective fields in their parent class; the

inheritance is shown by the open (unfilled) triangular arrows pointing up from the child classes to the parent class. Each of the three inherited classes has continuing levels of decomposition which are left out of the diagram for clarity except for a few key examples. Similarly, the network class has the five child classes which inherit characteristics from it, as previously described. The empty fields of the child classes indicate that they are not instantiatable – they are the equivalent of abstract classes in object-oriented programming.

The SWIM system class is the only class which is decomposed in greater detail in this example; the other classes at this level all decompose to one or more levels further down in the overall hierarchy. The SWIM system class is shown to be comprised of SWIM flight data and system status. The flight data class contains attributes of weather data, airplane flight plan data and pilot authentication data, which are all "inputs" to the class and are annotated as private data (the minus sign preceding the name). Private attributes are not exposed to other classes. The fourth attribute, SWIM data, represents the outgoing message to the ground station and is considered public data (the plus sign preceding the name) because it would be exposed to other objects as part of the transmission process allocated to the public method"provideFlightPlanData ()". In the system status class, the internal status methods and the data attributes are private and the message attributes, along with the display message method, are public.

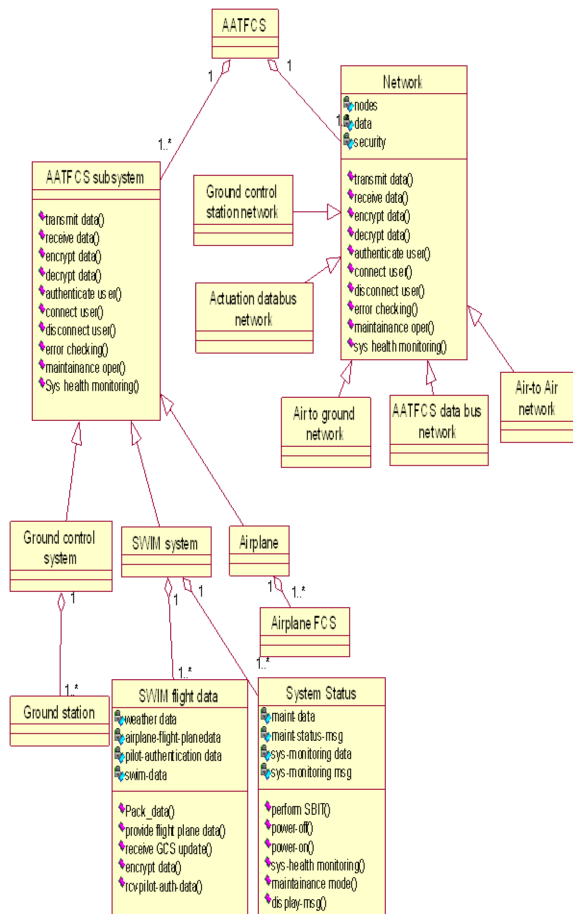

Fig. 4 – AATFCS Top-Level Class Diagram

By continuing this process for all objects, we can derive a hierarchical representation of each object in the system which describes not only the data but also the actions performed on that data.

Sequence diagram

A UML sequence diagram will "model logic flow within a system in a visual manner, especially dynamic modeling of system behavior [14].The sequence diagram does this by showing interactions between objects at various points in time, in sequential order. Time is represented increasing from top to bottom and each entity (e.g., object, actor, etc.) will have a dashed vertical line beneath it which indicates its lifetime within the sequence. Objects in particular are shown as instantiations of their class; they are specified as OBJECT: CLASS. Object lifelines turn into a wider bar (an 'activation') upon instantiation and return to the dashed lifeline when the object has been removed from the sequence (i.e., destroyed or de-allocated). "The activation represents an execution of an operation the object carries out. Each activity line contains the name of the message associated with the source object, along with data that is passed to the destination object,

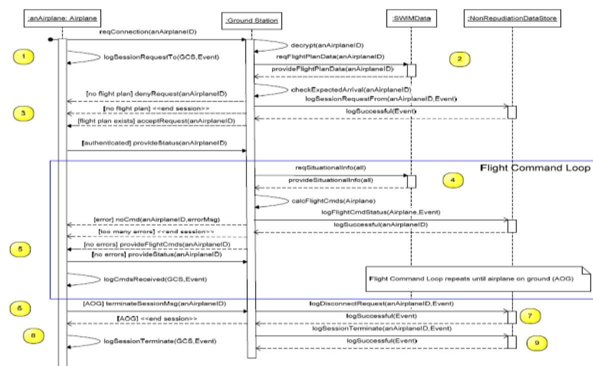

Fig. 6 – AATFCS Airplane Arrival Sequence Diagram Example

5. Architecture Analysis and Design Language for an Autonomous Air traffic Flight control system

A common way of modeling such meta-information in AADL is to associate AADL properties with the item in question and record information about the item. For example, the measurement unit and confidence of data may be recorded in properties. Since AADL is extensible and allows us to introduce new properties, we can define a set of properties specific to the data state variable. In some cases, this Meta information is communicated explicitly with the data and is checked by the application at runtime. In this case, the Meta information is declared to be part of the data representation, either just reflected in the increased size of the data type, or explicitly as a data subcomponent in a data component implementation declaration. State variables are communicated between Ground and Flight systems via telemetry. The data transport mechanism uses State Variables and State Variable Proxies. A State Variable represents the location in the deployment where the state is being locally estimated, and a Proxy State Variable represents a remote location that intends to utilize state variable content remotely. This deployment is shown in Figure 5.

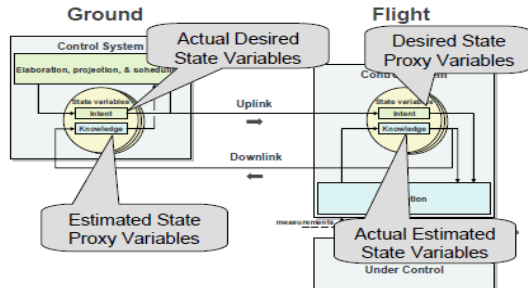

Fig 5 Deployment of State Variables

The deployment of these data is such that Estimators in a deployment update their corresponding State Variables (SV). The data transport mechanism occasionally collects the value histories stored in these SVs and transports these histories to appropriate Proxy SVs in other deployments. The same data transport t mechanism is used to transport measurement histories and command histories between deployments (from Basis Hardware Adapters to Proxy Hardware Adapters). Systems engineers specify what information needs to be transported between deployments, and the regularity of proxy updates.

The telemetry transport mechanism is used, then, to update the proxies with actual values with a specified periodicity or on demand. At a high level of abstraction of the AADL model, the state variable proxy notion can be encapsulated in the protocol used by the telemetry (Space Link) bus component. It is the responsibility of the protocol to distribute the state to the out data ports of components to other components. For data port connections across the Space Link bus, a different protocol is used to provide the desired caching strategy of the state variable proxy. The application model is agnostic to this proxy/caching scheme.

If it is desirable to explicitly model the proxy scheme, we can do so in two ways. We can model an implementation of the proxy/caching protocol of the telemetry bus component as a separate AADL model that is associated with the Space Link bus by property. This property is interpreted by the instance model generator to refine the bus abstraction by its implementation. Alternatively, we can model the proxies explicitly as application components (i.e., as threads that receive the original data port content by executing at a specified rate and make it available locally). In this case, users need to modify the model by inserting or removing the proxies as components that are migrated between flight system and ground [16].

Package Control Software
Public
-- This type is refined for a AATFCS instance by refining the
-- Classifiers of the features to be instance specific
Thread group controller
Features
State Estimates In: **port group** AATFCS Data::State Estimates In;
Estimate History In: **port group** Value Histories: Estimate History Inv;
Control Goals In: **port group** AATFCS Data::X goals in;
Commands Out: **port group** AATFCS Data::Commands Out;
End controller;
Thread group implementation controller. Basic

End controller. Basic;

Fig6 Example package of AATFCS system

5.1Operating System Thread Model

Hardware adapter, estimator, controller, planner, goal executive, and goal monitor are represented by logical threads, each with an execution rate, a deadline, and a worst-case execution time. Some of this functionality may be distributed between flight system and ground or may be distributed within the flight system or ground system. The latter distribution may occur due to a multiprocessor configuration or in anticipation of using multi-core chip architectures in a spacecraft.

Distribution decisions regarding ground or flight system are localized to changes in processor binding property values in the AADL model, unless state variable proxies are modeled explicitly as part of the application system. The collection of logical threads bound to the ground processor the flight processor is then grouped into rate groups. Each member of a rate group is executed by an operating system thread at the period of the rate group. Note that such rate group optimization must take into account execution order requirements between threads of the same rate or of different rates that require data to be communicated mid-frame (i.e., within the same execution cycle).

Property set Rate Groups **is**
Rate Groups : **type enumeration** (EstimatorRategroup, ControllerRateGroup, PlanExecutionRateGroup, PlanningRateGroup, HWARateGroup);
AssignedRateGroup : **inherit** RateGroups::RateGroups **applies to** (**thread**, **thread group**, **process**, **system**);
end RateGroups;

Fig 7: Rate Group Modeling by Properties

Rate group optimizations can be represented within the current version of AADL using the property mechanism. We can introduce a property type Rate Groups that is an enumeration of rate groups in a particular application and a property to specify the rate group that a thread is assigned o, as illustrated in Figure 7. The enumeration literals are an ordered set. AADL V2 introduces the concept of virtual processor to model hierarchical schedulers. The operating system threads, which execute the tasks of a rate group, act as schedulers that dispatch these tasks as a cyclic executive. Therefore, we represent each of them as a virtual processor to which the application AADL threads are bound. Each of these virtual processors is defined as a subcomponent of a given processor or is defined separately and bound to a processor.

5.2 Binding to Hardware

AADL supports modeling the computer platform of the embedded system. In Figure 8, we illustrate how flight system and ground system computer platforms can be modeled. The flight system consists of a processor, memory, and a flight system bus. In addition, the flight processor has access to a

device bus that is also accessible by devices representing the sensors and actuators outside the MDS computer hardware system component. The ground system consists of a processor, memory, and a ground system bus. The two computer platforms are interconnected via a Space Link bus that represents the downlink between the spacecraft and the ground station. Without having to model the internal details of the hardware, we can use properties to specify characteristics relevant to the analysis of embedded systems.

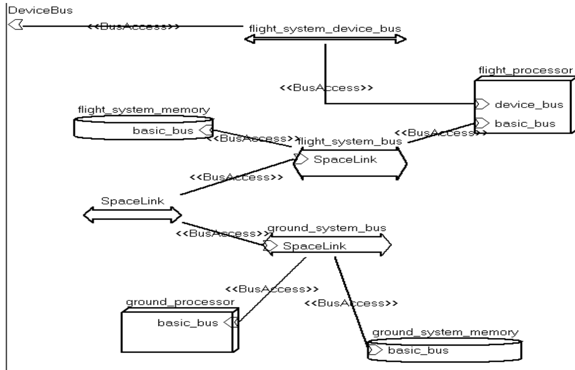

Figure 8 Flight and Ground Processing Systems

The binding of embedded software applications to the computer platform is also accomplished through properties. The Allowed_Processor_Binding property places constraints on the binding to processors. The binding may be constrained to a processor type or to a set of processors. Binding constraints are taken into consideration when a resource allocation tool makes its allocation decisions; the Actual_Processor_Binding property records the actual binding decisions shows the use of Allowed_Processor_Binding for the AATFCS architecture. This property is declared with the top-level system. Implementation allowing the property declaration to refer to the processor as the reference value and to the application component to which the property applies.

Package Complete AATFCS system::Camera
Public
System Complete AATFCS system
Extends Complete AATFCS system::Complete AATFCS System
End Complete AATFCS system;
System implementation Complete AATFCS system. Camera
Extends Complete AATFCS system::Complete AATFCS system. Basic
Subcomponents
AATFCS Control System: **refined to process** AATFCScontrolSystem::Camera:: AATFCS ControlSystem.camea;
Controlledsystem: **refined to system** SystemUnderControl::Camera::system_under_control.camera;
AATFCSPlatform: **refined to system** ExecutionHardware::Camera:: AATFCS Hardware.camera;
flows
TemperatureResponse: **end to end flow**
AATFCS systemUnderControl.Tempflow ->
SystemtoControllerConn -
AATFCS ControlSystem.ControlFlow ->
ControllertoSystemConn ->
AATFCS systemUnderControl.HeaterCmdFlow
{ Latency => 50 ms;};
properties
Allowed_Processor_Binding =>
reference mdsplatform.ground_processor **applies to** AATFCS ControlSystem.OperatorConsole;
Allowed_Processor_Binding =>
reference mdsplatform.ground_processor **applies to** AATFCS ControlSystem.GoalElaborator;
Allowed_Processor_Binding =>
reference mdsplatform.flight_processor **applies to** AATFCS ControlSystem.GoalExecutive;
Allowed_Processor_Binding =>
reference mdsplatform.flight_processor **applies to** AATFCS ControlSystem.StateEstimation;
Allowed_Processor_Binding =>
reference mdsplatform.flight_processor **applies to** AATFCS ControlSystem.DeviceControl;
Allowed_Processor_Binding =>
reference mdsplatform.flight_processor **applies to** AATFCS SystemUnderControl.Hardware Adapters;

Fig 9: Modeling of Processor Bindings

6. Conclusions

- This paper has presented an overview of the automated air traffic Flight control system andhasperformedanalyses using different systems architectural modeling methods. A description of the system was provided along with the many architectural views which are indicative of the Complex nature of the system.
- Several of the analyses presented in this paper are exemplary of the ability of models to reduce complexity and increase comprehension of the system architecture in an unambiguous manner. Models also serve to reduce miscommunication and can potentially reduce overruns in development cost, especially if the modeling activity is done sufficiently early in the program, as demonstrated by the spiral development model.
- The architectural analyses highlighted primary areas of interest in defining the AATFCS. The UML analyses presented examples of the system architecture from an object-oriented perspective: the use case, the class hierarchy diagram, and the sequence diagram.
- We presented AADL language for AATFCS system.

7. References

[1] “NASA &The Next Generation Air Transportation System (NextGen)” (n.d.), fromwww.aeronautics.nasa.gov/docs/nextgen_whitepaper_06_26_07.pdf.retrieved October 5, 2008.

[2] "Free flight (air traffic control)"retrievedfrom Wikipedia(FreeFlight):http://en.wikipedia.org/wiki/Free_flight_(air_traffic_control),retrieved October 21, 2008.

[3] Cureton, K. SAE 574 Lecture #1: Net-Centric Systems Architecting and Engineering. University of Southern California. (Aug. 26, 2008).

[4] Cureton, K.SAE 574 Lecture #1: Net-Centric Systems Architecting and Engineering. Universityof Southern California.(Aug. 26, 2008).

[5] "Fact Sheet – System-Wide Information Management (SWIM)", (May 2, 2006), retrieved fromGoogle (System-Wide Information Management):http://www.faa.gov/news/fact_sheets/news_story.cfm?newsId=7129October 23, 2008

[6] "OEP Plan Reference Sheet NNEW" (June 19, 2007)fromGoogle(NNEW):http://www.faa.gov/about/office_org/headquarters_offices/ato/publications/oep/version1/reference/nnew/retrieved October 23, 2008.

[7] Spitzer, C., The Avionics Handbook. CRC Press LLC. (Ed.). (2001).

[8] Wasson, C. System Analysis, Design, and Development: Concepts, Principles and Practices. New Jersey : John Wiley & Sons, Inc (2006).

[9] Rechtin, E. (1991). System Architecting: Creating and Building Complex Systems. New Jersey: Prentice-Hall, Inc. (1991).

[10] Hines, J. *Systems Engineering Theory and Practice, SAE 541, Session 1.* University of Southern California.(June 2, 2008)

[11] "Spiral model" retrieved from Google (spiraldevelopmentmodel):http://en.wikipedia.org/wiki/Spiral_model,Dec. 6, 2008

[12] "Spiral model" retrieved Dec. 6, 2008 from Google (spiraldevelopmentmodel):http://en.wikipedia.org/wiki/Spiral_modelDec. 2, 2008.

[13] Cureton, K. *SAE 574 Lecture #6: Architecture Modeling Concepts*. University of Southern California. (Oct. 14, 2008).
.

[14] Cureton, K. SAE 574 Lecture #7: Architecture Modeling Concepts. University of Southern California. (Oct. 28, 2008).

[15] Schmuller, J. Sams Teach Yourself UML in 24 Hours, Third Edition, Sams Publishing. (2004).

[16] Society of Automotive Engineers (SAE). "The Architecture Analysis and Design Language (AADL)." Society of Automotive Engineers (SAE) Standard AS-5506 (November 2004) Revised in January 2009 as AS-5506A.
http://www.sae.org/technical/standards/AS5506A.