Proceedings of the 1999 Winter Simulation Conference
P. A. Farrington, H. B. Nembhard, D. T. Sturrock, and G. W. Evans, eds.

TALUS – AN OBJECT ORIENTED AIR COMBAT SIMULATION

Sigurd Glærum

Norwegian Defence Research Establishment
Division for System Analysis
P. O. Box 25
NO-2027 Kjeller, NORWAY

ABSTRACT

TALUS (TAktisk LUftkampSimulering) is a discrete event, Monte Carlo based air combat simulation model developed at the Norwegian Defence Research Establishment (Norwegian acronym: FFI). The model was requested by the Royal Norwegian Air Force (RNoAF), but will mainly be used for operational research at the FFI. The development took place in a small, well integrated project with the authority to establish the requirements to the model. Much effort was dedicated to the establishment of a sound system development process, employing object-oriented analysis, design and implementation. The result has been a model that satisfies the initial requirements and verification of the fact that the time invested in a systematic approach to the system development is directly proportional to the quality of the end product.

1 INTRODUCTION

Engagement level air combat with multiple aircraft and aircraft types on both sides represents a highly complex and dynamic system. Analysis of such a system requires a well-balanced view of the questions that need to be answered and the tools that are used to answer them.

The background for the development of the air combat simulation model TALUS is the Combat Aircraft Analysis project that was carried out at the Norwegian Defence Research Establishment during the years 1993–1996. The purpose of this analysis was to evaluate a number of fighter aircraft - candidates for an acquisition by the Royal Norwegian Air Force which is to take place early in the next century. An important tool for the evaluation of the operational performance of these aircraft was an air combat simulation model, developed abroad in the seventies and eighties. During the analysis it became increasingly clear that this model – and most other current air combat models – was much too inflexible to meet the needs of post cold-war operational research. These old models usually combine an old programming language (e.g. FORTRAN), a procedural approach, hardcoded limitations, inferior documentation and an inflexible pilot model, and are therefore difficult to modify and extend.

Due to the limited availability of modern, flexible air combat simulation models, the decision was made that the FFI should develop such a model in-house. While the model was requested and financed by the Royal Norwegian Air Force, the specific requirements to the model were formulated by the project that would actually develop it. This made it possible to set up a very effective, unbureaucratic organization with clearly formulated and coherent goals. Three fundamental questions needed to be addressed:

- What questions should the model be able to answer?
- What properties should the model have?
- How to design and implement a complex model with limited resources?

The first question was relatively straightforward, given the needs arising from ongoing and future operational research at the FFI:

- What is the performance of a given aircraft type over a wide range of different air-to-air scenarios?
- What impact does subsystem performance have on overall system performance?
- What are the consequences of new technologies like sensor fusion, data link, stealth, anti-stealth etc.?
- What are the consequences of given tactics, rules of engagement, ID-requirements etc.?

The answers to the second question, which properties the model should have, were given partly by the answers to the

first, but mainly by the need to get a model that would be easy to use, modify and extend:

- A balanced model of engagement level air combat.
- A closed, Monte Carlo based discrete event simulation.
- User friendly, with a graphical interface.
- Well documented at all levels.
- Easily extendible to future requirements.

Given the limited resources available for the project, less than 10 man years in total, the answer to the last question – how to design and implement the model – was the most critical factor for a successful outcome. The standard way of system development in the field of military simulation – start in one corner, implement the bits and pieces and throw enough manpower at the problem until the whole colossus is tied together – was clearly not efficient. Instead, the project turned to the well established field of object oriented system development. It was decided to employ an iterative system development process:

- A detailed design using the object oriented design tool OOram Professional, (Numerica Taskon 1996) and (Reenskaug 1995).
- Implementation of the design using the object oriented programming language MODSIM III, (CACI Products Company 1997).
- Continuous documentation effort, at both the design and program code level.
- Milestones and testing.
- Iteration through the above points.

At the start of the project, much effort was put into the design phase of the development. This effort repaid itself later in the project by greatly reducing the risk of not being able to complete the development at all. With the overall structure of the model in place, it simply came down to implementing the necessary detail of any given module.

It must be stressed, however, that while much emphasis was put on a sound system development, a necessary prerequisite for a successful outcome is a good understanding of the problem domain. This had been achieved through the Combat Aircraft Analysis project and through continuos input from fighter pilots of the RNoAF.

2 MODEL DESCRIPTION

The area of interest for TALUS is primarily engagement level air-to-air combat, although no hardcoded limitations exist with regard to time, space nor the number of participating aircraft or aircraft types.

2.1 Scenario Definition

A scenario is defined through the graphical user interface to the model. Figure 1 below shows the main scenario

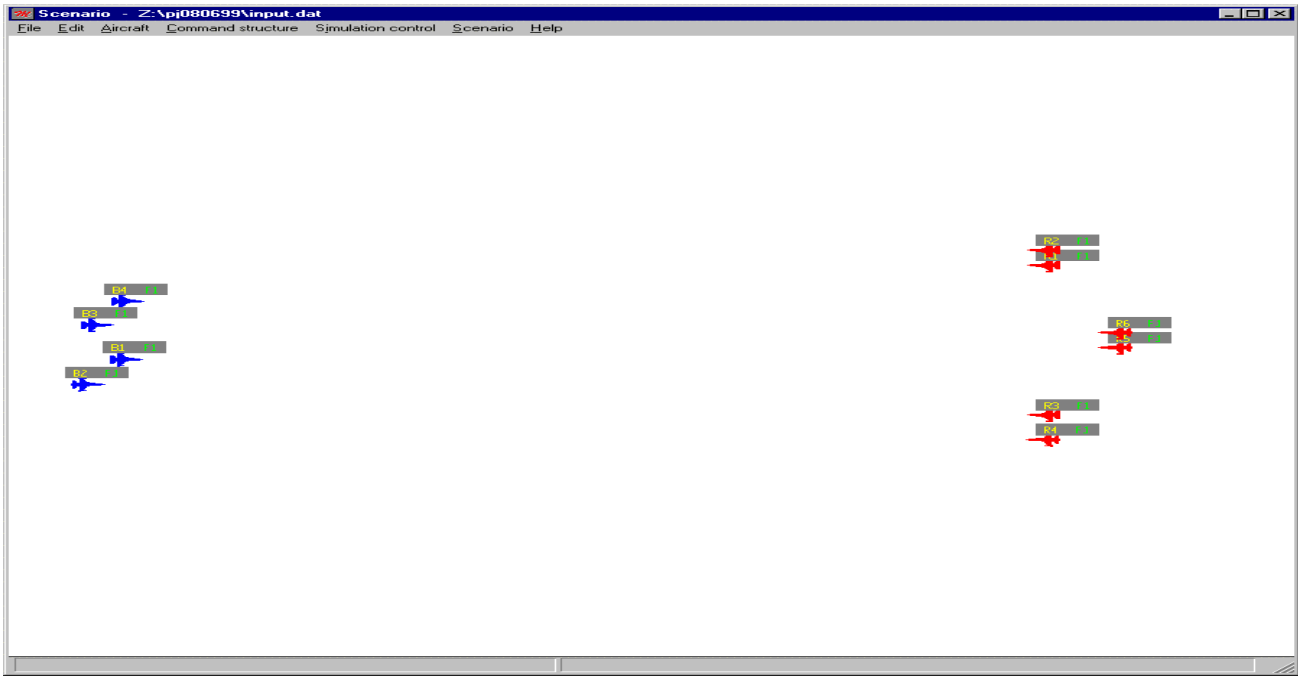

Figure 1: Scenario Definition Screen

definition screen, where individual aircraft are created and added to their respective formations. The initial geometry is defined either by dragging the individual aircraft to their desired starting positions, or by numerical input. The command and control hierarchy is defined in a separate screen. The hierarchy's smallest unit is the Section, which consists of a Section Leader (SL) and an optional Wingman (WM). The next level up, the Formation, consists of any number of sections, with one of the section leaders designated Formation Leader (FM). Furthermore, any number of formations can be combined to form a Mission, with one of the formation leaders designated Mission Commander (MC). One side may even consist of several missions, but there will be no coordination or communication between them. The hierarchy is depicted in Figure 2.

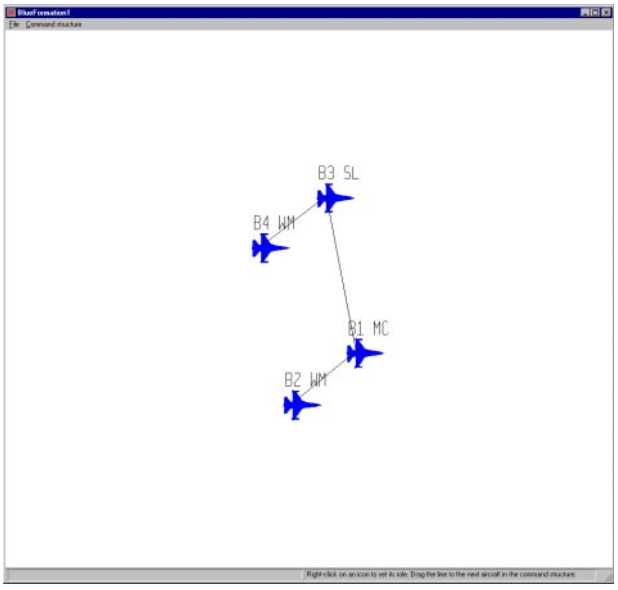

Figure 2: Command Hierarchy Definition

The main reason for implementing the command and control hierarchy was to facilitate communication and coordination between different aircraft and to allow flexible assignment of mission plans for groups of aircraft. Mission plans are defined for each formation and are either defined in detail through the interface or picked from a library of standard plans in the database. Each aircraft will read its designated plan and its pilot model will be responsible for its implementation, though responding to any coordination from the proper command element and/or any specific situation that requires a deviation from the predefined plans. There are three levels of plans built into the pilot model – which will be discussed below – it is the specific *parameters* of these plans that are defined through the user interface.

One major task for the user is to define all the different kinds of aircraft that are to participate in the scenario. These may either be predefined in an existing input file, picked from a database or composed by selecting the different subsystems for a given aircraft type from the database. Figure 3 shows one of the screens involved in defining a configuration. If the user so wishes, all aircraft in a scenario may be of completely different types - without any performance penalty.

Several other elements may be defined through the graphical interface – radars of different types, EW-systems, IRSTs and missiles, for instance. Aircraft types may also be specified to the required level of detail, with radar cross section, IR signature, drag coefficients, fuel consumption tables etc. All of these may be stored in the database for later use in other scenarios.

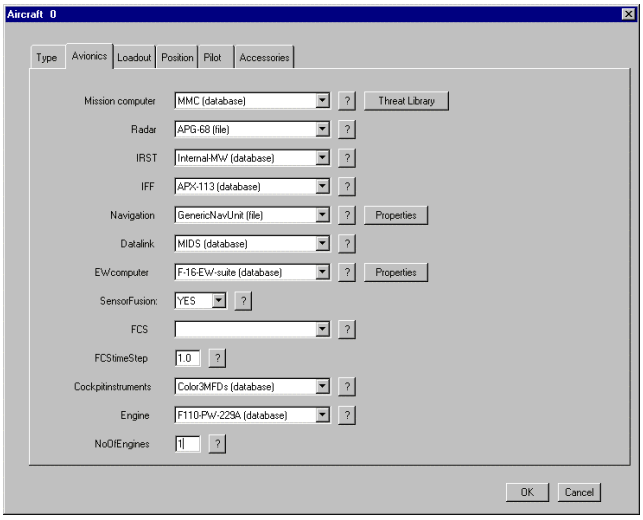

Figure 3: Aircraft Configuration Definition

2.2 Scenario Execution

A scenario, either given by an input file or defined (or modified) through the graphical interface is normally executed as a batch job, running through a number of Monte Carlo iterations in order to produce statistically valid results. For demonstration or debugging purposes the scenario can also be run with animated graphics. Figure 4 shows the resulting screen with five sub-windows (clockwise from top right):

- Current sensor plots and track files as maintained by the attack computer of the designated aircraft
- Current airpicture as perceived by the pilot of the designated aircraft
- History of important actions taken by the pilot of the designated aircraft
- Side view of the scenario
- Top view of the scenario (designated aircraft marked with yellow dot)

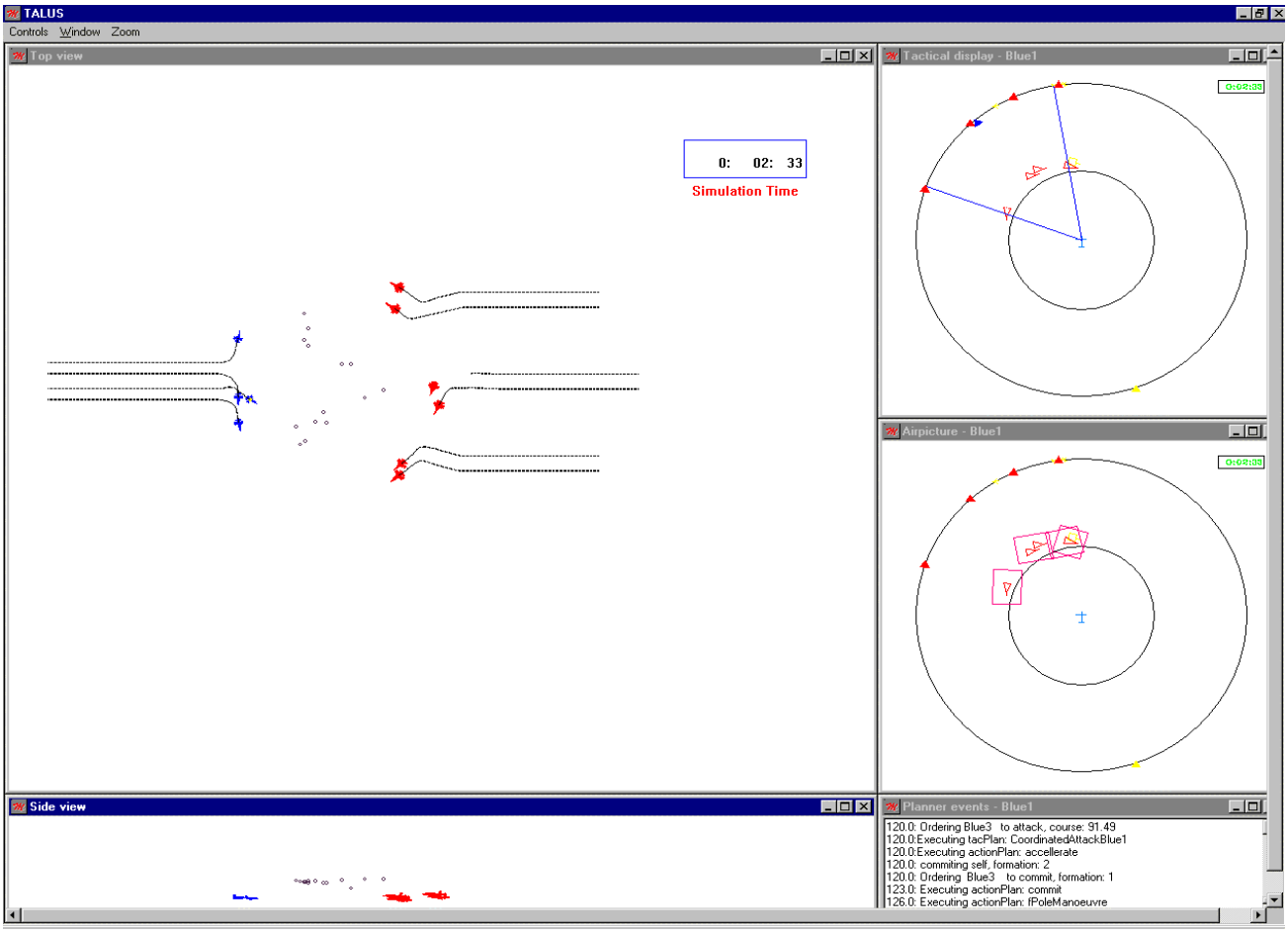

Figure 4: Scenario Animation

3 MODEL DESIGN

The real world of air-to-air combat is actually quite modular. Aircraft and missiles are discrete entities and they are themselves modular in structure, with subsystems like sensors, computers, cockpit displays pilot etc, all with formalized ways of communicating with each other.

This feature of the real system is readily exploited by the object-oriented design of TALUS, which exhibits a strong correspondence between real entities and objects of the model. Modularity is also enhanced by collecting objects that structurally and functionally belongs together – like the collection of objects that make up a radar – into program structures called Modules. Interfaces between objects change frequently, while interfaces between modules are relatively stable, simplifying work-share and reducing the need for constant synchronization between developers in both the design and implementation phase of the project.

A common mistake in object-oriented design is to start out trying to establish class hierarchies for the different objects in the model domain. In this way relationships like flyingVehicle → aircraft → fighterAircraft is created, each with more specialized functionality than its parent class. Through inheritance and reuse there is then the potential to save a lot of programming effort. The trouble with this approach is that it does not address the real problem of designing a simulation model, which is to identify the interactions between objects of *different* classes.

The concept of *role modeling* in the OOram design tool is centered on identifying these interactions and has been used successfully in designing TALUS. By creating role models for parts of the model that are reasonably independent, and by subsequent synthesizing of these role models, a complete design is created by a divide-and-conquer technique. The end result is a design where the roles translate directly to the objects of the model.

4 MODEL STRUCTURE

The main object within the simulation is the aircraft object, which again contains objects making up its component

parts. This is the only "living" object with an autonomous existence inside the model (apart from an airborne missile). Other, higher level, objects exist, but their presence is in general only for initialization and/or housekeeping purposes. Even the command and control hierarchy does not have an independent existence, but resides in the "mind" of each pilot instance – a parallel to the real world. A multitude of lower level objects exist, but they all reside in a "part-of" hierarchy with the aircraft object at the top. Figure 5 shows a simplified picture of this hierarchy. Note that not all subsystems need be present in a given aircraft instantiation (e.g. not all aircraft have an IRST).

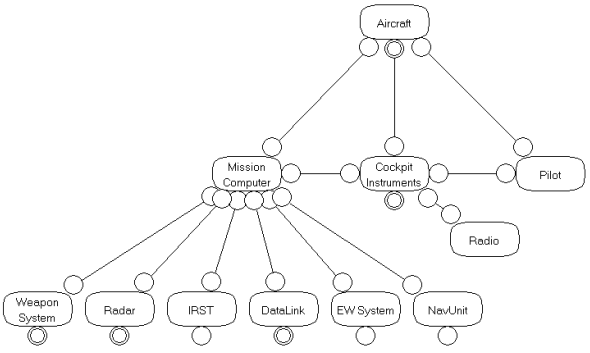

Figure 5: OOram View of the Aircraft Model

The main flow of events in the simulation is in two directions in this tree-structure. One direction is initiated by track updates/detections reported by subsystems like the radar, IRST or data-link – with simulated inaccuracies. These track-files are organized and possibly fused by the mission computer and relayed to the cockpit instruments. If the pilot is scanning his instruments at the time (or the tracks are displayed on the HUD/helmet mounted display), the track updates are sent to the pilot object, which will integrate it into its current airpicture. The downward flow of events is initiated by the pilot's planning process and subsystem management actions resulting from this process. These are routed through the cockpit instruments and the mission computer to the subsystem in question.

In TALUS, particular emphasis has been put on a realistic representation of the data content and accuracy in this message flow. A sensor will, for instance, only report a target position with a given accuracy, and target ID is unknown unless that particular sensor is capable of target identification in the given circumstances.

It is important to note that any instance of the aircraft object (including its subsystems) is a completely self-contained and autonomous entity within the simulation. There is no need for an explicit global clock or time-stepping mechanism to drive the simulation forward. In effect, the different aircraft instances executes in parallel using their own internal, local clocks to schedule events like position updates etc. This is also true for some of the subsystems. A radar will, for instance, schedule an update of itself when the time for a complete scan of its search volume has elapsed. This time management scheme is very flexible since it allows local control of update cycles. A radar will automatically adjust its update frequency after a mode change, while a missile will increase its update frequency when it gets close to a target. There are also mechanisms allowing for external interrupts of update cycles (or any other scheduled event). A sensor taking a peek at another aircraft will, for instance, interrupt the regular cycle of this aircraft to force an update ahead of schedule – in order to make the aircraft able to return an updated position.

The time management scheme described above is a feature built into the MODSIM III language and will – when used to its full extent – increase the modularity and eliminate the need for complex synchronization schemes. Also in TALUS there is, of course, synchronization of events against a global simulation clock, but this is managed by the compiler and hidden from the developer. An additional benefit of the MODSIM III language is that it gives the user the choice of running a simulation either in compressed time (discrete event) or synchronized against wall-clock time. This is selectable through the graphical user interface.

5 PILOT MODEL

In the introduction it was stated as a goal that the model should be *balanced*. The reason is previous experience with imbalanced simulation models – particularly in the field of military simulation. Imbalance occurs when certain parts of the real world system are modeled in great detail, while other significant parts receive just a cursory treatment. This may be due to a simple lack of understanding of the real system, or it may be more psychological in nature. It is, after all, a lot easier to model systems and processes that are thoroughly understood than systems that are not. It is relatively easy to model a radar warning receiver in minute detail and a high degree of accuracy, but it is *not* easy to model a human operator to the same degree of fidelity – yet the latter 'system' is much more important to the outcome of an air-to-air engagement.

Considering the sorry state of human behavior representation in military simulation, it was decided to devote a significant part of the development effort to build a credible pilot model. An overview of the pilot model is given below.

5.1 Main Design Principles

The pilot model is designed to mimic the human cognitive and decision making process to some extent. Figure 6 depicts the major roles (objects) involved in this process.

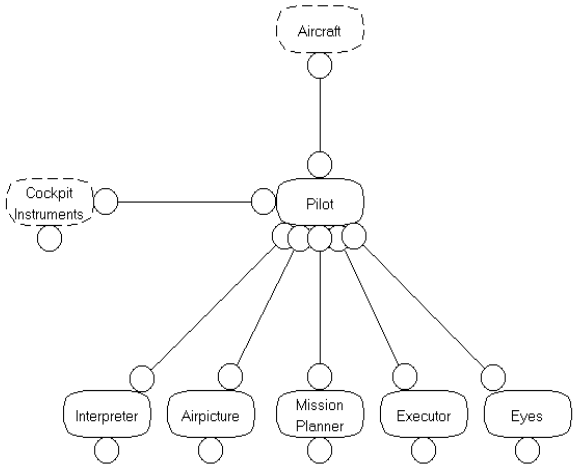

Figure 6: OOram View of the Pilot Model

The main tasks of the pilot model are twofold. One is to interpret all sensor inputs, radio messages and system status data and integrate it into his perception of the airpicture. The other task is to – based on the airpicture, pre-laid plans and coordination with other pilots – plan ahead and execute the resulting decisions.

All inputs to the pilot are routed through the main container object in the pilot model, the pilot object. Data relating to the airpicture is forwarded to the interpreter object, while data relating to coordination of the planning process is forwarded to the missionPlanner object.

5.2 The Interpreter

The task of the interpreterObj is, on a single-track basis, to convert the track to a more "humane" format and to evaluate the threat levels associated with this track. If the mission computer of the aircraft does not have a sensor fusing capability, a rough sensor fusion will be performed in the interpreter. For some tracks which do not pass through the mission computer – tracks originating from a radio message or from the pilot's own eyes – the sensor fusion will be performed regardless. Figure 7 shows the design of the interpreter.

After treatment by the interpreter, the airpicture is asked to integrate the new or updated track.

5.3 The Airpicture

The task of the airpicture is to maintain an updated situational awareness by synthesizing the continuous track updates received from the interpreter. It will organize the single tracks into formations and identify their higher level characteristics. It will also, when requested by the planner, give an assessment of the current situation by comparing the airpicture to a library of templates. Figure 8 shows how the airpicture is organized.

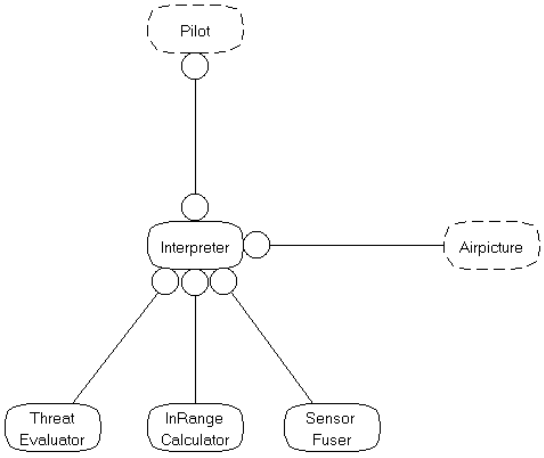

Figure 7: OOram View of the Interpreter

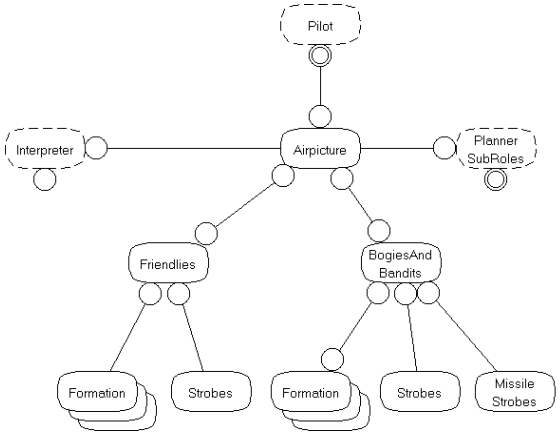

Figure 8: OOram View of the Airpicture

The formations are formed and updated based on the characteristics of the tracks they consist of. A correlation of target geometries, identities, types etc., may lead to either the creation of a new formation, a formation split or a formation merge. Furthermore, the airpicture has a certain 'memory' span. Even if all targets in a formation have been deleted, the formation will be maintained and extrapolated until it is decided to delete it altogether. As in the rest of the pilot model, there are built-in features that simulate the limits of human capacity, both with regard to the amount of data that can be integrated and the latency involved in decision-making and execution.

5.4 The Planner

The planner has a three-level organization. The topmost level is the missionPlanner which is responsible for carrying out the overall mission plan defined for this pilot in the user defined input. The next level down is the tacticalPlanner which consists of a library of intermediate-term tactical plans. These plans are objects derived from a

generic tactical plan with default behavior which they in turn override and extend as necessary. Examples here are tactical plans like: Combat Air Patrol, Intercept, Attack, and Return To Base. At the bottom level there is a library of short-term action plans, again with objects derived from a generic action plan, overriding and extending it. Examples here are action plans like: Cruise, Commit, F-Pole Maneuver, Missile Avoidance etc. Figure 9 shows a simplified overview of the full planning structure.

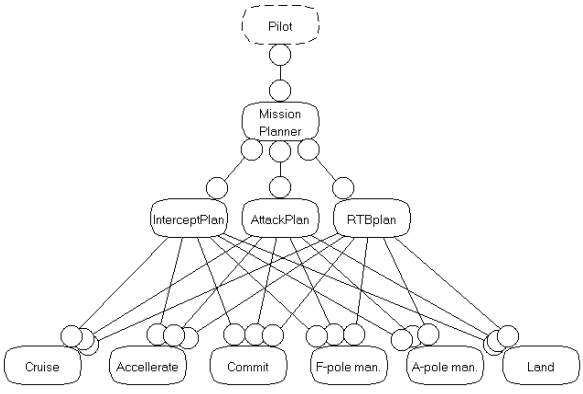

Figure 9: OOram View of the Planner

The three levels all operate on different update frequencies, the mission planner typically on a 5-10 second update cycle and the bottom-level action planner on a 1 second or less update cycle.

The mission planner will in effect execute a user-defined script of tactical plans and will oversee the transition between those plans according to the mission plan, the situation assessment or orders received from superior pilots.

A tactical plan is similarly based on a default, user-defined, script of action plans and will oversee the execution and transition between the action plans. It is important to note that these scripts are not set in stone. A pilot may have several flavors of a given tactical plan, to be used for different parts of the mission.

An action plan deals with short-term tasks like target-sort, sensor management, weapon management, aircraft maneuvering etc.

In order to allow for unforeseen or critical events, shortcuts have been incorporated into the planning structure. If, for instance, an incoming missile has been detected, the airpicture will raise a flag and the planner will suspend all other activity and execute the action plan Missile Avoidance. Another example is the reception of an order from a superior pilot, which will lead to an overriding of the standard script.

All in all, the planning structure has proven to be very flexible and easy to extend. New behavior patterns are implemented by inheritance of the generic plan objects and the amount of modification needed is usually both limited and localized.

5.5 The Executor

The executor is responsible for implementing the decisions taken by the planner. It interfaces to the cockpit instruments and flight control system and coordinates the different actions requested by the planner. It will also introduce action-specific delays in order to simulate human behavior.

6 FUTURE ENHANCEMENTS

While TALUS is presently complete to a certain degree as an air-to-air engagement level combat simulation, it will certainly be extended in the future. These augmentations will be implemented not on a nice-to-have basis, but as requirements appear. In the medium-term, however, some developments seem likely:

- Distribution of the model to operational squadrons to be used for evaluation of tactics.
- Interfacing the simulation to the Royal Norwegian Air Force F-16 simulators, to act as computer generated forces. The interface would be through HLA and the RPR-FOM.
- Incorporation of AEW (like AWACS) and GCI (Ground Controlled Intercept) into the model. This would be a relatively straightforward extension, given the command and control structure already present.
- Introduction of air-to-ground specific capability. Some aspects of this are present already, but could be extended to include SAM-sites, digital terrain, weapons delivery etc.

CONCLUSIONS

The main lesson to be drawn from the development of TALUS at the FFI is that in spite of limited resources of manpower and time, a full-blown, quite complex, simulation model can be successfully completed by employing sound system development methods. By using the right object-oriented design tools significant gains in terms of model structure and reduced implementation effort are achieved. Adding to that an object-oriented programming language that does not require developers to be software engineers, but rather scientists with an understanding of the problem domain, an integrated, efficient system development can be achieved. Since everybody is involved in analysis, design and

implementation, there is less room for misunderstandings and discontinuities in the process.

Another lesson is that a small group of people may be equally, or even more, productive than a larger group. A large system development group would have to employ a more rigid and bureaucratic approach to the various phases of the development, with strict adherence to time-lines, version control etc. While these aspects are important also to a smaller group, they can be treated in a more dynamic way.

Since the model is not to be fully completed until the end of '99, it has not yet been through any full-scale tests or employed for serious analysis. Preliminary tests using data and scenarios from the Combat Aircraft Analysis does, however, produce results that comply with expectations.

ACKNOWLEDGMENTS

The development of TALUS was partially funded by the Royal Norwegian Air Force and fighter pilots from the RNoAF made substantial contributions to the modeling effort.

In addition to the author, the following scientists at FFI were involved in the development of TALUS: Geir Edvardsen, Ottar Graasvoll, Lars Hovik and Jan Erik Torp.

REFERENCES

CACI Products Company 1997, *MODSIM III, The Language for Object-Oriented Programming, Reference Manual*, CACI Products Company, La Jolla, CA (http://www.caciasl.com).

Numerica Taskon AS 1996, *OOram, Object-Oriented System Development*, Numerica Taskon AS, Oslo, Norway (http://hotell.nextel.no/numerica/).

Reenskaug, T. 1995. *Working With Objects, The OOram Software Engineering Method*, Manning Publications Co., Greenwich, CT.

AUTHOR BIOGRAPHY

SIGURD GLÆRUM is a Senior Scientist at the Norwegian Defence Research Establishment. He holds a Ph.D. in Mathematical Modeling and a M.S. in Numerical Analysis, both from the University of Oslo. He is currently project leader for the TALUS project at the FFI, which is tasked to develop the TALUS model and to support the RNoAF in its acquisition of new combat aircraft.