NASA-CR-192920

IN-04-CR
153619
180

UNIVERSITY OF CENTRAL FLORIDA

DEPARTMENT OF INDUSTRIAL ENGINEERING AND MANAGEMENT SYSTEMS

FINAL REPORT
ON
NASA GRANT NUMBER NAG2-625

Design of an Air Traffic Computer Simulation System to Support Investigation of Civil Tiltrotor Aircraft Operations

March 15, 1993

[illegible]
Ralph V. Rogers, Ph.D.
Principal Investigator

Department of Industrial Engineering and Management Systems
University of Central Florida
P.O. Box 25000
Orlando, FL
32816

(NASA-CR-192920) DESIGN OF AN AIR TRAFFIC COMPUTER SIMULATION SYSTEM TO SUPPORT INVESTIGATION OF CIVIL TILTROTOR AIRCRAFT OPERATIONS Final Report (University of Central Florida) 180 p

N93-26052

Unclas

G3/04 0153619

TABLE OF CONTENTS

INTRODUCTION

The potential introduction into the National Airspace System (NAS) of common carrier commercial operations using high speed, heavy lift vertical flight aircraft such as the civil version of the V-22 Osprey tiltrotor created a need for computer simulation models to investigate the impacts of such technologies on the existing and proposed air transportation infrastructure. However, the fundamental paradigms and methodologies of the current air traffic simulation models do not directly support the broad range of operational options and environments necessary to study the issues associated with the tiltrotor's and other vertical flight aircraft's potential introduction. As a result, the National Aeronautical and Space Administration (NASA) and the Federal Aviation Administration (FAA) awarded the Department of Industrial Engineering and Management Systems of the University of Central Florida (UCF) a research grant (NASA Grant No. NAG2-625) to develop a software design for computer simulation models with which to investigate tiltrotor aircraft operation impacts on existing and proposed air transportation infrastructures. The subsequent research project was begun in May 1990 at UCF under the name Tiltrotor Air Traffic Simulation System (TATSS) Project. Funding for the TATSS Project ended August 30, 1992. This document is the Final Report for the TATSS Project.

TATSS PROJECT MODEL OBJECTIVES

The TATSS Project's goal was to develop a design for computer software that would support the attainment of the following objectives for the air traffic simulation model:

- Full freedom of movement for each aircraft object in the simulation model. Each aircraft object may follow any designated flight plan or flight path necessary as required by the experiment under consideration.
- Object position precision up to ± 3 meters vertically and ± 15 meters horizontally.
- Aircraft maneuvering in three space with the object position precision identified above.
- Air traffic control operations and procedures.
- Radar, communication, navaid, and landing aid performance.
- Weather.
- Ground obstructions and terrain.
- Detection and recording of separation violations.
- Measures of performance including deviations from flight plans, air space violations, air traffic control messages per aircraft, and traditional temporal based measures.

TATSS PROJECT ACHIEVEMENTS

The major achievements of the TATSS Project were:

- Identified the underlying paradigmatic and technical software implementation issues inhibiting current air traffic simulation modeling paradigms and methods.
- Identified simulation modeling paradigms and strategies as well as software development environments which support attainment of the TATSS Project air traffic simulation model objectives.
- Developed a conceptual software design to support the air traffic simulation model objectives based on the modeling paradigms and development environments identified by the TATSS Project.
- Implemented prototype constructs of the TATSS conceptual software design.
- Implemented simple, abstracted air traffic simulation model and situation scenario with TATSS prototype simulation constructs.
- Demonstrated proof of concept for TATSS conceptual software design.

These achievements are discussed in further detail in the following sections.

THE UNDERLYING AIR TRAFFIC SIMULATION ISSUES

The activity of simulation involves the creation of a model to serve as a behavioral analog for phenomena under consideration. Specifically, simulation attempts to unfold the state changes of a system over time. In this sense, simulation is not a particular kind of model. Simulation is more accurately and usefully considered as way of using a model. The popular term "simulation model" reflects any model designed for use in this way [Rothenberg, et al, 1989]. Thus, in principle, simulation (or simulation modeling) represent a broad-based systems analysis philosophy and methodology capable of supporting diverse inquiry goals.

In practice, unfortunately, the analyst's traditional view of what constitutes the simulation activity is severely limited. This traditional view sees simulation as the process of building behavioral models, setting up some initial configuration, and then exercising the model to see what happens [Rothenberg, et al, 1989]. Simulation so viewed is valued for its ability to address the question "***What if...?***" ***What*** would happen ***if*** a system having the given behavior were to proceed from the given initial state. This provides the appealing quality of showing how a system evolves under certain assumptions and conditions. Rothenberg refers to this approach as the "toy duck" approach ("wind it up and see where it goes"). Such an approach is most appropriate when applied to situations requiring a choice of one of a small number of alternatives.

Traditional simulation modeling specifies only what actions to take, based on the current situation. They contain no ***explicit*** description of why the actions are necessary. Perhaps most significantly, they contain no depiction of the reasoning process that leads to an action and no explicit notions of causality or other relationships among events [Rothenberg, *et.al.*, 1989]. As a consequence, these simulations cannot answer questions that require interpreting or reasoning about knowledge. For example, traditional simulations are generally incapable of explaining **why** a given sequence of events occurred. Additionally, they cannot answer definitive questions such as "Can this event ever happen?"

To illustrate the limitations of traditional simulation, consider the following scenario: New tiltrotor aircraft are entering the US. commercial air carrier fleet. This new class of aircraft is opening air service to many heavily congested as well as isolated areas. New satellite navigation systems are also becoming available for aircraft use. Their coverage is to the surface world-wide. The accuracy and precision of this system are high and well established. Several airlines and some private aircraft are equipped with the new system. All the new vertical flight aircraft are equipped with this new navigational technology. Additionally, current VOR/DME/ILS systems are still the primary approved air navigation systems in effect and used on all IFR aircraft even those with the new satellite navigation system. The general accuracy and precision of these systems are also well known as well as their areas of coverage. Other sensors such as radar and secondary radar (e.g. mode S) are available to the air traffic controller. These sensors' areas of coverage and accuracy are also known. Additionally, the pilot of each aircraft can report their position ***as they understand it*** to the controller. The controller, likewise, can report the location of the aircraft to the pilot as the ***controller understands it***. The question of interest is: For a given operational environment, what mix of circumstances including weather, aircraft, navigation equipment, and air traffic loads can give rise to collision between aircraft and between aircraft and ground obstructions such as terrain? Associated with this question are the goals of establishing the air space policies for mixing vertical flight and fixed wing IFR operations and for establishing rules/procedures for ***information fusion*** from all the possible sources of information for both pilot and controller.

Implementing this illustrative scenario using the approaches of traditional simulation practice, especially discrete-event approaches, would be very difficult if not impossible. Reasons for the difficulty of current simulation approaches to support implementing phenomena and inquiry goals represented in the scenarios above are found in the major demands such phenomena make on the basic modeling philosophies of traditional simulation. These major modeling demands are:

- ***Paradigms*** other than networks.
- ***Explicit knowledge representation*** .
- ***Autonomy***.

These demands reflect the problems facing simulation modeling in general and air traffic simulation, in particular. The following sections further define the concepts of paradigms, knowledge, and autonomy as they pertain to simulation and their importance to air traffic simulation modeling.

PARADIGMS.

Intrinsic in the traditional modeling paradigms of simulation and modeling is the abstraction process. Process may be simply defined as the transformation of input into output. The transformation typically concerns matter, energy, or information. Simulation's most fundamental modeling paradigm is a further specialization of the process paradigm. This further specialization is the network or graph theoretic models of systems including digraphs, directed graphs, Gert networks, decision trees, petri nets, neural networks, etc. Still further specialization is reflected in classical discrete-event simulation models such as those used in typical ATC simulation models (e.g. SIMMOD, NASPAC, etc.). In such traditional discrete-event simulation, the underlying modeling paradigm is queuing networks based on the concept of queuing processes (i.e. waiting for service).

This concern for the underlying modeling paradigms in simulation is an important consideration to simulation efforts because the attributes of the primitive constructs of the underlying system paradigm are the basis for representing all modeled phenomena. All higher ordered modeling constructs must eventually be based on the underlying paradigm's primitive constructs. Therefore, the choice of the basic paradigm and its primitive construct strongly affect the expressive power of the simulation modeling scheme and any software products based upon that scheme. Any attempt to model the phenomena such as that described in the previous illustrative scenario must begin with the basic paradigm of the modeling perspective. To illustrate the effects of the underlying paradigm on phenomena modeling, consider the following description of the SIMMOD airspace model (emphasis added):

> *"SIMMOD airspace is composed of an* ***interrelated network*** *of aircraft routes. These routes are defined by the analyst as a series of* ***node and links****. When two or more routes converge, some node and links will be held in common; that is to say, they will appear in the definition of more than one route.*
>
> *All aircraft move in the airspace along these routes, and every flight entering the simulation* ***must be assigned to a route*** *in the input data. As an aircraft moves through the airspace, separation requirements are checked between it and other aircraft on the same path, merging paths, and crossing paths.*
>
> *Unlike actual flights, aircraft in the simulation* ***cannot deviate from their designated paths****. This being the case, vertical and lateral* ***separation are not checked by the simulation****. These separations requirements are maintained insofar as routes are correctly defined by the user with vertical and lateral separation.*
>
> *Each node on a path is given an altitude by definition; the simulation uses the altitude to calculate fuel consumption and speeds not given as true airspeed . Altitude is not checked or adjusted by the simulation to resolve conflicts.*
>
> [**SIMMOD**: The Airport and Airspace Simulation Model
> **REFERENCE MODEL**
> **September, 1989]**

In the above reference, one can readily see the impact of networks and queuing theory on SIMMOD's airspace simulation models. Similar effects may be seen in other simulation languages and environments (e.g. SIMON, SLAM, GPSS, etc.). Much of the phenomena of interest identified in the earlier illustrative scenario (e.g. deviation from flight paths due to navigation/pilot errors and possibility of collisions due to faculty policies) and the goals for the TATSS project simply cannot be represented using queuing networks or their derivative simulation environments.

This is not to say that airspace and air traffic simulation models based on the queuing network paradigm are not useful or that SIMMOD is not a useful and powerful tool. Such approaches have proved extremely useful for aiding decision making associated with many of the issues of air transportation. However, the increased complexity in the issues resulting from new aviation technologies such as satellite navigation and vertical flight cannot be adequately represented with only network based modeling paradigms and methodologies. The functional primitives of the network approach cannot reflect all the concepts and phenomena required to meet TATSS project goals as represented in the illustrative scenario. The network based paradigms reflect a too narrow a view of air traffic operations. Airspace and air traffic simulation requires the integration of more diverse modeling paradigms into its interpretation of phenomena. Specifically, paradigms are required which are capable of representing phenomena such as sensor detection, collisions, terrain, weather, human decision making, modularity, decomposability, and autonomy.

KNOWLEDGE

A major part of the simulation model development process is identification of operational and decision-based knowledge. However, discrete-event simulation has only been effective when "what shall we do next'" can be reduced to some ***simple*** rule that can be easily embodied in the simulation software model. For example, in air traffic modeling, a simple rule would be "wait at intersection node until airway link is not occupied." The modeler may typically identify more complex and realistic decision rules during his model development than those that actually find their way into the simulation model. Such knowledge is typically omitted because traditional simulation models and environments are not able use it. Much of the acquired knowledge which does find its way into the *lumped* model (see Ziegler, 1976) is represented only implicitly. This implicit representation is eventually reflected in the decision rules *programmed* into the simulation software model.

Countering this practice requires explicitly representing the knowledge bases of the elements of the decision model itself. For example, the information fusion processes of the pilots and the controllers of the illustrative scenario represent independent knowledge bases. In these instances, pilots and controllers (even individual pilots and controllers with differing competency levels) may operate under their own direct knowledge bases within a single model. Explicit representation of modeling and phenomena knowledge increases the modularity and decomposability of the resulting model. Modularity increases by separating the knowledge base associated with the model and the model's phenomena (e.g. pilots) decisions from the modeled systems structure. For example, the rules which constitute the knowledge base for data fusion by pilots could be treated separately from the airspace model of pilots flying aircraft and fusing data. If new knowledge base rules are required, no change is necessary in the airspace model structure or to the he associated airspace model element, only in the knowledge base.

AUTONOMY.

Within the domain of interest, aútonomy is associated with phenomena which possess full independence of movement and decision making. Autonomous phenomena may alter their temporal and spatial goal trajectories conditional upon *their* evaluation of their own current state and the state of the rest of the system *as they know it*. Autonomous phenomena includes the concepts of state variables continuously changing with time. In simulation modeling, the functional primitive construct *autonomous objects* embodies the concept of autonomous phenomena. This fundamental concept of autonomy as well as the autonomous object construct are missing from most discrete-event simulation models and environments. The reasons for the absence of autonomy is due to both the underlying network modeling paradigms and the related issues associated with implementing simulation models via computer software.

Typical implementations of airspace and air traffic simulation modeling use the asynchronous (i.e. event driven) discrete-event simulation strategy. In the asynchronous simulation implementation strategy, the next scheduled event (i.e. state change) defines the next increment of time that advances the simulation clock (i.e. simulation time). Events may occur at anytime. Thus, objects (or phenomena) may have their states updated at different times and the resulting increments between time advances may vary widely through the course of a simulation exercise. However, a common reference is still required to identify and resolve resource conflicts and other interactions between entities. Such coordination of object interactions and dependencies in an asynchronous based modeled system requires specific operational decision points.

The network modeling paradigm readily correlates with this implementation strategy. Nodes represent decision points and arcs represent specific distances and/or times. Conflicts are resolved at the nodes. Thus, the frame of reference for asynchronous modeling is the fixed network or similar simulation construct. The specific operational decision points in this frame of reference are network nodes or similar simulation constructs corresponding to fixed points in model space.

That is, the asynchronous solution to phenomena synchronization is to fix the decision points in model space thereby fixing the spatial increments of the model. Conditional decision are normally triggered by the arrival of objects or entities to some point in the model space (i.e. they are spatially or position triggered). However, conditions referenced to simulated time may be difficult or impossible to implement correctly in an asynchronous model because the model may not recognize that the condition is true until the next event occurs. By that time, the condition may be false again .

Figure 1 provides a graphical illustration of the corresponding asynchronous simulation of two objects moving on a two-dimensional plane. Note that the trajectory paths are explicitly defined as part of the model. The coordination point for the two objects is point X_5,Y_5 of X-Y plane. In

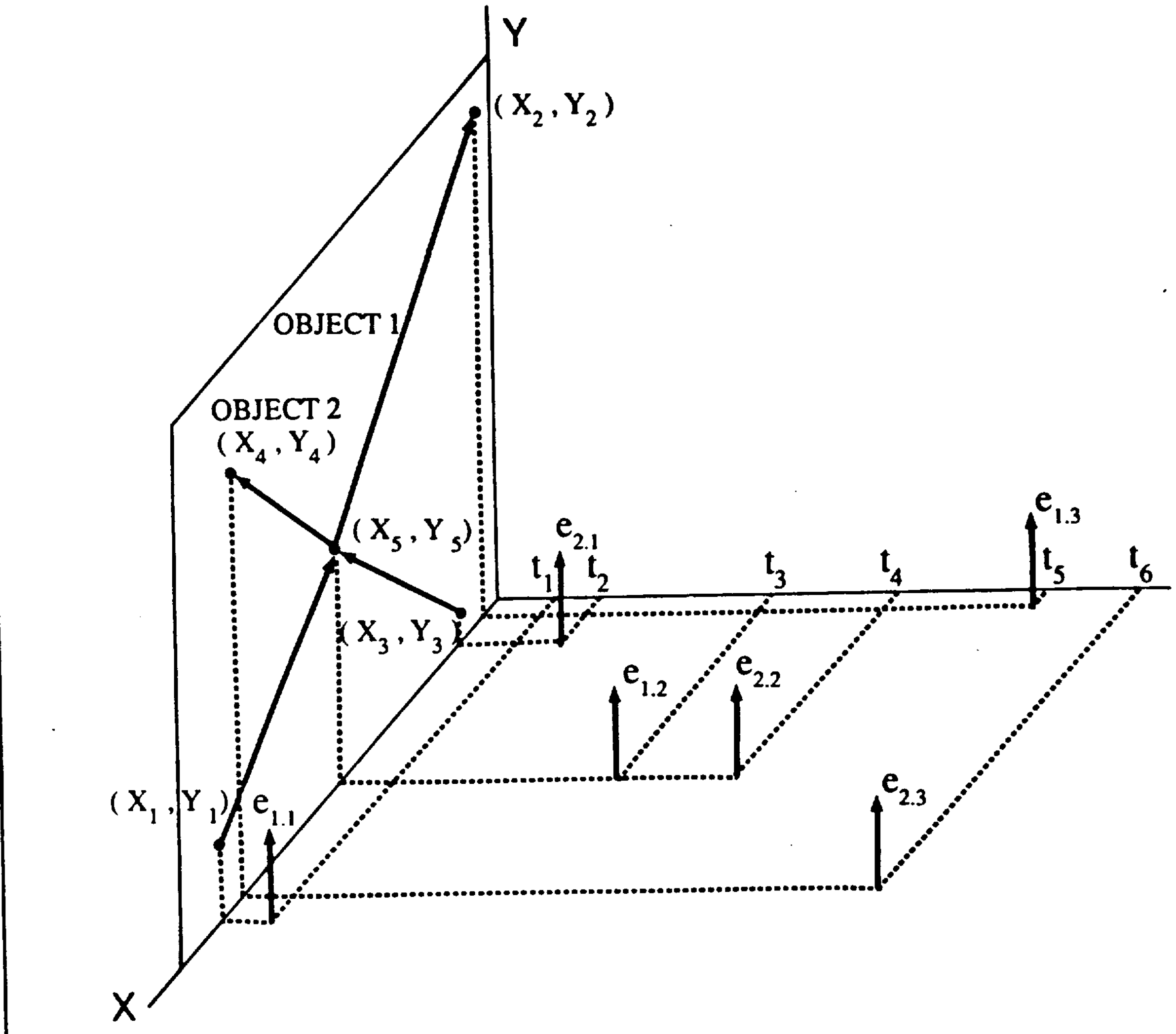

Figure 1. Asynchronous simulation of two moving objects.

this case, the decision or control logic must only consider the objects and interactions associated with and prior to events $e_{1,2}$ and $e_{2,2}$.

Figure 1 also illustrates the difficulty confronting autonomous object implementation in asynchronous simulation. Autonomous objects by definition must have the freedom to modify their trajectory through model space, to go where their goals and operational rules specify. However, to synchronize the actions of these or any objects, some common reference frame must

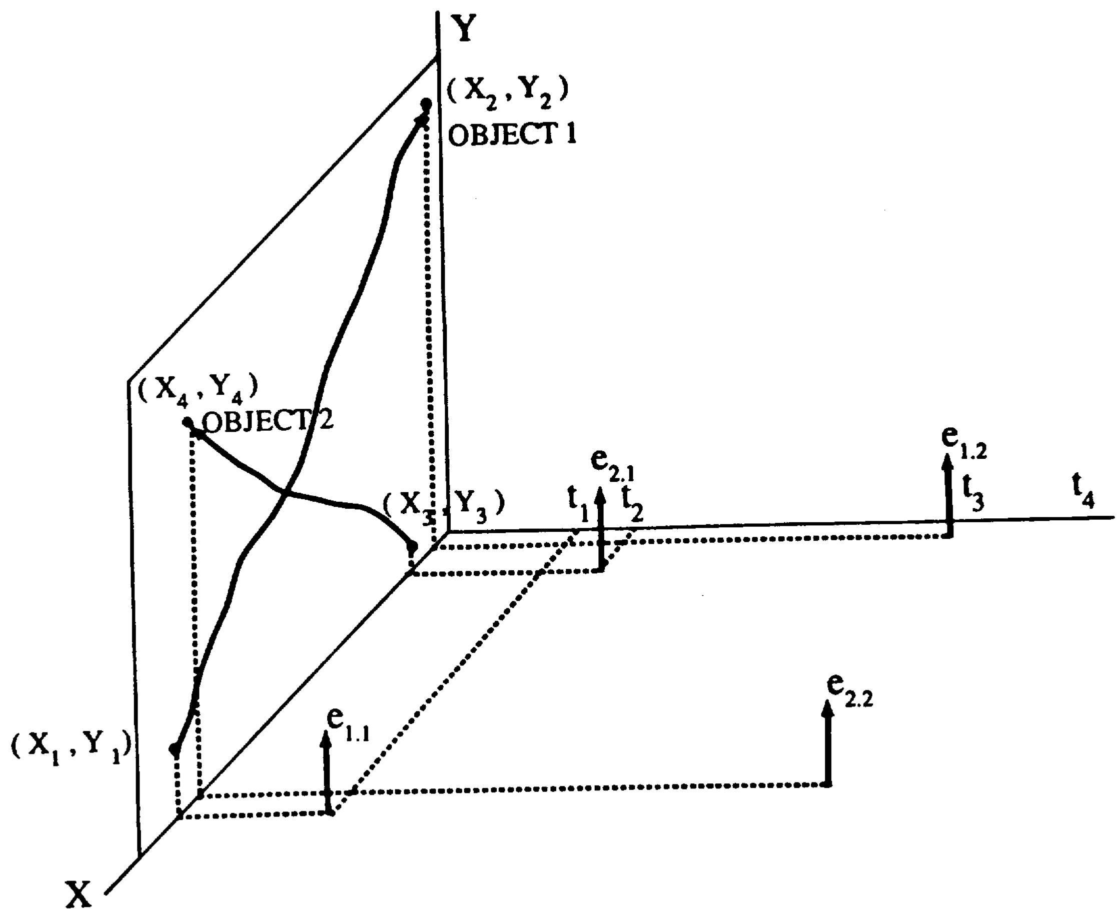

Figure 2. Two autonomous objects in event driven simulation.

be established. Figure 2 illustrates this problem for autonomous objects in an event driven simulation. In this example the objects schedule their next event (i.e. an arrival) some time in the future (t_3 and t_4). However, with no common points defined for them in the model in either time or space, there is no mechanism for synchronization.

To escape the limitations of fixed spatial increments associated with asynchronous simulation and in attempts to obtain more autonomy, discrete-event simulationist have often tried the synchronous (i.e. time driven) simulation implementation strategy. In the synchronous simulation implementation strategy, time is advanced in fixed increments. The model is examined only at regular intervals defined by the time increment used. At every advance of the clock, the state of each object, entity and process in the simulation must be updated. Conflicts and resolution must be identified and actions implemented. Conditional operational decisions are made (and synchronized) at these fixed time intervals. The reference for synchronization of actions is the common time defined by the time increment. Synchronous modeling is an advantageous approach when it is desired for a certain event to occur when a particular condition is satisfied (or identified)[Bratley, *et al,* 1987]. Figure 3 provides a graphical illustration of the synchronous approach for two autonomous objects moving through a plane. Notice that each object must evaluate its relationship to the other object at each clock increment Δt.

Synchronous modeling has four major disadvantages. First, such models are hard to implement in software. Second, non simultaneous events may be treated as simultaneous causing priority and sequencing problems. Third, to obtain sufficiently accurate performance measures, it may be necessary to make Δt very small. As Δt decreases, the number of times the model must be updated (and objects synchronized) increases. This increased updating also increases the computational resources and time required to exercise the model. Fourth, results may vary depending on the Δt time increment.

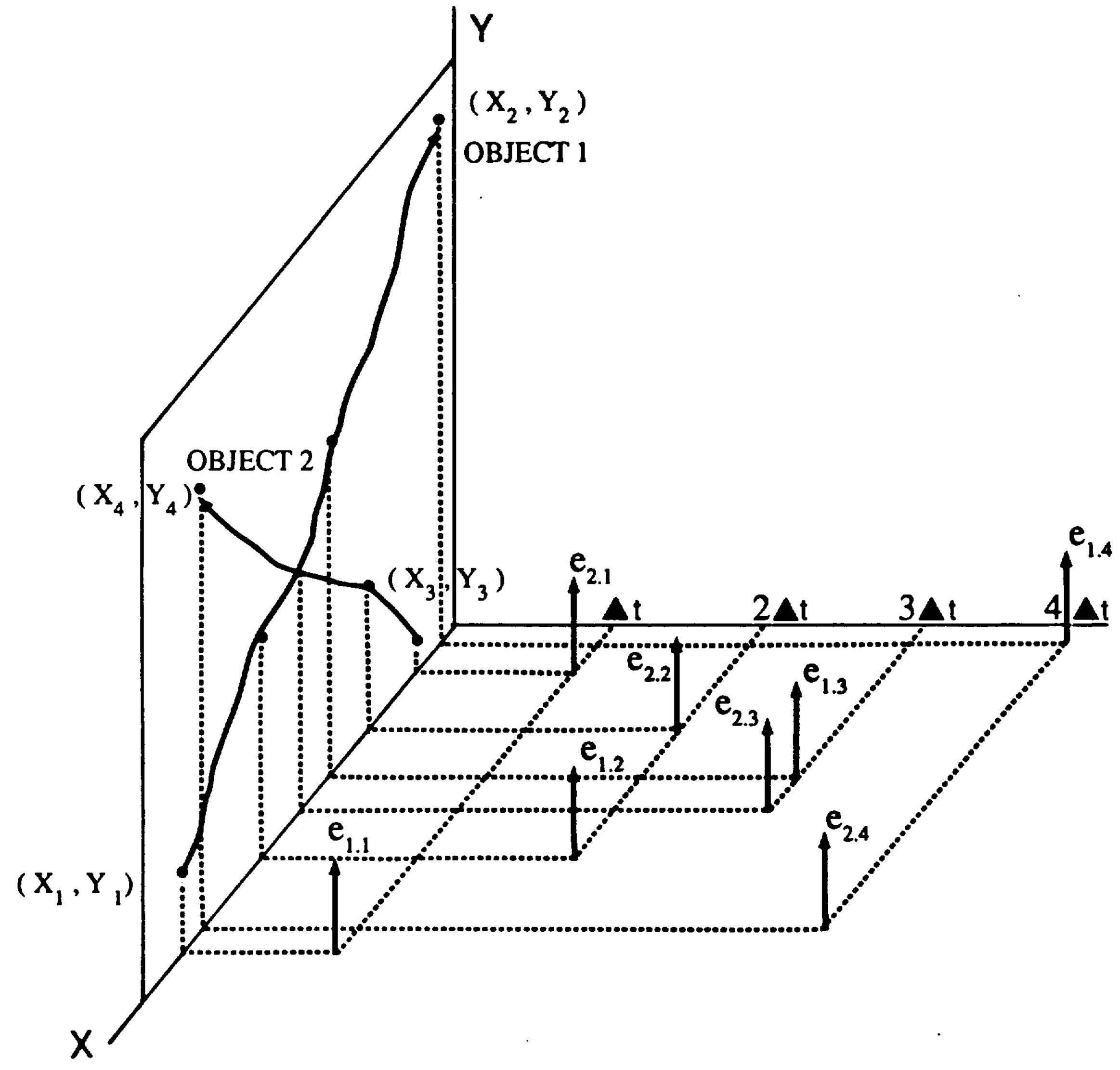

Figure 3. Synchronous simulation of two moving objects.

To include autonomy or autonomous like behavior in simulation models requires elaborate and sometimes clumsy workarounds in most simulation environments. For example, to obtain an apparent autonomous behavior, a scheduler asks each movable object to move each time the simulation clock advances. If interaction events between objects such as collisions, intersections, and detection by sensors are important (e.g. aircraft in the illustrative scenario) then movements and event detection must be made against some common frame of reference. To achieve true autonomy of movement, this typically requires using a fixed-time advance approach for the simulation clock (i.e. synchronous simulation). The synchronous simulation necessitates the evaluation of each entity's relationship to every other entity at every time advance of the simulation clock. The resultant computational and modeling complexity severely restricts the simulation modeling domains where autonomy my be used efficiently and effectively.

While conceptually desirable, implementing autonomy in system models compatible with traditional discrete-event simulation environments has been difficult to achieve. As previously discussed, what is desired is to allow each autonomous object to schedule its next event anywhere in the model space its operational rules permit, determine if there are any conflicts with any other objects in the model in achieving that next event, and implement conflict resolutions strategies among objects when necessary. Clearly, how objects efficiently and effectively recognize and resolve conflicts are the basic issues of hindering implementing autonomous objects in discrete-event simulation models.

DISCUSSION OF THE SIMULATION PROBLEM

Researchers and simulation vendors have addressed aspects of the paradigms and knowledge demands identified previously. The availability and popularity of object-oriented programming languages have provided some movement on the consideration of modeling paradigms other than networks. The arrival of simulation environments such as MODSIM II from the CACI Corporation as well as the wide availability of languages such as C++ and Smalltalk has facilitated the development of object-oriented simulation. However, the full impact of the object-oriented paradigm on modeling, in general, has yet to appear.

To date, most uses of the object-oriented paradigm has been in the implementation of traditional network models in object-oriented languages and simulation environments. The underlying or base model still remains the queuing network. However, the availability of object-oriented simulation languages such as MODSIM II does provide the capability to consider more encompassing modeling paradigms other than queuing networks. Indeed, it is perhaps the evolving impact of the object-oriented paradigm on base and lumped modeling which has potential to provide a larger impact on simulation than any single software modeling approach.

The knowledge-based modeling approach offers a new perspective for simulation by merging traditional modeling with knowledge-based expert system technologies such as knowledge representation and automated inferencing. The use of knowledge-based models permits simulation to go beyond ***"What if...?"*** and explore more open-ended questions, such as, finding optimal solutions, formulating new policy, explaining why a given sequence of events occurred, providing answers to "Can this event ever happen?", or to the question "Which events might lead to this event?" These types of questions can only be answered by integrating descriptive declarative components (i.e. knowledge bases) into model domains as well as providing the inferencing mechanism needed for planning and problem solving.

Not surprisingly, knowledge-based simulation has also been the focus of much research and development in the past decade. In these efforts, knowledge-based simulation tools have been typically developed on top of artificial intelligence languages such as LISP or PROLOG. Examples include ROSS [Rothenberg, *et al*, 1989] and BLOBS [Middleton and Zanconato, 1986]. This approach has been extended for practical reason to using an expert system development environments on dedicated LISP machines [O'Keefe, 1989]. Examples cited by O'Keefe of this approach include:

SimKit, a product of Intellicorp, developed using the Knowledge Engineering Environment (KEE) ;

Knowledge-Based Simulation system (KBS), developed at Carnegie-Mellon, which uses the facilities of Knowledge-Craft; and

Rule-Oriented Simulation System (ROSS) developed at the RAND Corporation and implemented on top of ART.

These approaches to simulation have come mostly from the AI community where a simulation model is used to aid the reasoning process in decision making. The modeling emphasis is on decision making and declarative knowledge representation. Such approaches have developed simulation models as sets of conditional events expressed as rules [O'Keefe, 1989]. Additionally, under these approaches, the simulation management and overhead functions (e.g. event calendar and data collection) are written based on rules or in the underlying AI language (i.e. LISP or PROLOG). Perhaps just as important, the base model constructs often still reflect the queuing network as the underlying base modeling paradigm and rely on asynchronous and synchronous simulation implementation strategies. The so-called lump model of the phenomena of interest is represented as the sets of conditional event expressed as rules. The software model is then implemented based on a LISP, PROLOG, or more specific AI programming paradigm.

The simulation community's interest in knowledge-based simulation has been driven by the need to model increasingly complex systems. This need is especially apparent where the domain of interest is considered complex and/or complex decision making is central to specific phenomena being modeled. The general approached used by the simulation researchers like the AI researchers has been to build the simulation software in a AI environment based in either LISP or Prolog. Typically, the approach has been to implement well understood modeling methods and constructs from traditional simulation modeling in the AI based language and then integrate some facilities for knowledge representation into the model [O'Keefe, 1989]. The emphasis of these approaches has been on the knowledge representation of the decision processes in the model.

Contrasting these two communities approaches provides some useful insights [O'Keefe, 1989]

(1) The AI community has made better use of available AI methods, software, and hardware.

(2) The simulation community has used a sounder methodological base for developing simulations.

(3) The converse is true of each.

(4) Both communities rely on the same underlying paradigm for their base models.

Perhaps what is most illustrating of the approaches of the two communities is the efforts each must expend to obtain the capability generally associated with the other. Both communities have well established tools and methodologies to perform their central goals. In AI, numerous expert system shells and development environments are available and widely used to support knowledge base development and inferencing. Similarly, in simulation, numerous simulation languages and environments are available to support a wide range of simulation models and goals. But, in their efforts to bring both AI and simulation together, each community appears to start at the base level of the other. The AI community tends to develop the simulation models from scratch in whatever development environment with which they are working. The simulation community tends to develop not only the expert systems from scratch including inferencing techniques in some base AI language but, to achieve integration of the AI language with a procedural language, the simulationist must also developed from scratch in the base procedural (or object-oriented) language his implementation of the simulation model. Both fail to exploit the strengths and general tools of the other.

This general situation is impeding the development of simulation to meet the challenges identified. Consider the efforts researcher must currently expend to do knowledge-base simulation. First, they must concentrate resources on building the side of the AI-simulation pair in which they are deficient. Such development efforts can severely delay arriving at a point where the researchers can focus on the problems associated with the methodology and philosophies of knowledge-based systems. Due to the effort and resources required to actually get a working simulation with knowledge-base capabilities or vice-versa, the attainment of a working model, not necessarily a useful model, appears to be perceived as the principle achievement. Indeed, such achievements should be acknowledged. However, because of the expenditure in resources to merely get a working model, further development , if pursued at all, tends to focus on the refinement of the basic approaches and goals of the project which initiated the effort. Fundamental experimentation and investigation of the issues and potential of knowledge-based simulation to address new phenomena (e.g. autonomy) and more complex manifestations of the phenomena currently being studied are generally not pursued.

What is needed for the simulation and AI communities is the marriage of each's tools and development environments which can reduce the development effort necessary to obtain the basic simulation and knowledge-base modeling tools. Further, what is needed is also the inclusion of environments which will enable simulation to investigate broader ranges of modeling paradigms and support the development of the concept of autonomy.

The need for integration of simulation and AI tools and methodologies is especially critical for air traffic modeling and the issues raised by vertical flight and the new navigation technologies. The demands for policy development, operational procedure development, and causal analysis in the increasingly complex air traffic environment envisioned for the twenty-first century cannot be adequately supported by current simulation tools and methodologies. The TATSS project has addressed these simulation problems by effectively integrating the object oriented simulation environment MODSIM II with the expert system shell and language CLIPS. Further, the TATSS project has developed and implemented autonomous objects in this integrated knowledge-based simulation environment. The following sections address these developments in further detail.

GENERAL APPROACH TO SIMULATION PROBLEM

Achieving the goals of the TATSS project required addressing the fundamental simulation problems associated with paradigms, knowledge, and autonomy. As discussed in the previous section, previous efforts involving these concerns did not provide a cohesive or integrated base of software tools or methodologies with which to efficiently support the wide-spread investigation and development efforts associated with TATSS Project simulation model goals. Availability of such software tools and development environments were critical to the TATSS Project efforts. However, time and resources did not permit development of a custom software simulation environment with which to explore the basic modeling issues of the TATSS Project.

The approach used to obtain a suitable software environment to develop the TATSS simulation model employed available off-the-shelf object-oriented simulation and expert system software. By employing off-the-shelf software, each designed for its particular mission, project resources would not be spent on duplicating basic implementation approaches of simulation and expert system. The obvious concern was identifying available simulation and expert system software which could be integrated effectively.

The two software packages were identified which satisfied the availability requirement. The two were MODSIM II from the CACI, Corporation and CLIPS developed and supported by NASA and available from COSMIC, NASA's Computer Software Technology Transfer Center. Both packages generated C language code as an intermediate stage between their base programming environment and executable code. With this common grounding in C, the two packages could interface and communicate via C subroutine calls. Further, modularity and system performance were increased by running both packages as separate processes under the UNIX's V operating system's multi-tasking capabilities. Communications between the two active processes were made using the FIFO pipes available under UNIX.

The establishment of a working simulation environment permitted research and experimentation to proceed on development of autonomy in the simulation. This experimentation led to the development of an approach to autonomy which could be incorporated into the integrated simulation environment of MODSIM II and CLIPS.

MODSIM II

MODSIM II is a general-purpose, modular, block-structured high-level programming language which provides direct support for object-oriented programming and discrete-event simulation. MODSIM II is also a general-purpose, strongly typed, procedural programming language which can be used to write traditional style computer programs. Objects in MODSIM II are defined as dynamically allocated data structures coupled with routines, called methods. The fields in the object's data structure define its state at any instant in time while its methods describe the action which the object can perform. The values of the fields of an object are modified only by its own methods.

In MODSIM II, simulation is supported by a library module which contains a number of simulation specific objects and procedures (e.g. simulation clock, event calendar, probability distributions, random number generators, etc.) All objects are allowed to perform actions which elapse simulation time. A method of one object might include a statement to WAIT until some future time before proceeding to the next statement, or it might send a message to another object so that the message arrives at that object at a specific simulation time. MODSIM II allows a related group of activities to be coded in one routine. When it is necessary to elapse simulation time, the routine suspends execution until the stated amount of simulation time has elapsed then the routine resumes execution

CLIPS.

CLIPS is an expert system tool developed by the Software Technology Branch, NASA/Lyndon B. Johnson Space Center. The acronym CLIPS stands for C Language Integrated Production System. CLIPS is both a type of computer language and a complete environment for developing expert systems. As a complete expert system environment, CLIPS includes features such as an expert systems **shell**, integrated editor, and debugging tools. The expert system shell refers to that portion of CLIPS which performs inferencing. The shell provides the basic elements of an expert system including: (1). fact-list, (2) knowledge-base, and (3) inference engine. A program written in CLIPS consists of rules and facts. The inference engine decides which rules should be executed and when. CLIPS versions 5.0 or later include an object extension to the language called CLIPS Object-Oriented Language.

CLIPS was designed for full integration with other languages, particularly C and Ada. Versions of CLIPS are available which are written in C and Ada. The most common form of CLIPS integration with other languages is for an executable CLIPS program to be called from a procedural language, perform its function, and then return control back to the calling program. Similarly, procedural code can be defined as external functions and called from CLIPS. When the external code completes its task, control returns to CLIPS.

Still another approach for CLIPS integration is available under multi-tasking operating systems such as UNIX. In this approach, the CLIPS based program element and the procedural based program element are run as a separate coexisting processes under the operating system. Communication between processes is through the operating system. For example, under UNIX, there are several forms of interprocess communication including *pipes* and *sockets*. The use of coexisting processes for integration enhances the efficiency of program execution and increases the modularity of the resulting software system.

AUTONOMY

A simple autonomy strategy requires each object to schedule its next model space position (and associated event) according to its own objectives. The object would then poll ever other object in the model to determine if the newly scheduled event would precipitate any conflicts. Conflicts would be resolved based upon the model's conflict resolution procedures. This is similar to the general approach used in synchronous modeling.

Unfortunately, such a simple conceptual strategy suffers from the same basic demand for computational resources as the synchronous modeling. Moreover, computational resource demand grows with at least the square of the number of objects in the model. What is needed is an approach which enables conflict recognition between objects while minimizing the number of objects and number of communication channels and interchanges which must be maintained between objects.

The conflict identification strategy pursued in the TATSS project uses the concept of a spatial template. Under this concept, all objects in the model space are represented as geometric shapes. To simplify, all object model shapes are polygons. Dynamic objects (e.g. airplanes, ships, guided vehicles, weather, etc.) are also represented by polygons but polygon size and shape is based on the object's model space trajectory. The parameters of the associated polygon of the dynamic objects in the spatial template are defined by the originating object event and the next scheduled event for that object as well as unique characteristics of the object.

For example, the trajectories of two moving objects could be represented as two polygons in the X-Y plane of their movement. The origin of Object 1's trajectory is point X_1,Y_1 and the end of its trajectory is point X_2,Y_2. These points correspond to events $\mathbf{e}_{1,1}$ and $\mathbf{e}_{1,2}$. Similarly for Object 2, its origin and end points are X_3,Y_3 and X_4,Y_4, respectively. The corresponding events are $\mathbf{e}_{2,1}$ and $\mathbf{e}_{2,2}$. The polygons on the spatial template for each of these objects could be defined by the origin and end points and by a Δy for each object. Thus, a polygon representative of Object 1 would be defined by the four points $(X_1,Y_1+\Delta y)$, $(X_1,Y_1-\Delta y)$, $(X_2,Y_2+\Delta y)$, and $(X_2,Y_2-\Delta y)$. A polygon representation of Object 2 would be defined by the four points $(X_3,Y_3+\Delta y)$, $(X_3,Y_3-\Delta y)$, $(X_4,Y_4+\Delta$ y), and $(X_4,Y_4-\Delta y)$. The two corresponding polygons and spatial template are illustrated in Figure 4.

This static representation in two dimensions implies a third dimension (time) by the extent of the polygon from the point of the arrival event. This static representation may also be regarded as a most probable model space trajectory for the object. Uncertainty in the proposed model space trajectory may be reflected in the shape and extent of the object polygon (e.g. the Δy's). Figure 4 illustrates this concept. ***Potential*** conflicts are identified when the representational polygons of objects intersect. Once intersection is detected, objects may then resolve conflicts. Clearly, the key methodological issues are associated with how to efficiently identify polygon intersections in the spatial template.

The problem of how to determine if any two geometric objects intersect in a given coordinate system is the principle implementation issue associated with this approach. Efficient and easily implemented methodologies are readily available. The point of issue with polygon intersection detection for the spatial template approach is the determination of what is an efficient way to identify which objects should be tested to determine if they intersect? Testing between all objects in the system every time a new event is scheduled merely recasts the problems of synchronous modeling.

The spatial template is fundamentally concerned with the identification of the ***POTENTIAL*** conflicts. That is, to identify those objects whose model space trajectories may compete for the same model space resources (e.g. space) at the same time. The goal is to reduce the message traffic between model objects necessary to determine if conflicts will occur. The approach taken

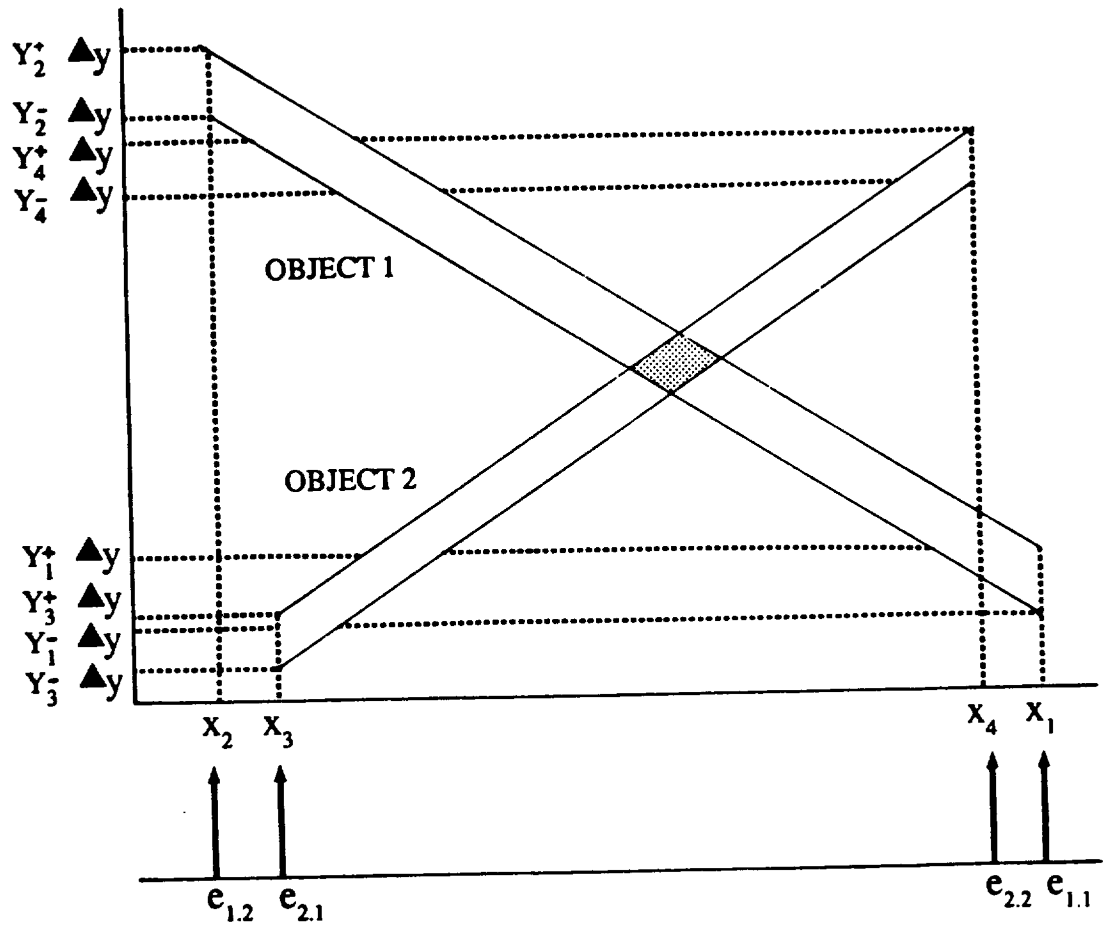

Figure 4. Spatial Template with object trajectory polygons.

by the TATSS Project is to graphically represent the model space trajectory of model system objects and to identify the intersection of trajectories before allowing the object to proceed. How to identify these potential conflicts in an efficient manner is dependent on the how to represent the model space information and the object trajectory information associated with trajectory polygons and scheduled simulation events.

The proposed spatial template approach to representing this information is to partition a two or three dimensional Cartesian model space into equal sectors. The model space trajectory of an object represented by its associated polygon is defined within the model space Cartesian coordinate system (See Figure 5) as discussed before. The Sector partitioning then overlays the model space Cartesian system (See Figure 6). The Sectors are identified by a coordinate sector numbers (e.g. sector 1,3,2). The Sectors through which the trajectory polygon overlays and/or intersects are identified. Associated with each sector then is a list of objects whose trajectory is schedule to go through some part or all of that sector (See Figure 7). When new object trajectories are added a the sector list, a check is made to determine if any other object trajectories are also associated with that Sector. If an object trajectory is already associated with a Sector, then a potential conflict exists. When potential conflicts are identified, the model communicates with each object in the identified Sector to determine if an actual conflict exists. When conflicts do exist, the model may then employ its conflict resolution strategies to remove the conflict.

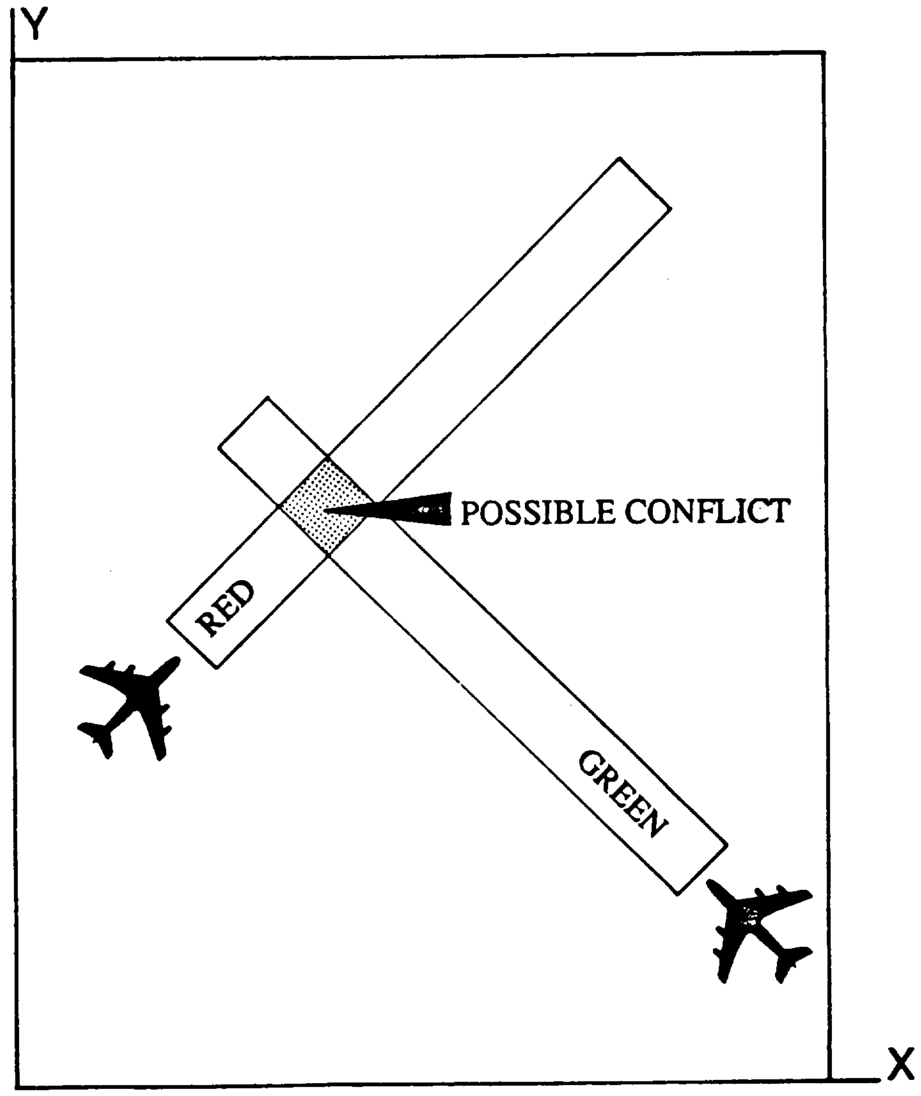

Figure 5. Example spatial template with polygons.

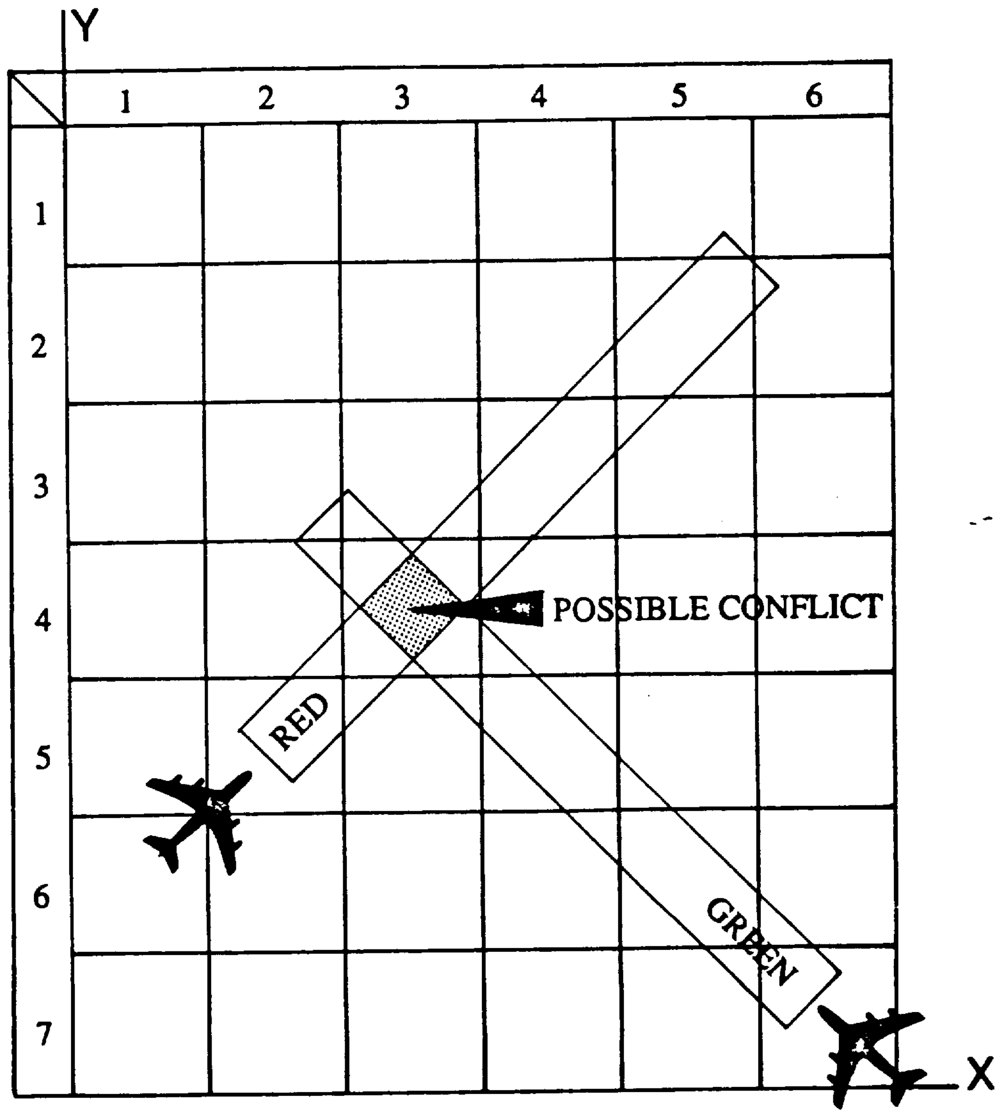

Figure 6. Sectors overlaying spatial template.

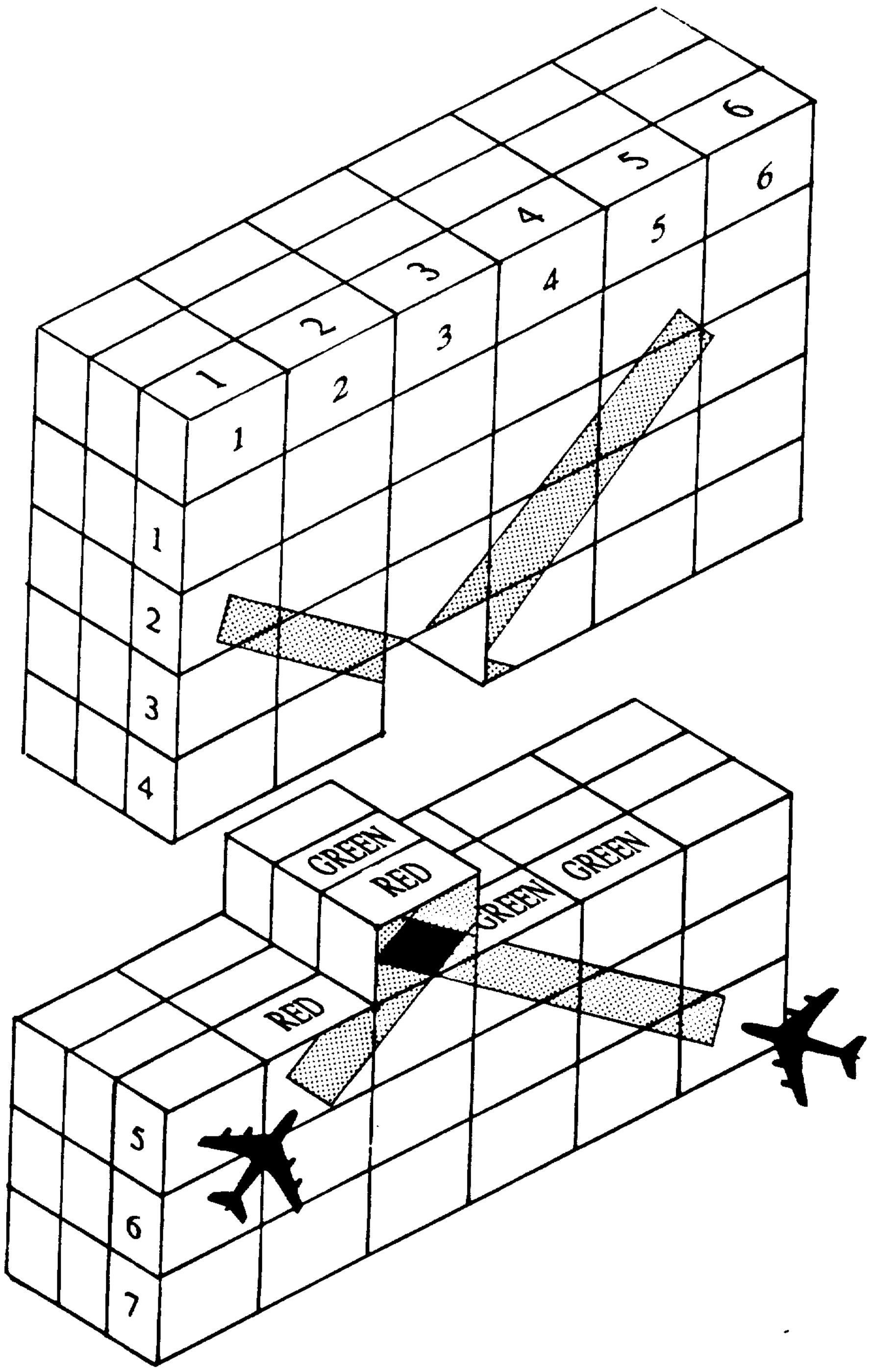

Figure 7. Spatial template with sector queues.

The data structure employed to maintain the list of objects associated with spatial template Sectors is a dynamic array of queues. The Sectors of the model space become elements in the array. Each element in the array is a queue. As object trajectories cross a sector, the name of the object is added to the sector (i.e. queue). Once a new model trajectory has been established for an object, the name of the object is remove from the sectors associated with the old object trajectory.

In the object-oriented software implementation, each sector element is defined as a queue object. The use of objects to describe sectors enables the exploitation of a dynamic array. A dynamic array will only use the memory necessary for the active sector objects. Thus, if no sectors have objects associated with them, they are not created, and do not require computational resources. Likewise, as a sector becomes empty, that sector object may be disposed of, freeing computational resources.

Implementing the spatial template as described above in a discrete-event simulation model provides an approach which supports autonomous object modeling. The spatial template approach reduces the coordination overhead required for autonomous objects in both synchronous and asynchronous simulation. Practical programming considerations dictate an object oriented approach is required for the software implementation.

CONCEPTUAL FRAMEWORK OF TATSS

The basic object model for the TATSS consists of two general types of objects with a total of five basic subtypes. The two general types are (1) Spatial Objects and (2) Model Management Objects. The Spatial Objects have two subtypes (a) Moving Objects and (b) Stationary Objects. The Model Management Objects consist of three subtypes: (a) the Spatial Template Object, (b) the Conflict Identifier Object, and (c) the Conflict Resolver Object. (Refer to Figure 8). In any given model there may be more than one instance of Moving and Stationary objects. However, there is generally only one instance per model of the Spatial Template, Conflict Identifier, and Conflict Resolver objects. Figure 9 provides an illustration of the inter connective relationship between these five types of objects. The general relationships between these objects are illustrated by following descriptive example.

DESCRIPTIVE EXAMPLE. A two dimensional model-space contains phenomena of interest. The phenomena consists of two dimensional entities and their behavior. Entities may have two basic behaviors; (1) they occupy space and (2) they may move from one location to another. Some entities may possess both behaviors (i.e. Moving Objects) while some may posses only the first (i.e. Stationary Objects). Stationary Objects are of random sizes and randomly assigned locations in the model-space. Moving Objects start at random locations within the model-space and move with a random velocity vector for a random period of time. Further, no two entities can occupy the same space at the same time.

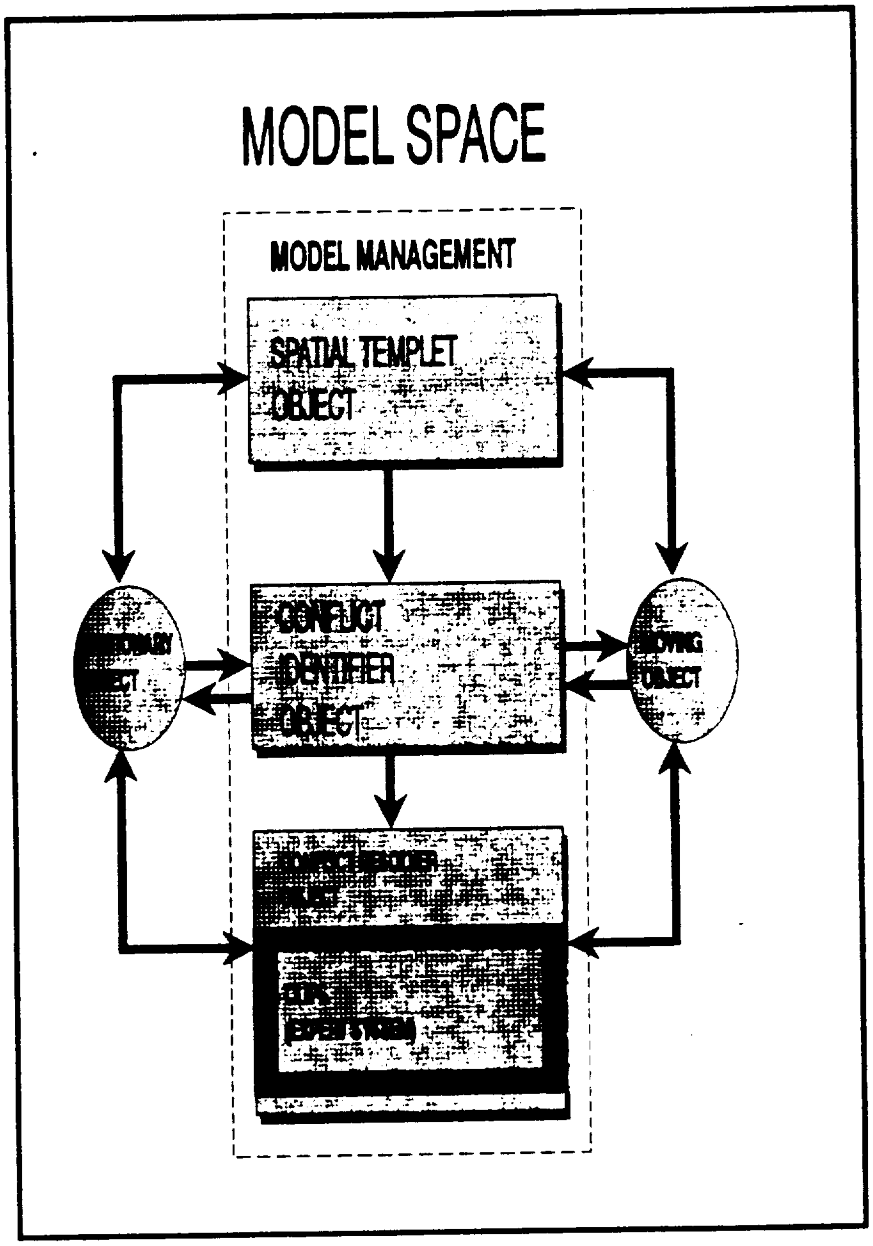

Figure 8. TATSS general object types and sub-types.

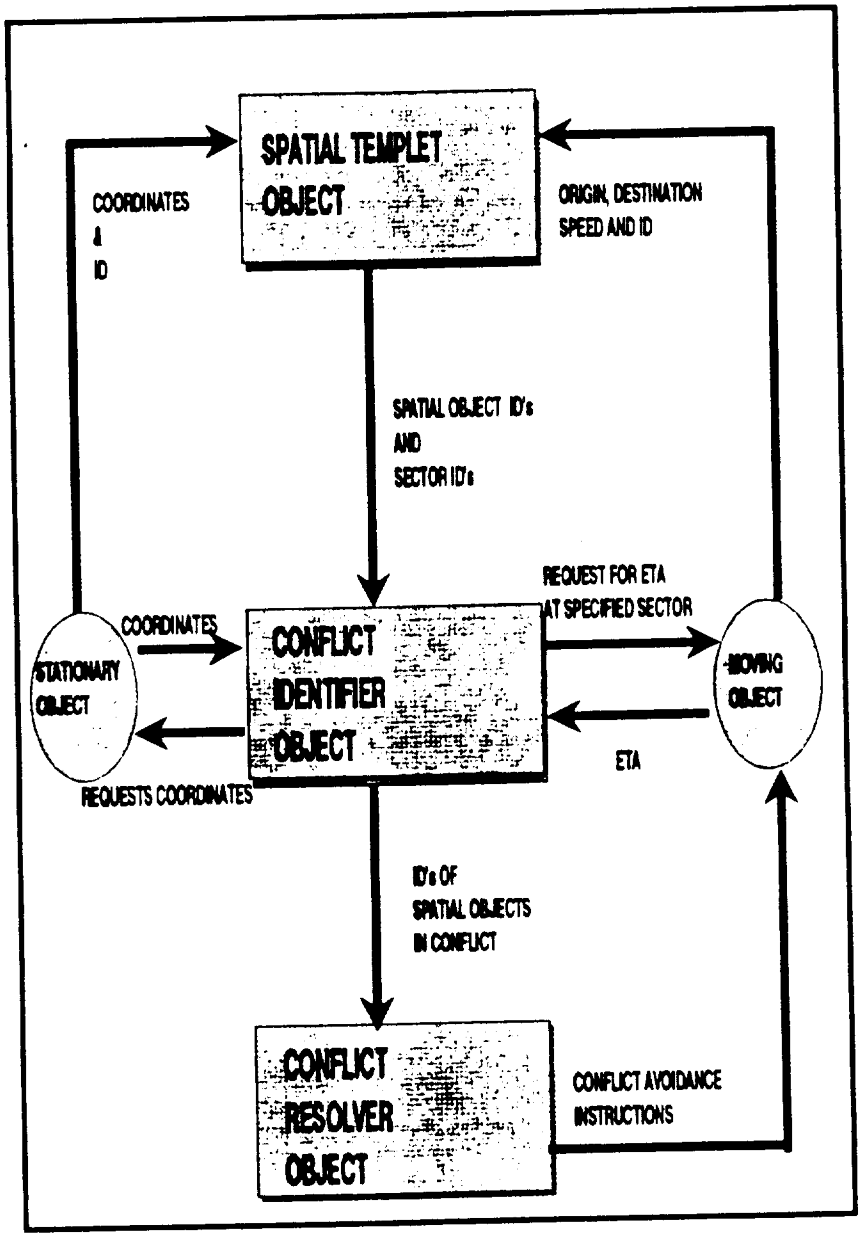
SPATIAL TEMPLET OBJECT
COORDINATES & ID
ORIGIN, DESTINATION SPEED AND ID
SPATIAL OBJECT ID's AND SECTOR ID's
REQUEST FOR ETA AT SPECIFIED SECTOR
COORDINATES
STATIONARY OBJECT
CONFLICT IDENTIFIER OBJECT
MOVING OBJECT
ETA
REQUESTS COORDINATES
ID's OF SPATIAL OBJECTS IN CONFLICT
CONFLICT RESOLVER OBJECT
CONFLICT AVOIDANCE INSTRUCTIONS

Figure 9. Inter conective relationship between object types.

TATSS OBJECT OVERVIEW

In the basic TATSS object model the illustrative example presented would function as follows: All stationary objects report their dimensions and location in the model-space to the Spatial Template Object when they are instantiated. Moving Objects report their dimensions and their current positions in the model-space to the Spatial Template Object when they are instantiated. Before a Moving Object moves, it notifies the Spatial Template Object of its next desired position or location in the model-space. This establishes a planned model-space trajectory for the moving object which is represented graphically in the Spatial Template.

If the planned model-space trajectory of a Moving Object intersects the model-space trajectory of another Moving Object or the model-space of a stationary object then a possibility of a conflict exists. The Spatial Template Object identifies such intersections of the model-space trajectory polygon. These intersections represent potential conflicts. When intersections are identified, the Spatial Template notifies the Conflict Identifier Object that a possible conflict exists between the two object instances in the model-space.

The Conflict Identifier then questions the two objects to determine when each object will arrive at the intersection neighborhood. Obviously for stationary objects, there is no arrival time. The objects are always there. If the arrival are separated by enough time, no actions are required and the moving object which wished to move to its next position is allowed to schedule its desired destination. The current position of the moving object and its desired/goal position establishes a spatial trajectory in the Spatial Template Object. If one of the two objects is a stationary object or if the separation time of two moving objects is insufficient, the Conflict Identifier Object notifies the Conflict Resolver Object that a conflict exists between two objects for model-space resources.

The Conflict Resolver determines the nature and extent of the conflict. The Conflict Resolver also determines the course action to be taken depending on the objects involved. For example, a conflict may be determined to exist between a stationary object and a moving object. The conflict resolver object might simply notify the moving object that it will intersect a stationary object if corrective action is not taken. Or, the conflict resolver may determine the best course of action and issue new course trajectory to the moving objects. The choice would depend on the design of a particular model. Once a new course action is determined either by the resolver or the objects themselves, the new course objective is sent to the Spatial Template and the process begins again.

With this overview of the object architecture in mind, the following sections describe each of the object subtypes in more detail.

MOVING OBJECT. A Moving Object occupies space and follows a model-space trajectory as it moves from one location to another in model-space. A Moving Object is represented in the model as a geometric surface (i.e. a polygon) in either two or three dimensional space. A Moving Object may follow its own goal directed model-space trajectory based upon its own decision making capabilities and rules or it may be directed to follow specified trajectories by other objects (e.g. Conflict Resolver) in the model based on system model state. A Moving Object communicates with the Spatial Template Object, the Conflict Identifier Object, and the Conflict Resolver Object.

STATIONARY OBJECT. A Stationary Object occupies space and does not normally change locations in the model-space. A Stationary Object is represented in the model-space as a geometric surface (i.e. a polygon) in either two or three dimensional space. A Stationary Object communicates with the Spatial Template Object, the Conflict Identifier Object, and the Conflict Resolver Object.

SPATIAL TEMPLATE OBJECT. The Spatial Template Object provides a graphical memory of the model-space status between model events. The Spatial Template Object maintains this memory by a graphical representation based on the proposed model-space trajectory of a moving object or the occupied space of a stationary object. When new graphical descriptions are created representing model-space trajectories and model-space occupation, these new graphical objects are incorporated into the spatial template.

As each new graphical representation is added to the spatial template, the Spatial Template Object simply identifies instances when a new trajectory may compete for the same model-space resources as another object with a previously approved model-space trajectory. That is, it identifies when a potential conflict occurs between two or more model-space objects. When potential conflicts are identified, the Spatial Template passes the identification of the involve objects to the Conflict Identifier Object.

There are two related principal technical issues associated with implementing the Spatial Template Objects of the TATSS base object architecture into an associated software model. The first technical issue is the identification of a data structure to represent the spatial template graphical based information. The second technical issue is how to detect when model-space object in the Spatial Template Object intersect. Each of these issues is addressed in more detail in subsequent sections of the report.

CONFLICT IDENTIFIER OBJECT. The Conflict Identifier Object receives from the Spatial Template Object the identification of spatial objects which have a potential conflict and the location in the model-space of that potential conflict. The Conflict Identifier Object sends messages to each of the potentially conflicting spatial objects asking them at what time each will arrive at the location of the potential conflict. If the difference in the arrival times of each is sufficiently large enough, then no conflict will occur. The spatial object whose new position change precipitated the conflict identification process is given permission to schedule its event past the location identified with the possible conflict. If the difference in the arrival times of each object is sufficiently small enough, then a conflict will occur. The Conflict Identifier Object then sends a message to the Conflict Resolver Object containing the identification of each of the conflicting spatial objects, the location in the model-space of the conflict, and the time of the conflict.

CONFLICT IDENTIFIER OBJECT. The Conflict Identifier Object receives from the Spatial Template Object the identification of spatial objects which have a potential conflict and the location in the model-space of that potential conflict. The Conflict Identifier Object sends messages to each of the potentially conflicting spatial objects asking them at what time each will arrive at the location of the potential conflict. If the difference in the arrival times of each is sufficiently large enough, then no conflict will occur. The spatial object whose new position change precipitated the conflict identification process is given permission to schedule its event past the location identified with the possible conflict. If the difference in the arrival times of each object is sufficiently small enough, then a conflict will occur. The Conflict Identifier Object then sends a message to the Conflict Resolver Object containing the identification of each of the conflicting spatial objects, the location in the model-space of the conflict, and the time of the conflict.

CONFLICT RESOLVER OBJECT. The Conflict Resolver Object determines the actions specified by the system model designed to resolve the conflict. After determining the appropriate course of actions, the Conflict Resolver Object sends message(s) to one or both affected spatial object instructing them on actions to be taken to resolve the conflict. If the system model design specifies that decision making to resolve a conflict be performed by the one or both of the objects in conflict, the Conflict Resolver Object informs the appropriate object or objects of the conflict and the identifies the other object involved. In such a case, each spatial object must be designed to make conflict resolution decisions.

If the system model dictates that determination of the conflict resolution actions be made by the Conflict Resolver Object, the conflict resolution mechanism must be directly or indirectly a part of the Conflict Resolver Object. Once the appropriate actions are determined by the Conflict Resolver Object, these actions are sent as messages to the involved objects. Each spatial object must be so designed to implement these specified actions.

The conflict resolver mechanism is best thought of as rule-based approach to conflict resolution. This mechanism may include hard coded software methods and procedures as part of the system software model or may be based on rule-based inferencing through a separate knowledge base typical of expert-system approaches. To obtain this latter capability, the Conflict Resolver Object must include or interface with an expert system shell or language. In TATSS this expert system shell is CLIPS.

SPATIAL TEMPLATE DATA STRUCTURE

In previous discussions, the Spatial Template was identified as a graphical representation of the model state space. The Spatial Template is both more and less than this. The fundamental interest is the identification of the **POTENTIAL** conflicts. That is, to identify those objects whose model space trajectories may compete for the same model space resources (e.g. space) at the same time. The goal is reduce the message traffic between model objects necessary to determine if conflicts will occur. The approach taken is to graphically represent the model space trajectory of model system objects and to identify the intersection of trajectories before allowing the object to proceed. How to identify these potential conflicts in an efficient manner is dependent on the how to represent the model state space information.

TATSS's approach to these issues is based on a partitioning of the model space. Currently, the model space is divided into equal size sectors. Thus, the model space can be represented by either a two or three dimensional Cartesian coordinate system. Sectors are identified by their coordinate numbers (e.g. sector 1,3, 2). The model space trajectory of an object represented by its associated polygon is position with reference to this coordinate system (Refer to Figure 6). The sectors through which the trajectory polygon goes are identified. Associated with each sector is a list of objects whose trajectory is schedule to go through some part or all of that sector. When new object trajectories are added a the sector list, a check is made to determine if any other object trajectories are also associated with that sectors. If an object trajectory is already associated with a sector, then a potential conflict exists. The Conflict Identifier is then notified and the names of the object with potential conflicts are sent with the notification.

The data structure employed to capture this approach is a dynamic array of queues. The sectors of the model space become elements in the array. Each element in the array is a queue. As object trajectories cross a sector, the name of the object is added to the sector (i.e. queue) (Refer to Figure 7). Once a new model trajectory has been established for an object, the name of the object is remove from the sectors associated with the old model trajectory.

In the object-oriented software implementation, each sector element is defined as a queue object. The use of objects to describe sectors enables the exploitation of dynamic arrays. A dynamic array uses only the memory necessary for the active sector objects. Thus, if no sectors have objects associated with them, they are not created, and do not require computational resources. Likewise, as a sector becomes empty, that sector object may be disposed of, freeing computational resources.

INTERSECTION IDENTIFICATION

The identification of the intersection of the model space trajectory of objects follows a hierarchical decision process. The first level in the process determines if the model space trajectories (i.e. the trajectory polygons) of objects transverse or occupy the same sector or sectors. If more than one model space trajectory intersects a sector, then the name of the objects associated with that sector are sent to the conflict identifier to establish if a conflict actually exists. The issue of concern is to identify what sectors an object model space trajectory intersects. This section discusses the approach used in TATSS to identify these sectors intersected by an object's model space trajectory.

When an object reports its current and goal positions to the Spatial Template, the Spatial Template defines the trajectory polygon in the model space. Determination of which sectors are intersected is made by scanning along the x direction of each side of two parallel sides of the polygon. The two parallel sides chosen are the two which are parallel to the velocity vector. Scanning is made in the direction of the x component of the velocity vector. Scanning along the other sides of the polygon is not necessary because the width of the polygons are less than one sector width.

As the scanning occurs along the polygon side, the contents of each sector queue intersected is checked. If another object's (or objects') trajectory has already been recorded in the queue, the name of the object(s) along with the object which is scanning is sent to the Conflict Identifier for further conflict evaluation.

Scanning is performed by establishing the coordinate border values for the sector under consideration. For example, in a two dimensional coordinate system, one side of a polygon could be defined by the two points (1,2) and (8,11). Movement is in the direction from (1,2) to (8,11). Sectors in this example are 3 unit square. Therefore, sector 1 would be defined by sector borders of X=3 and Y=3. The point (1,2) is less than the value for each border but greater than 0. Consequently, point (1,2) is in the first sector. Scanning continues along the line defined by the points (1,2) and (8,11). The next test point is the X border value of 6 and the Y value where the polygon side intersect the x axis with value 6. The scan determines which sectors the line intersects/crosses between this new point and the old sector value (i.e. sector 1, 0<X<3, 0<Y<3). The sector scan continues until the sector contains the end points of the polygon is identified or an identified conflict results in a new trajectory plan. The area of sector identification is the one area of the project which would benefit from more intensive investigation.

SOFTWARE OBJECTS USED TO IMPLEMENT THE MODEL OBJECTS

In this section, we will present the software objects which compose the prototype application part of the model and their relation to the conceptual model objects specified in the previous section. The conceptual *Moving object* is implemented in the prototype as a PlaneObject. PlaneObjects are capable of moving and stopping in space. Each instances of a PlaneObject has its own speed, direction, current spatial position and intended destination. The conceptual *Stationary Object* is implemented as the StationaryObject in the prototype. StationaryObjects are represented in the Model Space as polygons and have no associated movement.

The *Spatial Template* is implemented as the SpatialTempletObject. The SpatialTempletObject defines an object that represents a model space. This model space can be divided by rectangular coordinates into smaller areas called sectors. Each sector can be thought of as a queue that maintains a list of all objects that are associated with this sector. Maintenance of the sector queue lists is the responsibility of SectorManagerObject. The SectorManagerObject updates the information about the location of PlaneObjects and StationaryObject with respect to the spatial template sectors. Individual sectors are represented in the prototype by SectorObjects. SectorObjects report to the Spatial Template any time the trajectory polygon of an instance of a PlaneObject intersects a sector which already has a StationaryObject or PlaneObject instance associated with it.

The *Conflict Identifier* is represented in the prototype by the ConflictIdentifierObject. The ConflictIdentifierObject establishes if a collision will occur between the planned trajectory of a PlaneObject and with other PlaneObjects and StationaryObjects. The *Conflict Resolver* is represented by the ConflictResolverObject in the prototype. The ConflictResolverObject is responsible for resolving any conflict that might exist between PlaneObjects or between PlaneObjects and StationaryObjects in the Model Space. The implementation of this

ConflictResolverObject includes the capability to interface and use the CLIPS expert system shell. Each of these prototype objects are discussed in more detail in the following sections.

PLANE OBJECT

PlaneObjects are the software implementation constructs of the conceptual MovingObjects described in the approach to the problem. PlaneObjects represent aircraft in the air traffic simulation model. PlaneObjects control their own speed, direction, location, and report information to the spatial template. PlaneObjects also respond to instructions from the ConflictResolverObject. PlaneObjects have the following methods:

CalcTime
ChangeStatus
Delaymove
Direction
InitPlaneFields
NameMover
NewArrive
NewMove
NewTime

Each of these methods are discussed in more detail in the following sections.

CalcTime. The CalcTime Method provides the remaining time to the next scheduled destination point in the simulation model. The ConflictIdentifierObject calls the CalcTime Method to request the remaining time. The CalcTime Method calculates the time at which the plane will arrive at its destination by establishing the remaining distance between the PlaneObject's current position and its destination position then dividing that distance by its speed. The formula of time calculation is:

$$\text{Distance} = [\ (\text{Xposition} - \text{XDestination})^2 + (\text{Yposition} - \text{YDestination})^2\ ]^{1/2}$$

Time = Distance/ Speed.

Where,

Xposition = X coordinate of the plane's current position

Yposition = Y coordinate of the plane's current position

XDestination = X coordinate of the plane's destination

YDestination = Y coordinate of the plane's destination

Speed = Speed of the plane

ChangeStatus. The ChangeStatus Method is used to change the PlaneObject's status. In air traffic simulation model this the ConflictResolverObject sends this message to the PlaneObject. The PlaneObject's status can have one of the following values:

"AT DESTINATION"

"GIVEN PERMISSION"

"NEW POSITION"

"INITIALIZED"

Delaymove. The Delaymove Method is activated when the ConflictResolverObject does not permit a PlaneObject to move to its next destination point. The PlaneObject holds at its current position for a specified amount of time. The PlaneObject status remains the same and it is not changed. At the end of the time delay, the PlaneObject sends its next destination to the Spatial Template again. Delaymove may be called more than once for the same plane. When there are no further conflicts or when the ConflictResolverObject no longer delays the PlaneObject, the PlaneObject is permitted to begin its move to the next destination.

Direction. The Direction Method calculates the PlaneObject's direction in ± degrees referenced to the y-axis (or North of the air traffic simulation model). The PlaneObject direction is calculated by the following formula:

Heading = ARC SINE (YComp)

Direction = Heading * 180.0 / Pi

Where,

YComp = normalized Y component of the plane's path (i.e. (Destination_Y_positon - Current_ Y_ Position)/Distance)

InitPlaneFields. The InitPlaneFields Method assigns the initial values of the PlaneObject's attributes when a PlaneObject is instantiated. Each instance of a PlaneObject will be assigned an ID, starting position coordinates, speed, and destination coordinates. These values are obtained by calling methods of the Planner Object. InitPlaneFields also assigns the status "INITIALIZED" to PlaneObject and activates the Direction Method to establish the direction for the PlaneObject.

NameMover. The NameMover Method assigns a name to the PlaneObject. NOTE: this name is NOT the PlaneObject ID used to address instances of a PlaneObject. The name assigned by NameMover is a string variable. This is used for output reasons only.

NewArrive. The NewArrive Method assigns new destinations to the PlaneObject. New destinations are determined by sending a request to the NextPlan Method of the PlannerObject. The NextPlan Method will provide new values for speed and destination coordinates. NewArrive also request a new direction be established by the Direction Method. The NewArrive Method assigns the PlaneObject a new status "NEWPOSITION". It also sends the CalcTime Method a message to update the estimated time-of-arrival (ETA). This new information is provided to the ReportPosition Method of the Spatial TempletObject.

NewMove. The NewMove Method is activated by NoConflict Method of the ConflictIdentifierObject. The NoConflict Method is sent when the PlaneObject's destination plan does not result in a conflict with any other PlaneObject's destination plans. As a result, the

PlaneObject is given the permission to move to its destination. The NewMove Method advances the simulation clock with a duration equal to the estimated time-of-arrival. After arrival at the destination, the PlaneObject updates its status to "AT DESTINATION." NewMove also sends a message to the ReportPosition Method of the SpatialTempletObject.

NewTime. The NewTime Method calculates the time required by the PlaneObject arrive at a specified destination. The time is calculated by establishing the distance between the PlaneObject's current position and destination position and dividing that distance by the speed of the PlaneObject. The formula for this calculation is:.

Time = (((XCurrent -XnextDest)**2 + (YCurrent - YnextDest)**2)**1\2) / speed

STATIONARY OBJECTS

Each StationaryObject is permanently assigned to the sector(s) in which it lies and its envelope is the actual envelope of the prohibited travel area. In the event of a potential conflict, the conflict resolver must give priority to the stationary object over any PlaneObject. In the prototype software the stationary object's size, dimensions, and locations are randomly assigned. Stationary objects communicate with the SpatialTemplate Object. Stationary objects have the following method:

InitStationaryFields

InitStationaryFields. The InitStationaryFields randomly selects four points within the model space to specify a stationary object. This method also sets the ID name to the ObjectNumber and assigns the name of the stationary object created to name given the object at its creation. This method also communicates with the Spatial Template's method ReportStationaryPosition.

SPATIAL TEMPLATE OBJECT

The SpatialTemplateObject provides a graphical memory of the model-space status between model events. The SpatialTemplateObject maintains this memory by a graphical representation of the model-space, static objects, and the proposed model-space trajectory of PlaneObjects. Necessarily, the model-space must be bounded, constructed, defined, and referenced in terms of a basic coordinate system. The current SpatialTemplateObject implementation uses a two-dimensional Cartesian coordinate reference system based on the ranges of the two dimensions of the model-space. The ranges of model-space are arbitrary and determined by user specification. The SpatialTemplateObject constructs the coordinate reference system with point-of-origin (i.e. point 0,0) in the center of the defined model-space.

For example, a model space defined by the dimensions of 200 km by 150 km would be represented by the SpatialTemplateObject as follows: Assume the 150 km dimension represents an east-west direction and the 200 km dimension represents a north-south direction. The origin of the model space would be at the center of this 150 km X 200 km rectangle (i.e. point 0,0). From the origin, the boundaries of the model-space would be defined to be +75 km east, -75 km west, -100 km south, and +100 km north. All PlaneObjects and StaticObjects spatial attributes composing the air traffic simulation model would be defined and referenced in terms of this coordinate system.

The SpatialTemplateObject maintains the model-space memory between events by representing StaticObjects dimensions and PlaneObject trajectories as polygon constructs in the model-space coordinate reference system. Since the model-space environment is dynamic, the SpatialTemplateObject must send and receive messages from the model-space constituents (i.e. instances of StaticObjects and PlaneObjects) as events change the spatial relationships of the constituents. What actions taken by the SpatialTemplateObject depends upon the object originating the message and that objects status. For example, if the instance of a PlaneObject is newly initialized, the SpatialTemplateObject initiates the envelope that surrounds planned trajectory, determine the sectors the trajectory will pass through, and informs those SectorObject instances. If the plane has arrived at its next destination point, the SpatialTemplateObject informs the SectorObject instances to update reflecting that the PlaneObject has already cleared their sectors. This, in effect, is removing the trajectory polygon associated with a completed move.

The SpatialTempletObject is also responsible for the creating instances of the SectorManager, the SectorObjects, SectorQueueObjects, and the MasterSectorArray. The MasterSectorArray is a two dimensional array with pointers referencing SectorObject instances.

This object has the following methods:

GenSectors
InformPotentialConf
ReportPosition
ReportStationaryPosition
SPcheckupdate

GenSectors. The GenSector Method creates the instances of the SectorManagerObject and the SectorObjects. The GenSector Method also creates each sector and assigns it an identification number (reference number). The SectorObjects are represented as a MasterSectorArray.

InformPotentialConf. The InformPotentialConf Method sends a message to the ConflictIndentiferObject that the new trajectory polygon of a PlaneObject intersects a SectorObject which already has model-space static and/or trajectory polygons associated with it. These potential conflicts were detected by the MonitorSectorMethod of the SectorObject. The InformPotentialConf Method maintains the interface between the SpatialTemplate and all other objects of the air traffic simulation model.

ReportPosition. The ReportPosition Method receives messages from StaticObjects and PlaneObjects. ReportPositon receives the pointer to instances of the StaticObjects and PlaneObjects. The ReportPosition Method sends messages to the SectorManagerObject depending on the status of the PlaneObject. If a PlaneObject's status is "AT DESTINATION", the SectorManagerObject is requested to delete the PlaneObject's trajectory polygon and update the associated SectorObject information. If the PlaneObject status is "NEWPOSITION", the SectorMangerObject is requested to create a new trajectory polygon for the PlaneObject based upon its new destination and to send the appropriate SectorObjects messages informing them of their intersection by the new trajectory polygon. In both cases, the PlaneObjects pointers are sent to the SectorManagerObject.

ReportStationaryPosition. The ReportStationaryPosition Method communicates with SectorMangerObjects through the SectorMangerObject's Method SetStationaryEnvSectors. The ReportStationaryPosition Method only passes the name of the Stationary Object to the SectorMangerObject. The intent is maintain the Spatial Template as the principle object that model space object must communicate with.

SPcheckupdate. SPcheckupdate Method sends message to Checkupdate Method of SectorManagerObject informing the SectorManagerObject that the specified PlaneObject has reached its destination point. The Sector ManagerObject subsequently deletes the trajectory polygon of the PlaneObject.

SECTOR MANAGER OBJECT

This acts as a manager for the sectors. It initiates the plane objects' graphical representation, and determines the sectors that the plane will be passing near or occupying during its travel. The sector manager is in charge of updating the information about the plane object and its location with respect to the sectors. Like the sector object, the sector manager object is a third level derivative of the second level spatial template element.

One of the primary functions of the sector manager object is to calculate the vertices of the plane's envelope, and the sectors that the plane and its envelope are crossing. The first is determined by using the information reported about the current position of the plane and its final destination. The latter depends on the results obtained from the first method as basis for its calculations.

Knowing that the width of the envelope is 4 kilometers (2 kilometers on each side of the plane's path), the following calculations are applied to determine the envelopes vertices:

Sector Manager has the following methods:

Checkupdate
InitSectorQueue
SetStationaryEnvSectors
UpdateEnvSectors
UpdateSectors

Checkupdate. The Checkupdate Methods receives messages from the SpatialTemplateObject's SPCheckupdate Method. The message is received when a PlaneObject has reached its destination. Checkupdate's primary task is to update the SectorArray and the MasterSectorArray based on the arrival of the PlaneObject at its destination.

InitSectorQueue. The InitSectorQueue Method initiates the status of all elements of the SectorArray to "Absent" and creates the SectorQueue. InitSectorQueue receives messages from the SpatialTemplateObject's method GenSectors.

SetStationaryEnvSectors. The SetStationaryEnvSectors Method asks the specified stationary object what are its defining coordinates in the model space. This method then identifies the sector(s) occupied by the stationary objects and updates the master sector array.

UpdateEnvelope. The UpdateEnvelop Method determines the location of the vertices of the PlaneObject trajectory polygon based on the PlaneObject's current position and its destination positions. The locations are referenced to the specified model-space coordinate system. The width of the trajectory polygons of the air traffic simulation model is specified in the software code to be 4 Kilometers.

The following calculations identify the vertices for a trajectory polygon:

vertex[1].x =	current position of the plane in x direction - 2.0 * normalized y component of its trajectory.
vertex[1].y =	current position of the plane in y direction + 2.0 * normalized x component of its trajectory.
vertex[2].x =	current position of the plane in x direction + 2.0 * normalized y component of its trajectory.
vertex[2].y =	current position of the plane in y direction - 2.0 * normalized x component of its trajectory.
vertex[3].x =	destination position of the plane in x direction + 2.0 * normalized y component of its trajectory.
vertex[3].x =	destination position of the plane in x direction - 2.0 * normalized x component of its trajectory.
vertex[4].x =	destination position of the plane in x direction - 2.0 * normalized y component of its trajectory.
vertex[4].x =	destination position of the plane in x direction + 2.0 * normalized x component of its trajectory.

UpdateEnvSectors. The UndateEnvSectors Method determines the sectors the PlaneObject trajectory polygon intersects or encompasses. UndateEnvSectors updates the

SectorArray and the MasterSectorArray. TheUpdateEnvSectors receives messages from the SpatialTemplateObject's ReportPosition Method.

UpdateEnvSectors determines the sectors intersected by each side of the trajectory polygon. Intersection is determined by establishing the equation of the line of each side of trajectory polygon from one vertices to the next. Search begins with the current position vertex position with the minimum x value. The search follows the line to the associated destination vertex identifying the sectors the line intersects.

The sector edges are defined as horizontal and vertical lines. In the case of a non-horizontal and non-vertical trajectory polygons, the equation of a line defined by each edge of the polygon (i.e. $y = mx + b$) will intersect each line representing the edge of a sector if the reference coordinate system was of infinite dimensions. (Separate checks are made to handle vertical and horizontal envelopes). If the intersection point between the sector and the trajectory polygon line occurs between the vertices of the trajectory polygon then the trajectory polygon intersects the sector, that sector is then associated with the trajectory of the PlaneObject. The description which follows is only for the x direction. The formulation for the y direction is similar.

The reference coordinate system is scanned along the x axis for right to left (i.e. west-to-east in the air traffic simulation model). Scanning is done in segments corresponding to the borders of the sectors defined by the x coordinates of their vertices. The intersection of a sector is established if the border of a sector defined by its largest x component is between the two vertices of the trajectory polygon used to define the trajectory polygon side line. Next, the intersection point of the trajectory polygon edge and the vertical line defined by x coordinate of the first sector determined in the previous search. is determined. If this intersection's y component value is either less than the maximum y component of the vertex of the trajectory polygon edge defining the line or greater than the minimum y component of the vertex of the trajectory polygon edge defining the line, then the sector intersects trajectory polygon. After the first sector if identified, it is simply a matter of counting how many sectors fit from the left and bottom edges of the reference coordinate system to the point of intersection. The sectors immediately to the left and right of the intersection point will be identified as intersecting the trajectory polygon. The process is repeated for each side of the trajectory polygon identifying all sectors associated with the trajectory polygon.

UpdateSectors. UpdateSectors Method updates the SectorObjects concerning which after PlaneObjects have reached a destination. PlaneObjects are removed which are no longer associated with a SectorObject PlaneObjects scheduled for new destination have result in new SectorObjects being associated with that PlaneObject.

SECTOROBJECTS

The SectorObject corresponds to a subdivision of the model-space reference coordinate system of the SpatialTemplateObject into reference grids. Each SectorObject defines a square area of the specified model-space in the reference coordinate system. Each SectorObject is responsible for maintaining a record of all SpatialTemplateObject polygons which intersect or encompass it. The SectorObject identifies any new polygon additions to its sector as a possible conflict if any other polygons are currently on record for that sector. When such situations identified, they are reported as possible conflicts to SpatialTemplateObject. The SpatialTemplateObject in turn notifies the ConflictIdentifierObject of the potential conflict and the Static and PlaneObjects involved. The SectorObject type is defined with multiple inheritance. The SectorObject's parents are the SectorManagerObject and the MODSIM-defined Queue Object. Due to inheritance all methods of SectorMangerObject and the Queue Objects are available to SectorObjects.

Sector Objects have the following methods:

ChangeSectorStatus

MonitorSector

SetBorders

SetObjectID

ChangeSectorStatus. The ChangeSectorStatus Method changes the status of the SectorObject. Status can be either "PRESENT" or "ABSENT". In PlaneObject is assigned to a sector, its status is "PRESENT". If a PlaneObject is not assigned to a sector its status is "ABSENT". The Checkupdate Methods assigns the status to a SectorObject for a given PlaneObject. Based upon this status, a PlaneObject is either added to sector queue or removed from the sector queue. When PlaneObjects are added to sector queue, the MontitorSector Method is sent a message identifying the PlaneObject added and the SectorObject ID.

MonitorSector. The MonitorSector Method determines if two SpatialTemplateObjects polygons are associated with a specified SectorObject. If two polygons are associated with the specified SectorObject, the SpatialTemplateObject's method InformPotentialConf is sent a message identifying the two objects associated with the SpatialTemplateObject polygons and the SectorObject ID.

SetBorders. The SetBorders Method sets the boundaries for each sector by determining the values of its vertexes. Sector partitioning of the reference coordinate system is based on user input of the number of sectors into which the model-space area is to be divided Boundaries for each sector are defined by the coordinates of its vertices. For example, assume the user specified the following inputs:

number of sectors in x direction, numsecx = 20
number of sectors in y direction, numsecy = 20
range of the area in the x direction, xrange = 100
range of the area in the y direction, yrange = 100

The vertices of the lower right hand sector would be calculated based on an individual sector's identification numbers of, for example, (2,3) as follows;

sectorIDx = 2
sectorIDy = 3

xmin= xrange * ((sectorIDx - 1)/ numsecx - 0.5) = 100 * ((2 - 1)/ 20 - 0.5) = - 45.0
xmax= xrange * (sectorIDx/numsecx - 0.5) = 100 * (2/ 20 - 0.5) = - 40.0

ymin= yrange * ((sectorIDy - 1)/ numsecy - 0.5) = 100 * ((3 - 1)/ 20 - 0.5) = - 40.0
ymax= yrange * ((sectorIDy - 1)/ numsecy - 0.5) = 100 * (3/20 - 0.5) = - 35.0

Where,

sectorIDx = sector identification number in the x direction,

sectorIDy = sector identification number in the y direction,

xmin, xmax, ymin, ymax = the four vertices of the sector in reference coordinate system

xrange = the user input range in the x axis for the model space dimension

yrange= the user input range in the y axis for the model-space dimension

numsecx = the number of sectors in the x direction of the model-space

numsecy = the number of sectors in the y direction of the model-space

Therefore, for SectorObject ID of (2,3), the value of its four vertices are (-45,-40,-40,-35). Similar calculations are carried for all other sectors.

The SetBorders Method receives messages from the GenSector Method of the SpatialTemplateObject.

SetObjectID. The SetObjectID sets the sector identification number pair. The sector identification number pair references a specific SectorObject in the MasterSectorArray. The SetObjectID method receives messages from the GenSector Method of the SpatialTemplateObject.

SECTOR QUEUE

The SectorQueueObject is a child of the MODSIM II standard Queue Object. The SectorQueueObject incorporates an added method that requires conformation that the trajectory polygon of the specified PlaneObject is not already assigned to the sector in question. This method is used by the SectorManagerObject to update the list of sectors that are occupied by a PlaneObject's trajectory polygon. The added queue object method is the LogicalAdd .

LogicalAdd. The Logical Method determines the logical status of an object before adding it to the queue. That is, is there already an instance of an object in the queue. If there is, it is not added again. If there is not, then the object is added to the queue.

CONFLICT IDENTIFIER OBJECT

The ConflictIdentifierObject determines if the potential conflicts identified by the SpatialTemplateObject are actual conflicts. The ConflictIdentifierObject receives messages from the SpatialTemplateObject's InformPotentialConflict Method identifying a possible conflict exists between two objects in model-space. The ConflictIdentifierObject requests information from the model-space objects with potential conflicts identified by the SpatialTemplateObject. If no conflict exists, the identified PlaneObject are given permission to proceed as planned. If a conflict does exist, the situation and the objects are referred to ConflictResolverObject for resolution. The ConflictIdentifierObject defines a conflict as two or more model-space objects occupying the same space within five minutes from each other.

The ConflictIdentifierObject establishes conflicts by determining the points that each PlaneObject in the potential conflict the sector in question. Once these points are estimated, the conflict identifier request from the model-space objects concerned to report their estimated time of entering the sector. SpatialObjects are always in the concerned sector. The

ConflictIdentifierObject uses the reported information to determine if the model-space objects will arrive within five minutes from each other.

The following methods are used in this object:

Noconflict

ReportPC

Noconflict. The Noconflict Method is invoked by the SpatialTempletObject when it reports that there are no conflicts. PlaneObject are allowed to move as planned.

ReportPC. The ReportPC Method receives messages from the SpatialTemplate's InformPotentialConf Method identifying the sector, the model-space objects. ReportPC determines the points and times the PlaneObjects enter the sector in question. Conflicts are established if the times in the sector are within five minutes from each other.

CONFLICT RESOLVER OBJECT

The conflict resolver receives the information from the conflict identifier regarding the planes in conflict. Checking for the appropriate action to take is done by the conflict resolver. The conflict resolver would inform the proper plane what it should do, such as delaying its move or giving it the permission to carry on with its projected path. These decisions are made by an expert system located in the CLIPS portion of TATSS. The expert system which is interfaced with the MODSIM part of the TATSS through the UNIX systems FIFO utlities. The method used to resolve conflicts is:

Resolve

Resolve. The Resolve Method of the Conflict Resolver Object receives message containing the two planes in conflict and their respective distances from the intersection point of thier trajectories. The Resolver Method calls the appropriate C language subroutine which is the link between the MODSIM model and the UNIX operating system. The Resolve Method then sends the appropriate data concerning the state of the two objects in conflict to the expert system for conflict resolution. The expert system sends the appropriate command back to the Resolve Method through the C language subroutine. The Resolve Method then sends the appropriate object command to the object which it has been decided must alter it trajectory.

REFERENCES

P. Bratley, B.L.Fox, and L.E. Scharage. *A Guide to Simulation*, (2nd ed). Springer-Verlag, New York:, 1987.

S. Middleton and R. Zanconato, "BLOBS: an Object-Oriented Language for Simulation and Reasoning," in E.J.H. Kerckhoffs, G.C. Vansteenkiste and B.P. Zeigler (Eds.) *Artifical Intelligence Applied to Simulation*, The Society for Computer Simulation, San Diego, CA, 1986, p-p 130-135.

Robert M. O'Keefe. "The Role of Artifical Intelligence in Discrete-Event Simulation." in *Artifical Intelligence, Simulation and Modeling*. (Eds.) Lawrence E. Widman, Kenneth A. Loparo, and Norman R. Nielsen. John Wiley and Sons, New York, 1989.

Jeff Rothenberg, Sanjai Narain, Randall Steeb, Charlene Hefley, Norman Z. Shapiro. *Knowledge-Based Simulation: An Interim Report*. The RAND Corporation, Santa Monica, CA . July 1989.

Bennard P. Zeigler. *Theory of Modeling and Simulation*. Wiley and Sons New York, 1976.

APPENDIX A -1

```
MAIN MODULE CCon;

FROM SimMod IMPORT StartSimulation, SimTime;
FROM RandMod IMPORT RandomObj, FetchSeed, Random;
FROM CCGlobal IMPORT meanarrivaltime, numofhrs;
FROM CCGlobal IMPORT xrange,yrange,numofsecx,numofsecy;
FROM CCGlobal IMPORT numofplanes, planenames,GeneratorObj, ranNumGen;
FROM CCGlobal IMPORT SpatialTempletGenerator ,ConflictIdentifierGenerator;
FROM CCGlobal IMPORT ConflictResolverGenrator,StationaryGenerator;
FROM CCSpatialTempObj IMPORT MasterSectorArray;
FROM CCGlobal IMPORT Pi, ObjectNumber;
FROM CCGeoObj IMPORT GeoObj;
FROM GrpMod IMPORT QueueObj;
FROM UtilMod IMPORT Delay;
FROM CCSectorObj IMPORT SectorObj;
FROM CSectQObj IMPORT SectorQueueObj;
FROM ACMove IMPORT ModelInit,PlannerObj,PlaneObj;
FROM CCStationaryObj IMPORT StationaryObj,Stationary;
FROM CCGlobal IMPORT window,MasterGraphic;
FROM CCGlobal IMPORT clockwindow,clockgraphic;
FROM CCGlobal IMPORT GraphicLib,MasterPlaneIcon;
FROM GTypes IMPORT WorldXhi,WorldYhi,WorldXlo,WorldYlo;
FROM GTypes IMPORT ColorType(Blue,Green);
FROM Graphic IMPORT GraphicLibObj;

  VAR
       xs,ys:  REAL;

  BEGIN
   Pi := 3.14159265;
   ObjectNumber := 0;


   OUTPUT(" PLEASE ENTER : mean arrival time in minutes");
     INPUT(meanarrivaltime);
   OUTPUT(" PLEASE ENTER : number of simulation hours  ");
     INPUT(numofhrs);
   OUTPUT(" PLEASE ENTER :X-Range in kilometers ");
     INPUT (xrange);
   OUTPUT(" PLEASE ENTER :Y-Range in kilometers ");
     INPUT(yrange);
   OUTPUT (" PLEASE ENTER : number of sectors in the x direction ");
     INPUT(numofsecx);
   OUTPUT (" PLEASE ENTER : number of sectors in the y direction ");
     INPUT(numofsecy);
```

```
NEW(ranNumGen);
NEW(MasterSectorArray, 1..numofsecx , 1..numofsecy);
NEW(SpatialTempletGenerator);
NEW(ConflictIdentifierGenerator);
NEW(ConflictResolverGenrator);
NEW(StationaryGenerator);


(*** Initialize graphics ***)
NEW(window);
NEW(MasterGraphic);
NEW(GraphicLib);
 ASK window TO SetColor(Blue);
 ASK window TO AddGraphic(MasterGraphic);
 ASK MasterGraphic TO SetWorld (
     -xrange/2.0 , -yrange/2.0 , xrange/2.0 , yrange/2.0 );
 ASK MasterGraphic TO SetTranslation ( (WorldXhi-WorldXlo)/2.0 ,
             (WorldYhi-WorldYlo)/2.0 );
 ASK window TO Draw();
NEW(clockwindow);
 ASK clockwindow TO SetSize(20.0,10.0);
 ASK clockwindow TO SetTranslation(75.0,90.0);
 ASK clockwindow TO SetColor(Blue);
 ASK clockwindow TO Draw();

NEW(clockgraphic);
 ASK clockwindow TO AddGraphic(clockgraphic);
             ASK clockgraphic Scaling(xs,ys);
             OUTPUT("xs,ys = ",xs," , ",ys);
 ASK clockgraphic TO Scale(1.0,0.65);
 ASK clockgraphic TO SetTimeScale( 1.0/60.0 );
 ASK clockgraphic TO SetTranslation(0.0,30.0);
 ASK clockgraphic TO Draw;
 ASK clockgraphic TO Update;
 ASK clockgraphic TO StartMotion;
 ASK GraphicLib TO ReadFromFile("AConIcon.lib");


NEW(MasterPlaneIcon);
 ASK MasterPlaneIcon TO LoadFromLibrary(GraphicLib,"PlaneIcon");
 ASK window TO AddGraphic(MasterPlaneIcon);
 ASK window TO Draw();
 ASK MasterPlaneIcon TO Erase();
```

```
    {GenSpatialTemplet will ask Sector Manager to gensectors method}

    ASK SpatialTempletGenerator TO GenSpatialTemplet;
    ASK ConflictIdentifierGenerator TO GenConflictIdent;
    ASK ConflictResolverGenrator TO GenConflictResolve;
    ASK StationaryGenerator TO GenStationary;

     {make sure that the Sector Manager is generated before planes are
      created}
        ModelInit; {Create the plane}

        StartSimulation;

   OUTPUT;DEFINITION MODULE ACMove;

FROM SimMod IMPORT SimTime;
FROM RandMod IMPORT RandomObj;
FROM MathMod IMPORT SQRT, POWER,ATAN,pi,SIN,COS;
FROM CCGeoObj IMPORT GeoObj;
FROM GrpMod  IMPORT QueueObj;
FROM Fill IMPORT PolygonObj;
FROM Image IMPORT ImageObj;
FROM GTypes IMPORT FillStyleType(HollowFill,SolidFill,NarrowDiagonalFill,
        MediumDiagonalFill,WideDiagonalFill,NarrowCrosshatchFill,
        MediumCrosshatchFill,WideCrosshatchFill);
FROM GTypes IMPORT PointArrayType;

FROM AuxMathMod IMPORT DIST2D, INT;

TYPE
 PlaneObj =
        OBJECT(GeoObj)

         MyName    : STRING;
         Name : STRING;

      {Status  :  STRING;} {possibilities--
                    "IN FLIGHT",
                    "AT DESTINATION",
                    "ON GROUND",
                    "FLIGHT INTERRUPTED",
                    "INITIALIZED",
                            "NEWPOSITION",
                            "GIVEN PERMISSION"
```

```
                }
 {inherted xpos,ypos from GeoObj}{plane's current position}
 XStart, YStart,  {plane's origin}
 Speed ,
   Velocity,
 Degree,
 Heading,

{inherted xdest,ydest from GeoObj} {plane's final destination}

 Dist,
   FinDesttime,
   ETA,
 TimeStartFlight,
 TranTime,   {time to fly from last position to destination}
 est         {time to fly from current position to a specified position}
          : REAL;

   EnvelopePolygon: PolygonObj;

 {New code by Rajesh to test printing of graphics}

   {PlaneVertex: PointArrayType ;
   PlaneEnvelopePolygon: PolygonObj;
   PlanePolygon : PolygonObj;
 PlanePlaneIcon : ImageObj;
 PlanePlaneGraphic:ImageObj;}



ASK METHOD InitPlaneFields(IN Index : INTEGER;IN StringName : STRING;
                              IN localnameID : STRING);
ASK METHOD  GiveName():STRING;
ASK METHOD CalcTime(): REAL;
ASK METHOD Direction;
ASK METHOD Comp;
ASK METHOD NewTime (IN XnextDest,YnextDest:REAL):REAL;
TELL METHOD NewMove(IN Plane:PlaneObj);
ASK METHOD NameMover(IN Namin:STRING);
    TELL METHOD Delaymove(IN Plane:PlaneObj);
    ASK METHOD NewArrive;
    ASK METHOD ChangeStatus(IN Plane:PlaneObj; IN modstatus:STRING);

    OVERRIDE
            TELL METHOD NewArrive1(IN Indicator:STRING);
```

```
END OBJECT;

PlannerObj =
 OBJECT
   ASK METHOD NextPlan(OUT OUTSpeed, OUTXDest, OUTYDest: REAL);
   ASK METHOD FirstPosit(OUT X,Y : REAL);
   ASK METHOD ObjInit;
   ASK METHOD IdentifyPlaneInstance
            (IN IPIplaneID : INTEGER) : PlaneObj;
 END OBJECT;

PROCEDURE ReportEstimatedTime(IN RETplaneID : INTEGER; IN Sx,Sy : REAL;
                    OUT est : REAL);

PROCEDURE ModelInit;



VAR
     Stream    : RandomObj;
    MovPlanner: PlannerObj;
    Red,Blue,Green,Black : PlaneObj;
    PlaneQue : QueueObj;
    ranNumGen: RandomObj;


END MODULE.







   OUTPUT (" Simulation is done");

 END MODULE.
```

```
IMPLEMENTATION MODULE ACMove;

FROM SimMod IMPORT SimTime,PendingListDump,ActivityListDump;
FROM RandMod IMPORT RandomObj;
FROM MathMod IMPORT SQRT, POWER,ATAN,pi,SIN,COS,ASIN;
FROM GrpMod  IMPORT QueueObj;
FROM CCSpatialTempObj IMPORT SpatialTempletObj,SpatialTemplet;
FROM CConflictIdentObj IMPORT ConflictIdentifierObj,ConflictIdentifier;
FROM AuxMathMod IMPORT DIST2D, INT;
FROM CCGlobal IMPORT xrange,yrange, meanarrivaltime,Pi,  numofhrs,ObjectNumber;
FROM CCGeoObj IMPORT GeoObj;

FROM Fill IMPORT PolygonObj;
FROM Image IMPORT ImageObj;
FROM GTypes IMPORT ColorType(Yellow,Orange,Violet,White);
FROM GTypes IMPORT FillStyleType(HollowFill,SolidFill,NarrowDiagonalFill,
        MediumDiagonalFill,WideDiagonalFill,NarrowCrosshatchFill,
        MediumCrosshatchFill,WideCrosshatchFill);
FROM CCGlobal IMPORT GraphicLib;
FROM CCGlobal IMPORT MasterGraphic;
FROM CCGlobal IMPORT PlaneIcon, PlaneGraphic; {---------------------------------------
-----------------------------------}
OBJECT PlaneObj;

ASK METHOD InitPlaneFields(IN localObjectNumber :INTEGER;IN
StringName:STRING;
                        IN localnameID :STRING);
 VAR

   Xlocation, Ylocation,
   OutSpeed,OutXDest,OutYDest,
   scaleX, scaleY : REAL;

 BEGIN

        ASK SELF TO SetNameID(localnameID);
        ASK SELF TO SetID(localObjectNumber);
        FinDesttime:= ((numofhrs + 0.5)*60.0) ;
        ASK MovPlanner TO FirstPosit(Xlocation, Ylocation);
      XStart := Xlocation;
      YStart := Ylocation;
      XPos := Xlocation;
      YPos := Ylocation;
```

```
  ASK MovPlanner TO
      NextPlan(OutSpeed,OutXDest,OutYDest);
         Speed:= OutSpeed;
         Velocity:= Speed;
      XDest:=  OutXDest;
      YDest:=  OutYDest;

  { tranTime is the total flight time;
    the first time it is calculated it is the travel
       time from original position to destination;
    this time will change if the flight is interrupted;
    the est field will contain the original time;
  }

  ASK SELF TO Direction;

  Status := "INITIALIZED";
      ETA := ASK SELF TO CalcTime();
  {######rajesh move this to model init}
   {ASK SpatialTemplet TO ReportPosition(SELF);}  {move this to model init}

  {FUTURE:  ask SectorManager or SpatialTemp to init these fields}

       NEW ( PlaneVertex, 1..4);
       NEW ( PlanePlaneGraphic );
       NEW ( PlaneEnvelopePolygon );
       ASK PlaneEnvelopePolygon TO SetStyle(HollowFill);
       NEW ( PlanePlaneIcon ) ;
       ASK GraphicLib TO ReadFromFile("AConIcon.lib");
       ASK PlanePlaneIcon TO LoadFromLibrary(GraphicLib,"PlaneIcon");
       ASK PlanePlaneGraphic TO AddGraphic(PlanePlaneIcon);
       ASK PlanePlaneGraphic TO AddGraphic ( PlaneEnvelopePolygon ) ;
       ASK MasterGraphic TO AddGraphic ( PlanePlaneGraphic);
       ASK PlanePlaneIcon Scaling(scaleX, scaleY);
       ASK PlanePlaneIcon Scale(scaleX/300.0, scaleY/300.0);


END METHOD;

TELL METHOD NewMove(IN Plane: PlaneObj);
  VAR

   Distravel,
   Theta  : REAL;
  waittime: REAL;
  A:INTEGER;
BEGIN

   TimeStartFlight := SimTime();
   waittime:=ABS (ETA )+ SimTime();

   WAIT DURATION ETA
            {Plane has arrived at the destination}
      OUTPUT(" Plane waited ETA minutes", ETA);
   END WAIT;

    XPos := XDest;
    YPos := YDest;
    Status := "AT DESTINATION";

     {when at destination reporting position will
      remove: plane from Master sector Array and
       3-dim sector array}

       ASK SpatialTemplet TO ReportPosition(SELF);

     IF (ASK SELF Status ="AT DESTINATION")
         ASK SELF NewArrive;
       END IF;

 END METHOD;



 (**** ADDED on 2/5/1992 ****)

 ASK METHOD ChangeStatus(IN Plane:PlaneObj; IN modstatus:STRING);

  BEGIN
    OUTPUT("Plane  ", ASK Plane ID, " had a status ",
       ASK Plane Status);

    Status:= modstatus;

    OUTPUT("Plane  ", ASK Plane ID, " now have a status ",
       ASK Plane Status);

 END METHOD;
```

```
TELL METHOD Delaymove(IN Plane:PlaneObj);

 VAR
       DTime: REAL;
  BEGIN

  DTime:= 5.0;

  WAIT DURATION DTime
        OUTPUT(" Plane ", ASK Plane ID, " has delayed its move ");
  END WAIT;

   ASK SpatialTemplet TO ReportPosition(SELF);

END METHOD;


 ASK METHOD NewArrive;
 VAR

    OutSpeed,OutXDest,OutYDest : REAL;

 BEGIN

         IF (SimTime() =6.0)
             Status:= "ATFINALDESTINATION";
             DISPOSE(SELF);

         ELSE
           ASK MovPlanner TO
              NextPlan(OutSpeed,OutXDest,OutYDest);
        Speed:= OutSpeed;
          XPos:= XDest;
          YPos:= YDest;
        Velocity:= Speed;
        XDest:=  OutXDest;
        YDest:=  OutYDest;

       { tranTime is the total flight time;
        the first time it is calculated it is the travel
            time from original position to destination;
        this time will change if the flight is interrupted;
        the est field will contain the original time;
```

```
      }

       ASK SELF TO Direction;
       Status := "NEWPOSITION";
         ETA := ASK SELF TO CalcTime();
       ASK SpatialTemplet TO ReportPosition(SELF);

     END IF;

END METHOD;

TELL METHOD NewArrive1(IN Indicator :STRING);
 VAR

    OutSpeed,OutXDest,OutYDest : REAL;
        localName : STRING;

BEGIN
        localName := ASK SELF NameID;

        IF (SimTime() =6.0)
            Status:= "ATFINALDESTINATION";
            DISPOSE(SELF);

        ELSIF (localName = "Aircraft")
          ASK MovPlanner TO
             NextPlan(OutSpeed,OutXDest,OutYDest);
        Speed:= OutSpeed;

        Velocity:= Speed;
        XDest:=  OutXDest;
        YDest:=  OutYDest;

       { tranTime is the total flight time;
        the first time it is calculated it is the travel
           time from original position to destination;
        this time will change if the flight is interrupted;
        the est field will contain the original time;
       }

        ASK SELF TO Direction;
        Status := "NEWPOSITION";
          ETA := ASK SELF TO CalcTime();

        ASK SpatialTemplet TO ReportPosition(SELF);
    END IF;

 END METHOD;


 ASK METHOD CalcTime() : REAL;{  remaining time in Flight }
 BEGIN
   Dist:= SQRT(POWER((XPos-XDest),2.0) +
            POWER((YPos-YDest),2.0));

   RETURN (ABS(Dist/Speed));
 END METHOD;

 ASK METHOD NewTime(IN XnextDest,YnextDest:REAL):REAL;
  VAR
    YCurrentPos,
    XCurrentPos   : REAL;
     Distravel,
    Theta  : REAL;

  BEGIN
    { figure out where you are}

        Theta:= ATAN((ETA* Speed)/(YDest-YPos));
          Distravel:= (SimTime() - TimeStartFlight) *Speed;
          YCurrentPos:= Distravel*SIN (Theta) + YPos;
          XCurrentPos:= Distravel*COS (Theta) + XPos;
   RETURN(( SQRT(POWER((XCurrentPos-XnextDest),2.0) +
            POWER((YCurrentPos-YnextDest),2.0)))/Speed);
  END METHOD;

  ASK METHOD Comp;
  BEGIN
  XComp := XDest-XPos;
  YComp := YDest-YPos;
  END METHOD;

  ASK METHOD Direction;


          VAR
           Hypotenuse:  REAL;

          BEGIN
          Hypotenuse := DIST2D ( XPos, YPos, XDest, YDest );
          XComp := (XDest - XPos ) / Hypotenuse;
          YComp := (YDest - YPos ) / Hypotenuse;
          Heading := ASIN(YComp);

          IF ( XComp < 0.0 ) Heading := Pi - Heading;
             END IF;

          IF ( (XComp>0.0) AND (YComp<0.0) ) Heading := Heading + 2.0*Pi;
               END IF;

        Degree  := (Heading*180.0)/Pi;
END METHOD;

ASK METHOD GiveName() :STRING;
  BEGIN
    RETURN MyName;
END METHOD;

ASK METHOD NameMover(IN Namin:STRING);
 BEGIN
  MyName := Namin;

END METHOD;

END OBJECT;
{-----------------------------------------------------------------------}
OBJECT PlannerObj;

ASK METHOD ObjInit;
BEGIN
 NEW(ranNumGen);
END METHOD;


ASK METHOD NextPlan(OUT OUTSpeed, OUTXDest, OUTYDest : REAL);
BEGIN
        OUTSpeed :=  5.0 *(ASK ranNumGen Exponential(meanarrivaltime))  ;
      OUTXDest :=  (xrange- 4.0) * ((ASK ranNumGen Sample()) - 0.5);
      OUTYDest :=  (yrange- 4.0) * ((ASK ranNumGen Sample()) - 0.5);



END METHOD;
```

```
ASK METHOD FirstPosit(OUT X,Y : REAL);
BEGIN
 X:= (xrange-5.0) * ((ASK ranNumGen Sample()) - 0.5);
 Y:= (yrange-5.0) * ( (ASK ranNumGen Sample()) - 0.5);
END METHOD;


ASK METHOD IdentifyPlaneInstance
     (IN InputID : INTEGER) : PlaneObj;
VAR
 PlaneInstance : PlaneObj;
 FoundIt      : STRING;
BEGIN
 FoundIt := "NO";
 PlaneInstance := ASK PlaneQue TO First();
 WHILE( PlaneInstance <> NILOBJ) AND (FoundIt = "NO");
  IF (ASK PlaneInstance ID = InputID)
    FoundIt := "YES";
  ELSE
    PlaneInstance := ASK PlaneQue TO Next(PlaneInstance);
  END IF;
 END WHILE;
 RETURN PlaneInstance;
END METHOD;
END OBJECT;
{-----------------------------------------------------------------------}
PROCEDURE ReportEstimatedTime(IN ID : INTEGER;
                IN RETxNextDest,RETyNextDest : REAL;
                 OUT RETestTime : REAL);
VAR
 PlaneInstance :PlaneObj;
 BEGIN
  PlaneInstance := ASK MovPlanner TO IdentifyPlaneInstance(ID);
  IF PlaneInstance = NILOBJ;
    RETestTime := -1.0;
  ELSE
    RETestTime := ASK PlaneInstance TO
                NewTime(RETxNextDest,RETyNextDest);
  END IF;
 END PROCEDURE;
{-----------------------------------------------------------------------}
PROCEDURE ModelInit;
 {VAR
 Index :INTEGER;}


{ these vars are defined in DACMove.mod and are global to this
   module: MovPlanner: PlannerObj;
         Red,Blue,Green,Black : PlaneObj;
         PlaneQue : QueueObj;
 }

 BEGIN
  {create mov planner; it has an obj init (creates a RN generator)}
  NEW(MovPlanner);

 {creat a queue obj}
  NEW(PlaneQue);

  INC(ObjectNumber);
  NEW(Red);
  ASK Red TO InitPlaneFields(ObjectNumber,"Red","Aircraft");
  ASK Red TO NameMover("Red");
  ASK PlaneQue TO Add(Red);
  ASK SpatialTemplet TO ReportPosition(Red);


  INC(ObjectNumber);
  NEW(Blue);
  ASK Blue TO InitPlaneFields(ObjectNumber,"Blue","Aircraft");
  ASK  Blue TO NameMover("Blue");
  ASK PlaneQue TO Add(Blue);
  ASK SpatialTemplet TO ReportPosition(Blue);


  INC(ObjectNumber);
  NEW(Green);
  ASK Green TO InitPlaneFields(ObjectNumber,"Green","Aircraft");
  ASK Green TO NameMover("Green");
  ASK PlaneQue TO Add(Green);
  ASK SpatialTemplet TO ReportPosition(Green);


  INC(ObjectNumber);
  NEW(Black);
  ASK Black TO InitPlaneFields(ObjectNumber,"Black","Aircraft");
  ASK Black TO NameMover("Black");
  ASK PlaneQue TO Add(Black);
  ASK SpatialTemplet TO ReportPosition(Black);


  END PROCEDURE;
END MODULE.
```

```
DEFINITION MODULE ACVer2;

FROM SimMod IMPORT StartSimulation, SimTime;
FROM RandMod IMPORT RandomObj, FetchSeed, Random;
FROM GrpMod IMPORT QueueObj;
FROM GTypes IMPORT PointArrayType;
FROM Fill IMPORT PolygonObj;
FROM Image IMPORT ImageObj;
FROM Window IMPORT WindowObj;
FROM Animate IMPORT DynDClockObj;
FROM AuxMathMod IMPORT DIST2D, INT;
FROM Graphic IMPORT GraphicLibObj;

TYPE

MovingObj = OBJECT; FORWARD ;

ControllerObj = OBJECT; FORWARD;

PlaneInSectorStatus = ( ABSENT , PRESENT , ENVELOPE ) ;

SectorObj = OBJECT(QueueObj)
        sectorIDx , sectorIDy : INTEGER;
        xmin,ymin,xmax,ymax : REAL;
        vertex : PointArrayType ;
        Polygon : PolygonObj;
      ASK METHOD SetobjectID(IN i,j:INTEGER);
        ASK METHOD SetBorders;
        ASK METHOD ChangeSectorStatus ( IN Plane : MovingObj ;
                IN Status : PlaneInSectorStatus ) ;
        ASK METHOD Find ( IN Plane1 , Plane2 : MovingObj ) ;
        TELL METHOD MonitorSector(IN control:ControllerObj);
        TELL METHOD HighlightPath( IN DeltaT:REAL);

END OBJECT;


SectorQueueObj = OBJECT(QueueObj)
        ASK METHOD LogicalAdd ( IN elem1: ANYOBJ );
END OBJECT;

MovingObj = OBJECT

        objectID: INTEGER ;
        XDest, YDest:  REAL;
        velocity:REAL;
        XPos,YPos: REAL;
        vertex : PointArrayType ;
        XStart,YStart:REAL;
        XComp, YComp: REAL;
        Heading:REAL;
        location: STRING;
        ETA, TInc, TLast : REAL;
        SectorQueue:  SectorQueueObj;
        SectorArray:  ARRAY INTEGER, INTEGER OF PlaneInSectorStatus;
        SectorEnvQueue:  SectorQueueObj;
        EnvelopePolygon: PolygonObj;
        PlaneIcon : ImageObj;
        PlaneGraphic:ImageObj;
        ASK METHOD SetobjectID(IN numofplanes:INTEGER);
        ASK METHOD SetOrigination (IN XValue,YValue: REAL);
        ASK METHOD SetSpeed ( IN speed:  REAL);
        ASK METHOD SetCourse (IN XValue,YValue:  REAL);
        ASK METHOD SetLocation(IN loc:  STRING);
        ASK METHOD InitSectorQueue;
        ASK METHOD InitEnvelope;
        ASK METHOD UpdatePosition;
        ASK METHOD UpdateEnvelope;
        ASK METHOD CHECK;
        TELL METHOD Fly;
        ASK METHOD UpdateSectors;
        ASK METHOD UpdateEnvSectors;
        ASK METHOD UpdateEnvSectorsOld;
        TELL METHOD Talk(IN Plane1, Plane2:MovingObj);
        TELL METHOD Wait;
        TELL METHOD ReduceSpeed;
END OBJECT;

ControllerObj = OBJECT


        ConObjID: INTEGER;
        {CXPos : REAL;}
        TELL METHOD Inform(IN Plane1,Plane2:MovingObj);
        ASK METHOD SetControllerID(IN I:INTEGER);
        {ASK METHOD ConPlanes(IN Con:ANYOBJ);}

END OBJECT;
```

```
GeneratorObj = OBJECT
   TELL METHOD GenPlanes;
   TELL METHOD GenSectors;
  ASK METHOD GenController;

 END OBJECT;


VAR

 numofhrs,
 meanarrivaltime: REAL;

 movingobjgen: MovingObj;

 numofplanes:INTEGER;
 MasterSectorArray: ARRAY INTEGER , INTEGER OF SectorObj;
 MasterGraphic: ImageObj;
 sectorGenerator: GeneratorObj;
 planeGenerator: GeneratorObj;
 controlGenerator: GeneratorObj;
 planenames: QueueObj;
 ranNumGen: RandomObj;
Controller:ControllerObj;
 window: WindowObj;
 clockwindow:  WindowObj;
 clockgraphic: DynDClockObj;
 xrange, yrange: REAL;
 numofsecx,
 numofsecy: INTEGER;
 MasterPlaneIcon: ImageObj;
 GraphicLib : GraphicLibObj;
 Pi:REAL;
 END MODULE.
```

```
IMPLEMENTATION MODULE ACVer2;


FROM SimMod IMPORT StartSimulation, SimTime;
FROM RandMod IMPORT RandomObj, FetchSeed, Random;
FROM MathMod IMPORT SQRT,ASIN;
FROM GrpMod IMPORT QueueObj;
FROM GTypes IMPORT PointArrayType;
FROM GTypes IMPORT ColorType(Green,Yellow,Blue,Red,Orange,Violet,White);
FROM GTypes IMPORT FillStyleType(HollowFill,SolidFill,NarrowDiagonalFill,
        MediumDiagonalFill,WideDiagonalFill,NarrowCrosshatchFill,
        MediumCrosshatchFill,WideCrosshatchFill);
FROM Fill IMPORT PolygonObj;
FROM Image IMPORT ImageObj;
FROM AuxMathMod IMPORT DIST2D, INT;
FROM Graphic IMPORT GraphicLibObj;
FROM UtilMod IMPORT Delay;
FROM Line IMPORT PolylineObj;
FROM GTypes IMPORT LineStyleType(SolidLine);

OBJECT SectorQueueObj;

        ASK METHOD LogicalAdd (IN elem1:  ANYOBJ);
        VAR
                elem2: ANYOBJ;
                status: (PRESENT , ABSENT );
        BEGIN
                elem2 := ASK SELF First();
                status := ABSENT;
                WHILE ( (elem2<>NILOBJ) AND (status=ABSENT) )
                        IF (elem2=elem1) status:=PRESENT;
                                END IF;
                        elem2 := ASK SELF Next(elem2);
                        END WHILE;
                IF (status=ABSENT) ASK SELF TO Add(elem1);
                        END IF;
        END METHOD;


END OBJECT;
```

```
OBJECT SectorObj;

ASK METHOD SetobjectID (IN i,j: INTEGER);
BEGIN
  sectorIDx := i;
  sectorIDy := j;
END METHOD;

ASK METHOD SetBorders;
BEGIN
  xmin := xrange * ( FLOAT(sectorIDx-1)/FLOAT(numofsecx) - 0.5 ) ;
  xmax := xrange * ( FLOAT(sectorIDx)/FLOAT(numofsecx) - 0.5 ) ;
  ymin := yrange * ( FLOAT(sectorIDy-1)/FLOAT(numofsecy) - 0.5 ) ;
  ymax := yrange * ( FLOAT(sectorIDy)/FLOAT(numofsecy) - 0.5 ) ;
  NEW(vertex, 1..4);
  vertex[1].x := xmin ; vertex[1].y := ymin ;
  vertex[2].x := xmax ; vertex[2].y := ymin ;
  vertex[3].x := xmax ; vertex[3].y := ymax ;
  vertex[4].x := xmin ; vertex[4].y := ymax ;
  {OUTPUT("From Sector Object the vertex values are: vertex[1].x ", vertex[1].x, "
vertex[1].y = ", vertex[1].y, "vertex[2].x = ", vertex[2].x," vertex[2].y = ", vertex[2].y,
"vertex[3].x = ", vertex[3].x, " vertex[3].y = ", vertex[3].y, "vertex[4].x = ", vertex[4].x,"
vertex[4].y = ", vertex[4].y);}
   NEW(Polygon);
   ASK Polygon TO SetPoints(vertex);
   ASK Polygon TO SetColor(Yellow);
   ASK Polygon TO SetStyle(HollowFill);
   ASK MasterGraphic TO AddGraphic(Polygon);
 END METHOD;

 TELL METHOD MonitorSector(IN control: ControllerObj);

 VAR
   plane1, plane2 : MovingObj;

   cid:INTEGER;
 BEGIN

 (*** New part that constructs lines between all the planes****)
         plane1 := ASK SELF First();
         WHILE (plane1 <> NILOBJ)
            plane2 := ASK SELF Next(plane1);
                WHILE (plane2 <> NILOBJ)
                 {ASK SELF Find (plane1, plane2);}
            plane2 := ASK SELF Next(plane2);
           END WHILE;
       plane1 := ASK SELF Next(plane1);
     END WHILE;



WHILE ( SimTime() <  numofhrs*60.0 )

 (*** Here is where the control/collision algorithm will go; for now,
      issue an alert if two aircraft are simply in the same sector.  ***)

   plane1 := ASK SELF First();
   IF ( plane1 <> NILOBJ ) plane2 := ASK SELF Next(plane1);
        END IF;
   IF ( plane2 <> NILOBJ )
             {ASK SELF TO Find(plane1,plane2);}
        OUTPUT;
        OUTPUT(" ***************** DANGER!!  Possible collision! **********");
        OUTPUT("   Planes ",ASK plane1 objectID," and ",ASK plane2 objectID,
             " are both in or near Sector (",sectorIDx,",",
               sectorIDy,") at time ", SimTime());
        OUTPUT("   Plane ",ASK plane1 objectID,":  (",
                            ASK plane1 XPos," , ",ASK plane1 YPos,")");
        OUTPUT("   Plane ",ASK plane2 objectID,":  (",
                            ASK plane2 XPos," , ",ASK plane2 YPos,")");
        OUTPUT;
       (**** When the intersection is detected, the plane is asked to talk to
         the other plane ********)
       TELL plane1 TO Talk(plane1,plane2);
       cid:= ASK control ConObjID;
       OUTPUT("CID ", cid);
       TELL control TO Inform(plane1,plane2);


        END IF;
    WAIT DURATION 2.0
        ON INTERRUPT
        OUTPUT("Sector ",sectorIDx,",",sectorIDy," was interrupted at ",
               SimTime());
        END WAIT;
 END WHILE;


END METHOD;
```

```
TELL METHOD HighlightPath( IN DeltaT:REAL);

BEGIN
   WAIT DURATION DeltaT
     END WAIT;
   ASK window TO SetColor(Violet);
   ASK  window TO Draw();
    Delay (1);
   ASK window TO SetColor (Blue);
   ASK window TO Draw();
END METHOD;

ASK METHOD ChangeSectorStatus ( IN Plane : MovingObj ;
        IN Status : PlaneInSectorStatus ) ;

VAR
        elem : MovingObj ;

BEGIN
        (*** Note:  this will have to be redone when the envelope and
        airplane are colored differently, so that one plane does not
        clobber another one's graphics. ***)

        IF (Status = ABSENT)
                {OUTPUT("Changing status of sector ",sectorIDx,",",sectorIDy,
                        " to ABSENT for plane ",ASK Plane objectID);}
                elem := ASK SELF First();
                WHILE (elem <> NILOBJ)
                        OUTPUT("old queue contains:  ",ASK elem objectID);
                        elem := ASK SELF Next(elem);
                        END WHILE;
                END IF;


        IF ( Status = ABSENT )
                OUTPUT("Removing plane ",ASK Plane objectID);
                elem := ASK SELF First();
                WHILE (elem <> NILOBJ)
                        OUTPUT("Before remove, queue contains:  ",
                                ASK elem objectID);
                        elem := ASK SELF Next(elem);
                        END WHILE;
```

```
            IF(ASK SELF Includes(Plane)) OUTPUT("It is there.");
                    END IF;
            ASK SELF TO RemoveThis ( Plane ) ;
            IF(ASK SELF Includes(Plane)) OUTPUT("It is  still there.");
                    END IF;



            elem := ASK SELF First();
            WHILE (elem <> NILOBJ)
                    OUTPUT("After remove, queue contains:  ",
                            ASK elem objectID);
                    elem := ASK SELF Next(elem);
                    END WHILE;

            IF ( ASK SELF First() = NILOBJ )
                    ASK Polygon TO SetColor(Yellow) ;
                    ASK Polygon TO SetStyle(HollowFill) ;
                    ASK Polygon TO Draw() ;
                    END IF ;
        ELSE
            IF ( Status = PRESENT )
                    ASK Polygon TO SetColor(Red) ;
                    ASK Polygon TO SetStyle(SolidFill) ;
                    ASK Polygon TO Draw() ;
                    END IF ;
            IF ( Status = ENVELOPE )
                    ASK Polygon TO SetColor(Red) ;
                    ASK Polygon TO SetStyle(NarrowDiagonalFill) ;
                    ASK Polygon TO Draw() ;
                    END IF ;
            elem := ASK SELF First();
            WHILE ( (elem <> NILOBJ) AND (elem <> Plane) )
                    elem := ASK SELF Next(elem) ;
                    END WHILE ;
            IF (elem = NILOBJ ) ASK SELF TO Add(Plane) ;
                    END IF;
            END IF;
IF (Status = ABSENT)
            OUTPUT("New list of sector ",sectorIDx,",",sectorIDy,
                    " is");
            elem := ASK SELF First();
            WHILE (elem <> NILOBJ)
                    OUTPUT("new queue contains:  ",ASK elem objectID);
                    elem := ASK SELF Next(elem);
                    END WHILE;
                END IF;


        END METHOD;



        ASK METHOD Find ( IN Plane1 , Plane2 : MovingObj ) ;

        VAR
            Epsilon :  REAL;
            XComp1 , XComp2 : REAL ;
            YComp1 , YComp2 : REAL ;
            XPos1 , XPos2 : REAL ;
            YPos1 , YPos2 : REAL ;
            vel1 , vel2 : REAL ;
            a , b , c , d : REAL ;
            MinDist , t : REAL ;
            points: PointArrayType;
            line: PolylineObj;

        BEGIN
            OUTPUT("from find for planeid",ASK Plane1 objectID,"plane2id",ASK Plane2
        objectID);
            Epsilon := 0.0001;
            XComp1 := ASK Plane1 XComp ;    YComp1 := ASK Plane1 YComp ;
        OUTPUT("From FIND  for plane ", ASK Plane1 objectID,"XComp1,YComp1
        ",XComp1,YComp1);
            XPos1 := ASK Plane1 XPos ; YPos1 := ASK Plane1 YPos ;
            OUTPUT("From FIND  for plane ", ASK Plane1 objectID,"XPos1,YPos1
        ",XPos1,YPos1);

            XComp2 := ASK Plane2 XComp ;    YComp2 := ASK Plane2 YComp ;
        OUTPUT("From FIND  for plane ", ASK Plane2 objectID,"XComp2,YComp2
        ",XComp2,YComp2);

            XPos2 := ASK Plane2 XPos ; YPos2 := ASK Plane2 YPos ;
        OUTPUT("From FIND  for plane ", ASK Plane2 objectID,"XPos2,YPos2
        ",XPos2,YPos2);
            vel1 := ASK Plane1 velocity ; vel2 := ASK Plane2 velocity ;
        OUTPUT("From FIND  for plane ", ASK Plane2 objectID,"vel1,vel2 ",vel1,vel2);
            (*** Test to see if paths are either parallel or antiparallel. ***)
```

```
{IF ( ( (ABS(XComp1) - ABS(XComp2)) < Epsilon) AND
            ( (ABS(YComp1) - ABS(YComp2)) < Epsilon ) )}

IF ( ( ABS( ABS(XComp1) - ABS(XComp2) ) < Epsilon ) AND
            ( ABS( ABS(YComp1) - ABS(YComp2) ) < Epsilon ) )
      OUTPUT;
      OUTPUT (" Paths are either parallel or antiparralel");
      MinDist := XComp1 *
            ( YPos1 - YPos2 + XComp1/YComp1*(XPos1 - XPos2) );
      OUTPUT("Paths are either parallel or antiparallel; distance = ",
            MinDist);
   ELSE
      a := XPos2 - XPos1 ;  c := YPos2 - YPos1 ;
      b := vel2*XComp2 - vel1*XComp1 ;
      d := vel2*YComp2 - vel1*YComp1 ;

      t := -(a*b + c*d)/(b*b + d*d) ;
      MinDist := SQRT( (a+b*t)*(a+b*t) + (c+d*t)*(c+d*t) ) ;
      OUTPUT("From sector (",sectorIDx,",",sectorIDy,", planes ",
            ASK Plane1 objectID," and ",ASK Plane2 objectID,
            "will have a minimum distance of ",MinDist,
            "at time ",t, " Simulation time= ", SimTime());
      END IF;


   NEW ( points , 1..2);
   NEW ( line );
   points[1].x := XPos1 + vel1*XComp1*t;
   points[1].y := YPos1 + vel1*YComp1*t;
   points[2].x := XPos2 + vel2*XComp2*t;
   points[2].y := YPos2 + vel2*YComp2*t;
      OUTPUT("Coordinates of the shortest distance are:");
      OUTPUT(points[1].x,",",points[1].y);
      OUTPUT(points[2].x,",",points[2].y);
   ASK line TO SetPoints(points);
   ASK line TO SetStyle(SolidLine);
   ASK line TO SetColor(White);
   ASK MasterGraphic TO AddGraphic(line);
   ASK line TO Draw();
   {TELL SELF TO HighlightPath(t);}

END METHOD;
END OBJECT;

OBJECT MovingObj;


          ASK METHOD SetobjectID (IN numofplanes:INTEGER);
          BEGIN
          objectID:= numofplanes;
          END METHOD;

          ASK METHOD SetOrigination (IN XValue , YValue : REAL);
          BEGIN
            XStart := XValue;
            YStart := YValue;
            XPos := XValue;
            YPos := YValue;
          END METHOD;

          ASK METHOD SetSpeed ( IN speed : REAL );

          BEGIN
            velocity := speed;
          END METHOD;

          TELL METHOD ReduceSpeed;
           VAR
            redvel: REAL;

          BEGIN
          redvel:= ASK SELF velocity;
          velocity:= redvel -0.2;
          WAIT DURATION 3.0;
          END WAIT;
          velocity:= redvel;
          END METHOD;


          ASK METHOD SetCourse ( IN XValue , YValue : REAL );
          VAR Hypotenuse:  REAL;
          BEGIN
            XDest := XValue ;

            YDest := YValue ;
            Hypotenuse := DIST2D ( XPos, YPos, XDest, YDest );
            XComp := (XDest - XPos ) / Hypotenuse;
          YComp := (YDest - YPos ) / Hypotenuse;
          Heading := ASIN(YComp);
               OUTPUT("From SetCourse Plane ", ASK SELF objectID, "XDest ",
XDest," YDest ", YDest, "Hpotenuse ", Hypotenuse, " XComp ",XComp,
"YComp",YComp, "Heading ",Heading );

          IF ( XComp < 0.0 ) Heading := Pi - Heading; END IF;
          IF ( (XComp>0.0) AND (YComp<0.0) ) Heading := Heading + 2.0*Pi;
               END IF;
          ASK SELF TO UpdateEnvelope;
        END METHOD;

        ASK METHOD SetLocation ( IN loc: STRING);
        BEGIN
          location := loc;
        END METHOD;

        ASK METHOD InitSectorQueue;
        VAR
                i , j : INTEGER ;
        BEGIN
          NEW ( SectorQueue );
          NEW ( SectorEnvQueue );
          NEW ( SectorArray , 1 .. numofsecx , 1 .. numofsecy ) ;
          FOR j := 1 TO numofsecy
                FOR i := 1 TO numofsecx
                        SectorArray [ i , j ] := ABSENT ;
                END FOR ;
          END FOR ;
        END METHOD;

        ASK METHOD InitEnvelope;
        VAR sx,sy:REAL;
        BEGIN
          NEW ( vertex , 1..4);
          NEW ( PlaneGraphic );
          NEW ( EnvelopePolygon );
          ASK EnvelopePolygon TO SetStyle(HollowFill);
          NEW ( PlaneIcon ) ;
          ASK GraphicLib TO ReadFromFile("AConIcon.lib");
          ASK PlaneIcon TO LoadFromLibrary(GraphicLib,"PlaneIcon");
          ASK PlaneGraphic TO AddGraphic(PlaneIcon);
          ASK PlaneGraphic TO AddGraphic ( EnvelopePolygon ) ;
          ASK MasterGraphic TO AddGraphic ( PlaneGraphic);
          ASK PlaneIcon Scaling(sx,sy);
          ASK PlaneIcon Scale(sx/300.0,sy/300.0);
        END METHOD;

        ASK METHOD UpdateEnvelope;
        BEGIN
          vertex[1].x := XPos - 2.0*YComp ;
          vertex[1].y := YPos + 2.0*XComp ;
          vertex[2].x := XPos + 2.0*YComp ;
          vertex[2].y := YPos - 2.0*XComp ;
          vertex[3].x := XDest + 2.0*YComp ;
          vertex[3].y := YDest - 2.0*XComp ;
          vertex[4].x := XDest - 2.0*YComp ;
          vertex[4].y := YDest + 2.0*XComp ;
        {OUTPUT("From the Moving Object UpdateEnvelope routine ", "vertex[1].x =",
vertex[1].x,"vertex[1].y =", vertex[1].y,"vertex[2].x =", vertex[2].x,"vertex[2].y =",
vertex[2].y,"vertex[3].x =", vertex[3].x,"vertex[3].y =", vertex[3].y,"vertex[4].x =",
vertex[4].x,"vertex[4].y =", vertex[4].y);}
          ASK EnvelopePolygon TO SetPoints(vertex);
          ASK PlaneIcon TO SetTranslation(XPos,YPos);
          ASK PlaneIcon TO Rotate(Heading-Pi/2.0);
          ASK PlaneGraphic TO Draw();
        END METHOD;


ASK METHOD UpdateEnvSectors;

VAR
  queueOld, queueNew:  SectorQueueObj;
  loopsec, secA, secB, secAtemp:  SectorObj;
  secnum, loopsecnum: INTEGER;
  REMOVED:  BOOLEAN;
  qx, qy, xstart, ystart, xstop, ystop: INTEGER;
  xscale, yscale:  REAL;
  xsector,ysector : INTEGER;
  xborder,yborder: REAL;
  m , b , yint,xint : REAL;
  NewSector : ARRAY INTEGER , INTEGER OF PlaneInSectorStatus ;
  i,j : INTEGER;

BEGIN

  (***Initialize the NewSector array ***)
  NEW ( NewSector , 1 .. numofsecx , 1 .. numofsecy ) ;
  FOR j := 1 TO numofsecy
    FOR i := 1 TO numofsecx
```

```
        NewSector [ i , j ] := ABSENT ;
      END FOR ;
    END FOR ;


    (*** Determine the new list of sectors ***)

    FOR i := 1 TO 4
      j := (i MOD 4) + 1;


      (*** Define m and b for the line equation y=mx+b only for non-vertical
            lines   ***)
      IF ( vertex[i].x <> vertex[j].x )
          m := ( vertex[j].y - vertex[i].y ) / ( vertex[j].x - vertex[i].x ) ;
          b := vertex[i].y - m*vertex[i].x ;
          {OUTPUT(" From Moving Object Update EnvelopeSector vertex[i]= ",
  vertex[i].x, " Vertex[i].y= ", vertex[i].y);}
                  END IF;

          (*** first, add the "x pairs"  if line is not vertical ***)
      IF ( vertex[i].x <> vertex[j].x )
      FOR xsector := 2 TO numofsecx
                  xborder := xrange * ( FLOAT(xsector-1)/FLOAT(numofsecx) - 0.5 );
          IF ((MINOF (vertex[i].x,vertex[j].x) < xborder) AND
              (MAXOF(vertex[i].x,vertex[j].x) > xborder))

                  yint := m*xborder + b ;
                  yborder := -yrange/2.0;
                  ysector := 1;
                  WHILE ( yborder < yint )
                          ysector := ysector + 1 ;
                          yborder := yrange *
                                  ( FLOAT(ysector-1)/FLOAT(numofsecy) - 0.5 );
                          END WHILE;

                  qx := xsector-1;
                  qy := ysector-1;
                  NewSector [ qx , qy ] := ENVELOPE ;
                  qx := xsector;
                  NewSector [ qx , qy ] := ENVELOPE ;
                  END IF;
          END FOR;
          END IF;
    (*** now add the "y pairs" ***)
   FOR ysector := 2 TO numofsecy
        yborder := yrange * (FLOAT (ysector-1)/FLOAT(numofsecy) - 0.5 );
        IF ( (MINOF(vertex[i].y, vertex[j].y) < yborder) AND
              (MAXOF(vertex[i].y, vertex[j].y) > yborder) )
              IF ( vertex[i].x <> vertex[j].x )
                     xint := (yborder-b)/m ;
                 ELSE
                     xint := vertex[i].x;
                 END IF;
              xborder := -xrange/2.0;
              xsector := 1;
              WHILE ( xborder < xint )
                     xsector := xsector + 1;
                     xborder := xrange *
                            ( FLOAT (xsector-1)/FLOAT(numofsecx) - 0.5 );
                     END WHILE;
              qx := xsector -1;
              qy := ysector -1;
              NewSector [ qx , qy ] := ENVELOPE ;
              qy := ysector;
              NewSector [ qx , qy ] := ENVELOPE ;
              END IF;
      END FOR;
  END FOR;

  FOR j := 1 TO numofsecy
    FOR i := 1 TO numofsecx
      IF ( SectorArray [ i , j ] <> NewSector [ i , j ] )
              SectorArray [ i , j ] := NewSector [ i , j ] ;
              ASK MasterSectorArray [i,j] TO
                     ChangeSectorStatus ( SELF , SectorArray [i,j]  );




              END IF;
    END FOR ;
  END FOR ;
END METHOD ;


ASK METHOD UpdateEnvSectorsOld;

VAR
```

```
  queueOld, queueNew:  SectorQueueObj;
  loopsec, secA, secB, secAtemp:  SectorObj;
  secnum, loopsecnum: INTEGER;
  REMOVED:  BOOLEAN;
  qx, qy, xstart, ystart, xstop, ystop: INTEGER;
  xscale, yscale:  REAL;
  xsector,ysector : INTEGER;
  xborder,yborder: REAL;
  m , b , yint,xint : REAL;
  i,j : INTEGER;
       BEGIN

(*** This is to empty the SectorEnvQueue *****)
 secA := ASK SectorEnvQueue First();
   WHILE ( secA <> NILOBJ )
             ASK SectorEnvQueue TO RemoveThis ( secA);
       secA := ASK SectorEnvQueue First();
   END WHILE;


  (*** Set queueOld to be the previous list of sectors   ***)
  NEW ( queueOld );
  secA := ASK SectorQueue First();
  WHILE ( secA <> NILOBJ)
       ASK queueOld TO Add(secA);
     secA := ASK SectorQueue Next(secA);
     END WHILE;


  (*** Determine the new list of sectors ***)
  NEW ( queueNew );

  FOR i := 1 TO 4
    j := (i MOD 4) + 1;


    (*** Define m and b for the line equation y=mx+b only for non-vertical
         lines   ***)
    IF ( vertex[i].x <> vertex[j].x )
       m := ( vertex[j].y - vertex[i].y ) / ( vertex[j].x - vertex[i].x ) ;
       b := vertex[i].y - m*vertex[i].x ;
              END IF;

       (*** first, add the "x pairs"  if line is not vertical ***)
    IF ( vertex[i].x <> vertex[j].x )
```

```
FOR xsector := 2 TO numofsecx
          xborder := xrange * ( FLOAT(xsector-1)/FLOAT(numofsecx) - 0.5 );
   IF ((MINOF (vertex[i].x,vertex[j].x) < xborder) AND
      (MAXOF(vertex[i].x,vertex[j].x) > xborder))

          yint := m*xborder + b ;
          yborder := -yrange/2.0;
          ysector := 1;
          WHILE ( yborder < yint )
                ysector := ysector + 1 ;
                yborder := yrange *
                      ( FLOAT(ysector-1)/FLOAT(numofsecy) - 0.5 );
                END WHILE;

          qx := xsector-1;
          qy := ysector-1;
          ASK SectorEnvQueue TO LogicalAdd ( MasterSectorArray[qx,qy] );
          qx := xsector;
          ASK SectorEnvQueue TO LogicalAdd ( MasterSectorArray[qx,qy] );
          END IF;
   END FOR;
   END IF;

   (*** now add the "y pairs" ***)
FOR ysector := 2 TO numofsecy
    yborder := yrange * (FLOAT (ysector-1)/FLOAT(numofsecy) - 0.5 );
    IF ( (MINOF(vertex[i].y, vertex[j].y) < yborder) AND
         (MAXOF(vertex[i].y, vertex[j].y) > yborder) )
         IF ( vertex[i].x <> vertex[j].x )
                xint := (yborder-b)/m ;
            ELSE
                xint := vertex[i].x;
            END IF;
         xborder := -xrange/2.0;
         xsector := 1;
         WHILE ( xborder < xint )
                xsector := xsector + 1;
                xborder := xrange *
                      ( FLOAT (xsector-1)/FLOAT(numofsecx) - 0.5 );
                END WHILE;
         qx := xsector -1;
         qy := ysector -1;
         ASK SectorEnvQueue TO LogicalAdd ( MasterSectorArray[qx,qy]);
         qy := ysector;
         ASK SectorEnvQueue TO LogicalAdd ( MasterSectorArray[qx,qy]);
```

```
            END IF;
        END FOR;
  END FOR;

  secA := ASK SectorEnvQueue First();
  WHILE ( secA <> NILOBJ)
        ASK (ASK secA Polygon) TO SetColor(Red);
        ASK (ASK secA Polygon) TO SetStyle(NarrowCrosshatchFill);
        ASK (ASK secA Polygon) TO Draw();
        secA := ASK SectorEnvQueue Next(secA);
     END WHILE;



{  xscale := (XPos + xrange/2.0)/xrange*FLOAT(numofsecx);
  yscale := (YPos + yrange/2.0)/yrange*FLOAT(numofsecy);
  qx := INT(xscale)+1;
  qy := INT(yscale)+1;
  OUTPUT("Plane ",objectID," is in sector (",qx,",",qy);
  xstart := qx;
  xstop := qx;
  ystart := qy;
  ystop := qy;
  IF ( ( (xscale - FLOAT(INT(xscale))) <= 0.2 ) AND (qx <> 1) )
        xstart := xstart - 1;
        END IF;
  IF ( ( (yscale - FLOAT(INT(yscale))) <= 0.2 ) AND (qy <> 1) )
        ystart := ystart - 1;
        END IF;
  IF ( ( (xscale - FLOAT(INT(xscale))) >= 0.8 ) AND (qx <> numofsecx ) )
        xstop := xstop + 1;
        END IF;
  IF ( ( (yscale - FLOAT(INT(yscale))) >= 0.8 ) AND (qy <> numofsecy ) )
        ystop := ystop + 1;
        END IF;
  FOR qx := xstart TO xstop
        FOR qy := ystart TO ystop
           ASK queueNew TO Add( MasterSectorArray[qx,qy] );
        END FOR;
  END FOR;


{  secA := ASK queueOld First();
  WHILE ( secA <> NILOBJ)
        OUTPUT("Before cancellations, queueOld:  sector ",ASK secA sectorID);
```

```
      secA := ASK queueOld Next(secA);
      END WHILE;
   secB := ASK queueNew First();
   WHILE ( secB <> NILOBJ)
         OUTPUT("Before cancellations, queueNew:  sector ",ASK secB sectorID);
         secB := ASK queueNew Next(secB);
         END WHILE;}


   (*** Remove from both queueOld and queueNew those sectors which are in
         both queues.                                              ***)

   secA := ASK queueOld First();
   LOOP
    IF ( secA = NILOBJ ) EXIT;
       END IF;
    secB := ASK queueNew First();
    REMOVED := FALSE;
    LOOP
         IF ( secB = NILOBJ ) EXIT;
             END IF;
         IF ( secA = secB )
             secAtemp := secA;
             secA := ASK queueOld Next(secA);
             ASK queueOld TO RemoveThis ( secAtemp ) ;
             ASK queueNew TO RemoveThis ( secB ) ;
             REMOVED := TRUE;
             EXIT;
             END IF;
         secB := ASK queueNew Next(secB);
         END LOOP;
    IF (NOT REMOVED) secA := ASK queueOld Next(secA);
         END IF;
    END LOOP;

    WHILE ( secA <> NILOBJ )
       secB := ASK queueNew First();
       WHILE ( secB <> NILOBJ)
              IF ( secA = secB )
                  ASK queueOld TO RemoveThis ( secA ) ;
                     ASK queueNew TO RemoveThis ( secB ) ;
                     END IF ;
              secB := ASK queueNew Next(secB);
       END WHILE;
       secA := ASK queueOld Next(secA);
```

```
 END WHILE;



{ secA := ASK queueOld First();
 WHILE ( secA <> NILOBJ)
      OUTPUT("After cancellations, queueOld:  sector ",ASK secA sectorID);
    secA := ASK queueOld Next(secA);
    END WHILE;
 secB := ASK queueNew First();
 WHILE ( secB <> NILOBJ)
      OUTPUT("After cancellations, queueNew:  sector ",ASK secB sectorID);
      secB := ASK queueNew Next(secB);
      END WHILE;}




  (*** Inform any sectors remaining in queueOld that this plane is no longer            in
its airspace. and remove the sector from the plane's SectorQueue.
                                                                   ***)

  secA := ASK queueOld First();
  WHILE ( secA <> NILOBJ )
       ASK secA TO RemoveThis ( SELF );
    ASK SectorQueue TO RemoveThis ( secA);
       secA := ASK queueOld Next(secA);
  END WHILE;

  (*** Inform any sectors remaining in queueNew that this plane has entered
       its airspace, and add the sector to the plane's SectorQueue.  ***)

  secB := ASK queueNew First();
  WHILE ( secB <> NILOBJ)
       ASK secB TO Add ( SELF );
       ASK SectorQueue TO Add ( secB );
       secB := ASK queueNew Next(secB);
 END WHILE;
}
END METHOD;
```

```
ASK METHOD UpdateSectors;

VAR
  queueOld, queueNew:  QueueObj;
  loopsec, secA, secB, secAtemp:  SectorObj;
  secnum, loopsecnum: INTEGER;
  REMOVED:  BOOLEAN;
  qx, qy, xstart, ystart, xstop, ystop: INTEGER;
  xscale, yscale:  REAL;

BEGIN

  (*** Set queueOld to be the previous list of sectors   ***)
  NEW ( queueOld );
  secA := ASK SectorQueue First();
  WHILE ( secA <> NILOBJ)
        ASK queueOld TO Add(secA);
     secA := ASK SectorQueue Next(secA);
     END WHILE;

  (*** Determine the new list of sectors ***)
  NEW ( queueNew );
        {OUTPUT("XPos,YPos = ",XPos,",",YPos);}
  xscale := (XPos + xrange/2.0)/xrange*FLOAT(numofsecx);
  yscale := (YPos + yrange/2.0)/yrange*FLOAT(numofsecy);
  qx := INT(xscale)+1;
  qy := INT(yscale)+1;
  OUTPUT("Plane ",objectID," is in sector ",qx,",",qy);
  xstart := qx;
  xstop := qx;
  ystart := qy;
  ystop := qy;
  IF ( ( (xscale - FLOAT(INT(xscale))) <= 0.2 ) AND (qx <> 1) )
        xstart := xstart - 1;
        END IF;
  IF ( ( (yscale - FLOAT(INT(yscale))) <= 0.2 ) AND (qy <> 1) )
        ystart := ystart - 1;
        END IF;
  IF ( ( (xscale - FLOAT(INT(xscale))) >= 0.8 ) AND (qx <> numofsecx ) )
        xstop := xstop + 1;
        END IF;
  IF ( ( (yscale - FLOAT(INT(yscale))) >= 0.8 ) AND (qy <> numofsecy ) )
        ystop := ystop + 1;
        END IF;
```

```
FOR qx := xstart TO xstop
      FOR qy := ystart TO ystop
          ASK queueNew TO Add(MasterSectorArray[qx,qy]);
      END FOR;
END FOR;


{  secA := ASK queueOld First();
 WHILE ( secA <> NILOBJ)
       OUTPUT("Before cancellations, queueOld:  sector ",ASK secA sectorID);
    secA := ASK queueOld Next(secA);
    END WHILE;
 secB := ASK queueNew First();
 WHILE ( secB <> NILOBJ)
       OUTPUT("Before cancellations, queueNew:  sector ",ASK secB sectorID);
       secB := ASK queueNew Next(secB);
       END WHILE;}


 (*** Remove from both queueOld and queueNew those sectors which are in
       both queues.                                                ***)

 secA := ASK queueOld First();
 LOOP
   IF ( secA = NILOBJ ) EXIT;
      END IF;
   secB := ASK queueNew First();
   REMOVED := FALSE;
   LOOP
       IF ( secB = NILOBJ ) EXIT;
           END IF;
       IF ( secA = secB )
           secAtemp := secA;
           secA := ASK queueOld Next(secA);
           ASK queueOld TO RemoveThis ( secAtemp ) ;
           ASK queueNew TO RemoveThis ( secB ) ;
           REMOVED := TRUE;
           EXIT;
           END IF;
       secB := ASK queueNew Next(secB);
       END LOOP;
   IF (NOT REMOVED) secA := ASK queueOld Next(secA);
       END IF;
   END LOOP;
 WHILE ( secA <> NILOBJ )
   secB := ASK queueNew First();
   WHILE ( secB <> NILOBJ)
         IF ( secA = secB )
             ASK queueOld TO RemoveThis ( secA ) ;
                ASK queueNew TO RemoveThis ( secB ) ;
                END IF ;
         secB := ASK queueNew Next(secB);
   END WHILE;
   secA := ASK queueOld Next(secA);
 END WHILE;




{  secA := ASK queueOld First();
 WHILE ( secA <> NILOBJ)
       OUTPUT("After cancellations, queueOld:  sector ",ASK secA sectorID);
    secA := ASK queueOld Next(secA);
    END WHILE;
 secB := ASK queueNew First();
 WHILE ( secB <> NILOBJ)
       OUTPUT("After cancellations, queueNew:  sector ",ASK secB sectorID);
       secB := ASK queueNew Next(secB);
       END WHILE;}




  (*** Inform any sectors remaining in queueOld that this plane is no longer             in
its airspace. and remove the sector from the plane's SectorQueue.
                                                                                   ***)

  secA := ASK queueOld First();
  WHILE ( secA <> NILOBJ )
       ASK secA TO RemoveThis ( SELF );
     ASK SectorQueue TO RemoveThis ( secA);
       secA := ASK queueOld Next(secA);
  END WHILE;

  (*** Inform any sectors remaining in queueNew that this plane has entered
       its airspace, and add the sector to the plane's SectorQueue.  ***)

  secB := ASK queueNew First();
 WHILE ( secB <> NILOBJ)
       ASK secB TO Add ( SELF );
       ASK SectorQueue TO Add ( secB );
       secB := ASK queueNew Next(secB);
 END WHILE;

END METHOD;



       ASK METHOD CHECK;
       VAR i,j:INTEGER;
     absent: PlaneInSectorStatus;
       BEGIN
           absent := ABSENT;
      IF ( ABS(XPos-XDest)<ABS(XComp*velocity*0.01 )) AND
                  ( ABS(YPos-YDest)<ABS(YComp*velocity*0.01))

         XPos := XDest;
              YPos := YDest;
              location:= "AT DESTINATION";
              ETA := SimTime();
              {OUTPUT("Erasing polygon . . .");}
              ASK PlaneGraphic TO Erase();
              ASK PlaneIcon TO Erase();
              ASK EnvelopePolygon TO Erase();
              DISPOSE(PlaneIcon);
              DISPOSE(EnvelopePolygon);
              DISPOSE(PlaneGraphic);
              FOR j := 1 TO numofsecy
                 FOR i := 1 TO numofsecx
                      IF ( SectorArray [ i , j ] <> ABSENT )
                              ASK MasterSectorArray [i,j] TO
                              ChangeSectorStatus ( SELF , absent );
                              END IF;
                 END FOR ;
              END FOR ;


        ELSE
              ETA := SimTime() +
                      DIST2D(XPos,YPos,XDest,YDest)/velocity;
        END IF;
      IF ( (TInc < 2.0) OR ( INT(SimTime()) MOD 10 = 0 ) )
        OUTPUT("PLANE ",objectID," is ",location," at time ",SimTime() );
        IF(location="IN FLIGHT") OUTPUT("     ETA = ",ETA);
              END IF;
        END IF;
   END METHOD;


ASK METHOD UpdatePosition;
BEGIN

       (**** Change the position ****)
        XPos := XPos + (SimTime() - TLast)*XComp*velocity ;
        YPos := YPos + (SimTime() - TLast)*YComp*velocity ;

       (**** Update TLast ****)
       TLast := SimTime() ;

       (*** Update the envelope ***)
       ASK SELF TO UpdateEnvelope;

       (*** Update the sectors.  ***)
       ASK SELF TO UpdateSectors;

       (*** Update the envelope sectors ***)
       ASK SELF TO UpdateEnvSectors;

       (*** Check to see if the plane is at its destination ***)
       ASK SELF TO CHECK;


END METHOD;

  (***** Method for the planes to talk ******)

 TELL METHOD Talk(IN Plane1, Plane2:MovingObj);
  VAR
   vel1,vel2: REAL;

  BEGIN
       OUTPUT(" Plane ", ASK Plane1 objectID," has velocity of = ", ASK Plane1
velocity);
       OUTPUT(" Plane ", ASK Plane2 objectID, "has velocity of = ", ASK Plane2
velocity);
       vel1 := ASK Plane1 velocity;
       vel2 := ASK Plane2 velocity;
```

```
        IF (vel1 <= vel2)
{       vel1 := vel1 + ABS(vel1-vel2);
        velocity:= vel1;
        OUTPUT("After the change the new velocity is equal = ", ASK Plane1 velocity);
}

        TELL Plane2 TO Wait;
        END IF;
END METHOD;


TELL METHOD Wait;

 BEGIN
   WAIT DURATION 3.0
  END WAIT;
        {velocity:= ABS(velocity -0.3);}
        OUTPUT("from Wait the object waiting is  ", ASK SELF objectID, "with a
velocity equal to = ", ASK SELF velocity);
END METHOD;

 TELL METHOD Fly;

        VAR
      abc:MovingObj;
        Hypotenuse: REAL;
 BEGIN


        {(*** Initiate the envelope ***)
        ASK SELF TO UpdateEnvelope;

        (*** Initiate the sectors.  ***)
        ASK SELF TO UpdateEnvSectors;}

        location:= "IN FLIGHT";
        ETA := SimTime() +
                MINOF(4.0, DIST2D(XStart,YStart,XDest,YDest)/velocity);
        WHILE ( location <> "AT DESTINATION" )
                (*** Calculate an appropriate time increment ***)
                TInc := MINOF(ABS(ASK ranNumGen Sample() - 0.5),MAXOF(
                        (ETA-SimTime())/2.0,0.01));
                OUTPUT("Time Increment = ", TInc);
                WAIT DURATION TInc
              END WAIT;
              ASK SELF TO UpdatePosition;

        END WHILE;
         END METHOD;
 END OBJECT;

OBJECT ControllerObj;

        TELL METHOD Inform(IN Plane1,Plane2: MovingObj);

        BEGIN

        OUTPUT( "From Control Plane ", ASK Plane1 objectID,
        "From Control Plane", ASK Plane2 objectID);
        TELL Plane2 TO ReduceSpeed;

  END METHOD;
        ASK METHOD SetControllerID(IN I:INTEGER);

        BEGIN
        ConObjID:= I;

        END METHOD;

{       ASK METHOD ConPlanes(IN Con:ANYOBJ);

        VAR
        ETA: REAL;
        ConobjID: INTEGER;
        ConXpos: REAL;
        {ConYpos,
        ConXDes,
        ConYDes : REAL;}
        CAirplane: MovingObj;
        def: MovingObj;
        BEGIN
        CAirplane:= Con;

        IF ASK CAirplane objectID = 2
        ConobjID:= ASK CAirplane objectID;
        ConXpos:= ASK CAirplane XPos;
        OUTPUT ("FROM CONTROL OBJECT OBJECT ID AND POSITION",
 ConobjID, ConXpos);
        END IF
```

```
    def := ASK planenames First();
            WHILE def <> NILOBJ
            {OUTPUT(" ***From Controller Name in QUeue ",ASK def  objectID,
                 "  ",  ASK def XPos);}

            def:= ASK planenames Next(def);
            END WHILE;

      END METHOD;}

END OBJECT;

OBJECT GeneratorObj;

  TELL METHOD GenSectors;
  VAR
    i,j : INTEGER;
    sector : SectorObj;
  BEGIN

    FOR j:= 1 TO numofsecy;
       FOR i:=1 TO numofsecx ;
             NEW ( sector );
             ASK sector TO SetobjectID(i,j);
             ASK sector TO SetBorders;
             ASK (ASK sector Polygon) TO Draw();
             MasterSectorArray[i,j] := sector;
             TELL sector TO MonitorSector(Controller);
       END FOR;
    END FOR;
  END METHOD;

  ASK METHOD GenController;
   VAR

    i: INTEGER;
       BEGIN
       i:=1;
        NEW(Controller);
       OUTPUT("From the control generator, I got created");
         ASK Controller TO SetControllerID(i);
       END METHOD;

  TELL METHOD GenPlanes;
  VAR
  def := ASK planenames First();
          WHILE def <> NILOBJ
          {OUTPUT(" ***From Controller Name in QUeue ",ASK def  objectID,
                 " ",  ASK def XPos);}

          def:= ASK planenames Next(def);
          END WHILE;

     END METHOD;}

END OBJECT;

OBJECT GeneratorObj;

  TELL METHOD GenSectors;
  VAR
    i,j : INTEGER;
    sector : SectorObj;
  BEGIN

    FOR j:= 1 TO numofsecy;
       FOR i:=1 TO numofsecx ;
             NEW ( sector );
             ASK sector TO SetobjectID(i,j);
             ASK sector TO SetBorders;
             ASK (ASK sector Polygon) TO Draw();
             MasterSectorArray[i,j] := sector;
             TELL sector TO MonitorSector(Controller);
       END FOR;
    END FOR;
  END METHOD;

  ASK METHOD GenController;
   VAR

    i: INTEGER;
       BEGIN
       i:=1;
        NEW(Controller);
       OUTPUT("From the control generator, I got created");
         ASK Controller TO SetControllerID(i);
       END METHOD;

  TELL METHOD GenPlanes;
  VAR
```

```
 ranNumGen: RandomObj;
Airplane : MovingObj;

     waittime:REAL;
 k : INTEGER;
     i,j: INTEGER;
     junk:MovingObj;

 BEGIN
     NEW(planenames);
     NEW(ranNumGen);
     numofplanes:= 0;

 FOR k:=1 TO 5;
   NEW(Airplane);
      INC(numofplanes);
      ASK planenames TO Add(Airplane);
   ASK  Airplane TO SetobjectID(numofplanes);
      ASK Airplane TO InitEnvelope;
      ASK Airplane TO InitSectorQueue;

      (*** Define the flight path of the plane with random values. ***)

      ASK Airplane TO
               SetOrigination ( (xrange-5.0) *
                                     ( (ASK ranNumGen Sample()) - 0.5) ,
                          (yrange-5.0) *
                                     ( (ASK ranNumGen Sample()) - 0.5) ) ;
      ASK Airplane TO
               SetSpeed ( 5.0 *(ASK ranNumGen Exponential(meanarrivaltime)) ) ;
      ASK Airplane TO
               SetCourse (  xrange * ((ASK ranNumGen Sample()) - 0.5) ,
                          yrange * ((ASK ranNumGen Sample()) - 0.5) ) ;

      ASK Airplane TO SetLocation ( "ON GROUND" ) ;

      (*** Initiate the envelope ***)
      ASK Airplane TO UpdateEnvelope;

      (*** Initiate the sectors.  ***)
      ASK Airplane TO UpdateEnvSectors;


      TELL Airplane TO Fly;
```

```
    END FOR;


  WHILE SimTime() < (numofhrs * 60.0)
 WAIT DURATION 5.0;
      END WAIT;
      END WHILE;

      {INTERRUPT ALL(MovingObj);}

DISPOSE (ranNumGen);
      OUTPUT("Ready . . .");
      FOR j := 1 TO numofsecy
      FOR i := 1 TO numofsecx
        junk := ASK MasterSectorArray[i,j] First();
       IF ( junk <> NILOBJ)
             OUTPUT;
             OUTPUT("Sector ",i,",",j," contains ");
             WHILE (junk<>NILOBJ)
               OUTPUT("Plane ",ASK junk objectID);
               junk := ASK MasterSectorArray[i,j] Next(junk);
               END WHILE;
             END IF;
       END FOR;
       END FOR;

END METHOD;
END OBJECT;
END MODULE.
```

```
DEFINITION MODULE AuxMathMod;

FROM MathMod IMPORT SQRT;

PROCEDURE DIST2D ( IN x1, y1, x2, y2:  REAL) : REAL;
PROCEDURE INT ( IN x: REAL) : INTEGER;

END MODULE.
```

```
IMPLEMENTATION MODULE AuxMathMod;

FROM MathMod IMPORT SQRT;

PROCEDURE DIST2D ( IN x1, y1, x2, y2:  REAL) : REAL;
        BEGIN
          RETURN( SQRT( (x2-x1)*(x2-x1) + (y2-y1)*(y2-y1) ) );
        END PROCEDURE;

PROCEDURE INT ( IN x: REAL): INTEGER;
        BEGIN
         RETURN ( ROUND (x-0.5) );
        END PROCEDURE;

END MODULE.
```

```
DEFINITION MODULE CCGeoObj;

TYPE
        GeoObj = OBJECT
        NameID : STRING;
        XPos,YPos,YDest,XDest,XComp,YComp :REAL;
        ID    :  INTEGER;
        Status  :   STRING;
        ASK METHOD SetNameID(IN LocalNameID : STRING);
        ASK METHOD SetID(IN localObjNumber : INTEGER);
        TELL METHOD NewArrive1(IN Indicator:STRING);


END OBJECT;

PROCEDURE ReportEstimatedTime;

VAR


        Geo : GeoObj;


END MODULE.
```

```
IMPLEMENTATION MODULE CCGeoObj;

OBJECT GeoObj;

  ASK METHOD SetNameID (IN LocalNameID : STRING);

   BEGIN

         NameID := LocalNameID;

  END METHOD;

  ASK METHOD SetID(IN localObjNumber : INTEGER);

   BEGIN
         ID := localObjNumber;

    END METHOD;

   TELL METHOD NewArrive1(IN Indicator :STRING);

    BEGIN

     END METHOD;


  END OBJECT;

  END MODULE.
```

```
DEFINITION MODULE CCGlobal;
{FROM ACMove IMPORT ModelInit,PlannerObj,PlaneObj;}
{FROM CCSectorObj IMPORT SectorObj;}
FROM CSectQObj IMPORT SectorQueueObj;
FROM GrpMod IMPORT QueueObj;
FROM RandMod IMPORT RandomObj, FetchSeed, Random;
FROM AuxMathMod IMPORT DIST2D, INT;
FROM GTypes IMPORT PointArrayType;
FROM SimMod IMPORT StartSimulation, SimTime;
{FROM CCSpatialTempObj IMPORT SpatialTempletObj,SpatialTemplet;
FROM CCStationaryObj IMPORT StationaryObj, Stationary;
FROM CConflictIdentObj IMPORT ConflictIdentifierObj,ConflictIdentifier;
FROM CConflictResolve  IMPORT ConflictResolverObj,ConflictResolver;}

FROM Image IMPORT ImageObj;
FROM Fill IMPORT PolygonObj;
FROM Window IMPORT WindowObj;
FROM Animate IMPORT DynDClockObj;
FROM Graphic IMPORT GraphicLibObj;
TYPE


 GeneratorObj = OBJECT

    ASK METHOD GenSpatialTemplet;

     ASK METHOD  GenConflictIdent;
    ASK METHOD GenConflictResolve;

    ASK METHOD GenStationary;


  END OBJECT;




  VAR
   {simuluation parameters}
   numofhrs,
   meanarrivaltime: REAL;
   numofplanes:INTEGER;

   SpatialTempletGenerator,
ConflictIdentifierGenerator,
ConflictResolverGenrator,
StationaryGenerator          :GeneratorObj;

planenames: QueueObj;

{fields necessary for generating planes}
ranNumGen: RandomObj;

{parameters for defining the sectors by the spatial templet}
xrange, yrange: REAL;

numofsecx,
numofsecy: INTEGER;


ObjectNumber : INTEGER;


Pi:REAL;
{Graphics variables }
MasterGraphic: ImageObj;
window: WindowObj;
clockwindow:  WindowObj;
clockgraphic: DynDClockObj;
MasterPlaneIcon: ImageObj;
GraphicLib : GraphicLibObj;
Polygon: PolygonObj;

PlaneIcon : ImageObj;
PlaneGraphic:ImageObj;



END MODULE.
```

```
IMPLEMENTATION MODULE CCGlobal;
FROM CCGeoObj IMPORT GeoObj,Geo;
FROM ACMove IMPORT PlannerObj,PlaneObj,MovPlanner;
FROM CCSectorObj IMPORT SectorObj;
FROM CSectQObj IMPORT SectorQueueObj;
FROM GrpMod IMPORT QueueObj;
FROM RandMod IMPORT RandomObj, FetchSeed, Random;
FROM AuxMathMod IMPORT DIST2D, INT;
FROM GTypes IMPORT PointArrayType;
FROM SimMod IMPORT StartSimulation, SimTime;
FROM CCSpatialTempObj IMPORT SpatialTempletObj,SpatialTemplet;
FROM CCStationaryObj IMPORT StationaryObj,Stationary;
FROM CConflictIdentObj IMPORT ConflictIdentifierObj,ConflictIdentifier;
FROM CConflictResolve  IMPORT ConflictResolverObj,ConflictResolver;
FROM Image IMPORT ImageObj;
FROM Fill IMPORT PolygonObj;
FROM Window IMPORT WindowObj;
FROM Animate IMPORT DynDClockObj;
FROM Graphic IMPORT GraphicLibObj;



 OBJECT GeneratorObj;


   ASK METHOD GenSpatialTemplet;
    BEGIN
          NEW(SpatialTemplet);

       {create and set the Bvertex attribute of the Spatial Templet}
          ASK SpatialTemplet TO SetFrame(xrange,yrange);

          ASK SpatialTemplet TO GenSectors;

    END METHOD;

   ASK METHOD GenConflictIdent;

          BEGIN
           NEW (ConflictIdentifier );



    END METHOD;
```

```
ASK METHOD GenConflictResolve;
      BEGIN
        NEW(ConflictResolver);
      END METHOD;



ASK METHOD GenStationary;

      BEGIN
         INC(ObjectNumber);

              NEW(Stationary);
              ASK Stationary TO InitStationaryFields("Static");

      END METHOD;


END OBJECT;
END MODULE.
```

```
DEFINITION MODULE CConflictIdentObj;

            (***** Conflict Indentifier Object *****)

FROM CCGeoObj IMPORT GeoObj;
FROM RandMod IMPORT RandomObj, FetchSeed, Random;
FROM GrpMod IMPORT QueueObj;
FROM GTypes IMPORT PointArrayType;

FROM AuxMathMod IMPORT DIST2D, INT;
FROM ACMove IMPORT ModelInit,PlannerObj,PlaneObj;

FROM CCSpatialTempObj IMPORT SpatialTempletObj;
FROM CCSectorObj IMPORT SectorObj;

TYPE



ConflictIdentifierObj = OBJECT

ASK METHOD ReportPC(IN conflictsector:SectorObj; IN Member,Rock:GeoObj;
INOUT Indicator:STRING);
ASK  METHOD Noconflict(IN plane1:PlaneObj);
{ASK METHOD Doit(IN plane1:PlaneObj; INOUT Indicator:STRING);}

END OBJECT;{ConflictIdentifierObj}

VAR
ConflictIdentifier: ConflictIdentifierObj;
DesVar1, DesVar2: REAL;
END MODULE.
```

```
IMPLEMENTATION MODULE CConflictIdentObj;

FROM CCGeoObj IMPORT GeoObj;
FROM ACMove IMPORT PlannerObj,PlaneObj,ReportEstimatedTime;
FROM CCStationaryObj IMPORT StationaryObj;
FROM CCSectorObj IMPORT SectorObj;
FROM SimMod IMPORT PendingListDump;
FROM CCSpatialTempObj IMPORT SpatialTempletObj,SpatialTemplet;
FROM CConflictResolve IMPORT ConflictResolverObj, ConflictResolver;
OBJECT ConflictIdentifierObj;

ASK METHOD ReportPC
      (IN conflictsector:SectorObj;
       IN Member,Rock:GeoObj;
       INOUT Indicator:STRING);


VAR
sxpos1,sypos1:REAL;
sxpos2,sypos2:REAL;
estime2,estime1:REAL;
DesVar1, DesVar2: REAL;
p1ID,p2ID: INTEGER;

         xposp1,
         yposp1,
       xdestp1,
         ydestp1: REAL;

       xposp2,
         yposp2,
       xdestp2,
         ydestp2: REAL;

       sectxmin,
       sectxmax,
       sectymin,
       sectymax :REAL;

       sectidx,
       sectidy: INTEGER;
          m1,m2, b1,b2: REAL;
          yint01,yint11: REAL;
          xint01,xint11: REAL;
          yint02,yint12: REAL;
      xint02, xint12:REAL;
      xxx1,yyy1: REAL;
      xxx2,yyy2:REAL;
      localnameID1,localnameID2:STRING;

      planestatus1,planestatus2: STRING;

      Diff: REAL;
      Plane:PlaneObj;
      Stationary:StationaryObj;
      plane1,plane2: GeoObj;


BEGIN

      localnameID1 := ASK Member NameID;
      localnameID2 := ASK Rock NameID;

      IF (localnameID1 = "Aircraft") AND (localnameID2 = "Aircraft")


      plane1 := Member;
      plane2 := Rock;


      p1ID:= ASK plane1 ID;
     p2ID:= ASK plane2 ID;

      xposp1:= ASK plane1 XPos;
      yposp1:= ASK plane1 YPos;
     xdestp1:= ASK plane1 XDest;
      ydestp1:= ASK plane1 YDest;

     xposp2:= ASK plane2 XPos;
      yposp2:= ASK plane2 YPos;
     xdestp2:= ASK plane2 XDest;
      ydestp2:= ASK plane2 YDest;


       sectxmin:= ASK conflictsector xmin;
       sectxmax:= ASK conflictsector xmax;
       sectymin:= ASK conflictsector ymin;
       sectymax:= ASK conflictsector ymax;

      (**************** Determining the Intersection ********)
```

```
(*** FOR PLANE # 1 ***)

m1:= (ydestp1 -yposp1)/ (xdestp1 -xposp1);
b1:= yposp1 - (m1* xposp1);


yint01 := (m1* sectxmin) - b1;


yint11 := (m1 *sectxmax) -b1;


xint01 := (sectymin - b1)/m1;

xint11:= (sectymax - b1)/m1;

xxx1:= ASK plane1 XComp;
yyy1:= ASK plane1 YComp;


IF((xxx1 > 0.0) AND (xint11 > xint01)) OR
  (( yyy1 > 0.0) AND (yint11 > yint01))

        sxpos1:= xint01;
        sypos1:= yint01;
ELSE

        sxpos1:= xint11;
        sypos1:= yint11;

END IF;


(*** FOR PLANE # 2 ***)

m2:= (ydestp2 -yposp2)/ (xdestp2 -xposp2);
b2:= yposp1 - (m2* xposp2);


yint02 := (m2* sectxmin) - b2;


yint12 := (m2 *sectxmax) -b2;
```

```
xint02 := (sectymin - b2)/m2;

xint12:= (sectymax - b2)/m2;



xxx2:= ASK plane2 XComp;
yyy2:= ASK plane2 YComp;

IF((xxx2 > 0.0) AND (xint12 > xint02)) OR
  (( yyy2 > 0.0) AND (yint12 > yint02))
        {OUTPUT(" point 1 is before point2");}
        sxpos2:= xint02;
        sypos2:= yint02;
 ELSE
        {OUTPUT(" point 2 is before point 1");}
        sxpos2:= xint12;
        sypos2:= yint12;

 END IF;


 {**** Enquire from the planes their estimated time
   of entering the sector***}


  ReportEstimatedTime(p1ID,sxpos1,sypos1,estime1);
DesVar1:= estime1;
ReportEstimatedTime(p2ID,sxpos2,sypos2,estime2);
  DesVar2:=estime2;

(***New section done on Sat 2/8/92 *****)

{Included for the Resolver}


  IF (ABS(DesVar1 - DesVar2) < 5.0)

   {conflict exists call conflict resolver}

  ASK ConflictResolver TO
       Resolve(plane1,plane2,Indicator,DesVar1,DesVar2);
END IF;
```

```
        ELSIF (localnameID1 = "Aircraft") AND (localnameID2 = "Static")

           OUTPUT("testing resolve 1; plane 1st");

        Indicator :="ON STATIC";



        ELSIF  (localnameID1 = "Static") AND (localnameID2 = "Aircraft")

           OUTPUT("testing resolve 1; plane 2nd");
        Indicator :="ON STATIC";

         END IF;

  END METHOD;



  ASK METHOD Noconflict(IN plane1:PlaneObj);
  VAR
  planestatus: STRING;
  BEGIN
   planestatus:= ASK plane1 Status;
      OUTPUT("CALLING NOCOFLICT--  ERROR");
          IF (planestatus <> "GIVEN PERMISSION")
          {TELL plane1 NewMove(plane1);}

            END IF;

   END METHOD;

   END OBJECT;
   END MODULE.
```

```
IMPLEMENTATION MODULE CConflictIdentObj;

FROM CCGeoObj IMPORT GeoObj;
FROM ACMove IMPORT PlannerObj,PlaneObj,ReportEstimatedTime;
FROM CCStationaryObj IMPORT StationaryObj;
FROM CCSectorObj IMPORT SectorObj;
FROM SimMod IMPORT PendingListDump;
FROM CCSpatialTempObj IMPORT SpatialTempletObj,SpatialTemplet;
FROM CConflictResolve IMPORT ConflictResolverObj, ConflictResolver;
OBJECT ConflictIdentifierObj;

ASK METHOD ReportPC
      (IN conflictsector:SectorObj;
       IN Member,Rock:GeoObj;
       INOUT Indicator:STRING);


VAR
sxpos1,sypos1:REAL;
sxpos2,sypos2:REAL;
estime2,estime1:REAL;
DesVar1, DesVar2: REAL;
p1ID,p2ID: INTEGER;

        xposp1,
        yposp1,
      xdestp1,
        ydestp1: REAL;

      xposp2,
        yposp2,
      xdestp2,
        ydestp2: REAL;

      sectxmin,
      sectxmax,
      sectymin,
      sectymax :REAL;

      sectidx,
      sectidy: INTEGER;
        m1,m2, b1,b2: REAL;
        yint01,yint11: REAL;
        xint01,xint11: REAL;
        yint02,yint12: REAL;
        xint02, xint12:REAL;
        xxx1,yyy1: REAL;
        xxx2,yyy2:REAL;
        localnameID1,localnameID2:STRING;

        planestatus1,planestatus2: STRING;

        Diff: REAL;
        Plane:PlaneObj;
        Stationary:StationaryObj;
        plane1,plane2: GeoObj;


BEGIN

        localnameID1 := ASK Member NameID;
        localnameID2 := ASK Rock NameID;

        IF (localnameID1 = "Aircraft") AND (localnameID2 = "Aircraft")


        plane1 := Member;
        plane2 := Rock;


        p1ID:= ASK plane1 ID;
      p2ID:= ASK plane2 ID;

        xposp1:= ASK plane1 XPos;
        yposp1:= ASK plane1 YPos;
      xdestp1:= ASK plane1 XDest;
        ydestp1:= ASK plane1 YDest;

      xposp2:= ASK plane2 XPos;
        yposp2:= ASK plane2 YPos;
      xdestp2:= ASK plane2 XDest;
        ydestp2:= ASK plane2 YDest;


         sectxmin:= ASK conflictsector xmin;
         sectxmax:= ASK conflictsector xmax;
         sectymin:= ASK conflictsector ymin;
         sectymax:= ASK conflictsector ymax;

       (**************** Determining the Intersection ********)
```

```
(*** FOR PLANE # 1 ***)

m1:= (ydestp1 -yposp1)/ (xdestp1 -xposp1);
b1:= yposp1 - (m1* xposp1);


yint01 := (m1* sectxmin) - b1;


yint11 := (m1 *sectxmax) -b1;


xint01 := (sectymin - b1)/m1;

xint11:= (sectymax - b1)/m1;

xxx1:= ASK plane1 XComp;
yyy1:= ASK plane1 YComp;


IF((xxx1 > 0.0) AND (xint11 > xint01)) OR
  (( yyy1 > 0.0) AND (yint11 > yint01))

        sxpos1:= xint01;
        sypos1:= yint01;
ELSE

        sxpos1:= xint11;
        sypos1:= yint11;

END IF;


(*** FOR PLANE # 2 ***)

m2:= (ydestp2 -yposp2)/ (xdestp2 -xposp2);
b2:= yposp1 - (m2* xposp2);


yint02 := (m2* sectxmin) - b2;


yint12 := (m2 *sectxmax) -b2;
```

```
xint02 := (sectymin - b2)/m2;

xint12:= (sectymax - b2)/m2;



xxx2:= ASK plane2 XComp;
yyy2:= ASK plane2 YComp;

IF((xxx2 > 0.0) AND (xint12 > xint02)) OR
  (( yyy2 > 0.0) AND (yint12 > yint02))
        {OUTPUT(" point 1 is before point2");}
        sxpos2:= xint02;
        sypos2:= yint02;
ELSE
        {OUTPUT(" point 2 is before point 1");}
        sxpos2:= xint12;
        sypos2:= yint12;

END IF;


{**** Enquire from the planes their estimated time
  of entering the sector***}


 ReportEstimatedTime(p1ID,sxpos1,sypos1,estime1);
DesVar1:= estime1;
ReportEstimatedTime(p2ID,sxpos2,sypos2,estime2);
 DesVar2:=estime2;

(***New section done on Sat 2/8/92 *****)

{Included for the Resolver}


  IF (ABS(DesVar1 - DesVar2) < 5.0)

  {conflict exists call conflict resolver}

  ASK ConflictResolver TO
       Resolve(plane1,plane2,Indicator,DesVar1,DesVar2);
END IF;
```

```
      ELSIF (localnameID1 = "Aircraft") AND (localnameID2 = "Static")

         OUTPUT("testing resolve 1; plane 1st");

      Indicator :="ON STATIC";



      ELSIF  (localnameID1 = "Static") AND (localnameID2 = "Aircraft")

         OUTPUT("testing resolve 1; plane 2nd");
      Indicator :="ON STATIC";

      END IF;

END METHOD;



ASK METHOD Noconflict(IN plane1:PlaneObj);
VAR
planestatus: STRING;
BEGIN
 planestatus:= ASK plane1 Status;
    OUTPUT("CALLING NOCOFLICT--  ERROR");
       IF (planestatus <> "GIVEN PERMISSION")
       {TELL plane1 NewMove(plane1);}

        END IF;

END METHOD;

END OBJECT;
END MODULE.
```

```
DEFINITION MODULE CCSectorObj;
FROM GTypes IMPORT PointArrayType;
FROM GrpMod IMPORT QueueObj;
FROM ACMove IMPORT ModelInit,PlannerObj,PlaneObj;
FROM CCStationaryObj IMPORT StationaryObj,Stationary;
FROM CCGeoObj IMPORT GeoObj;
FROM SimMod IMPORT StartSimulation, SimTime;
FROM AuxMathMod IMPORT DIST2D, INT;
FROM CSectQObj IMPORT SectorQueueObj;
FROM Fill IMPORT PolygonObj;
FROM GTypes IMPORT ColorType(Green,Yellow,Blue,Red,Orange,Violet,White);
FROM GTypes IMPORT FillStyleType(HollowFill,SolidFill,NarrowDiagonalFill,
        MediumDiagonalFill,WideDiagonalFill,NarrowCrosshatchFill,
        MediumCrosshatchFill,WideCrosshatchFill);




TYPE


SectorManagerObj =OBJECT

        SectorQueue:  SectorQueueObj;
        SectorEnvQueue: SectorQueueObj;
        SectorArray:  ARRAY INTEGER , INTEGER, INTEGER OF STRING;
        vertex: PointArrayType ;
        EnvelopePolygon: PolygonObj;
        Polygon : PolygonObj;

        ASK METHOD InitEnvelope; {called by a sector manager}
        ASK METHOD InitSectorQueue; {called by a sector manager}


        ASK METHOD UpdateEnvelope(IN Plane:PlaneObj);

        ASK METHOD checkupdate(IN Plane: PlaneObj; IN TempInd: STRING);
        ASK METHOD UpdateEnvSectors(IN Plane:PlaneObj);
        ASK METHOD SetStationaryEnvSectors(IN Stationary:StationaryObj);
        ASK METHOD UpdateSectors(IN Plane:PlaneObj);
        ASK METHOD DumpSectors;
        ASK METHOD DumpStationarySectors;
```

```
END OBJECT;


{####Rajesh elimate this inheritance add vertex to this object first
  check for other things}

SectorObj = OBJECT(SectorManagerObj,QueueObj)

        sectorIDx , sectorIDy : INTEGER;
        xmin,ymin,xmax,ymax : REAL;

      ASK METHOD SetobjectID(IN i,j:INTEGER);
        ASK METHOD SetBorders(IN xrange,yrange:REAL; IN
numofsecx,numofsecy:INTEGER);
      ASK METHOD ChangeSectorStatusAbsent (IN Plane : PlaneObj);
        ASK METHOD ChangeSectorStatus ( IN Member : GeoObj ;
                               IN Status : STRING;
                             INOUT Indicator : STRING ) ;
      ASK METHOD LogicallyRemove ( IN  Plane: PlaneObj );
        ASK METHOD MonitorSector(IN Member:GeoObj; INOUT Indicator :
STRING);
END OBJECT;



VAR

SectorManager: SectorManagerObj;  {I believe this is here by error as only 1
                          manager is created and it's reference
                          pointer is a global variable to the
                          DCCSpatialTempObj.mod and its routines}
SectorArray: ARRAY INTEGER, INTEGER OF STRING;

END MODULE.
```

```
IMPLEMENTATION MODULE CCSectorObj;

FROM GTypes IMPORT PointArrayType;
FROM GrpMod IMPORT QueueObj;
FROM ACMove IMPORT ModelInit,PlannerObj,PlaneObj;
FROM CCStationaryObj IMPORT StationaryObj,Stationary;
FROM CCGeoObj IMPORT GeoObj;
FROM SimMod IMPORT StartSimulation, SimTime;
FROM MathMod IMPORT SQRT,ASIN,POWER, SIN, COS,ATAN;
FROM CCGlobal IMPORT xrange, yrange, numofsecx, numofsecy,numofhrs,Pi;
FROM CSectQObj IMPORT SectorQueueObj;
FROM AuxMathMod IMPORT DIST2D, INT;
FROM CCSpatialTempObj IMPORT  MasterSectorArray,
                     SpatialTempletObj,SpatialTemplet;
FROM Fill IMPORT PolygonObj;
FROM Image IMPORT ImageObj;
FROM GTypes IMPORT ColorType(Green,Yellow,Blue,Red,Orange,Violet,White);
FROM GTypes IMPORT FillStyleType(HollowFill,SolidFill,NarrowDiagonalFill,
        MediumDiagonalFill,WideDiagonalFill,NarrowCrosshatchFill,
        MediumCrosshatchFill,WideCrosshatchFill);
FROM Line IMPORT PolylineObj;
FROM GTypes IMPORT LineStyleType(SolidLine);
FROM CCGlobal IMPORT GraphicLib;
FROM CCGlobal IMPORT MasterGraphic, MasterPlaneIcon,window, PlaneIcon,
PlaneGraphic;
FROM CConflictIdentObj IMPORT ConflictIdentifierObj,ConflictIdentifier;
FROM CConflictResolve  IMPORT ConflictResolverObj,ConflictResolver;


OBJECT SectorObj;

ASK METHOD SetobjectID (IN i,j: INTEGER);

  BEGIN
        sectorIDx := i;
        sectorIDy := j;
END METHOD;


ASK METHOD SetBorders(IN localxrange,localyrange:REAL;
             IN localnumofsecx,localnumofsecy: INTEGER);


  BEGIN
```

```
    xmin := localxrange * ( FLOAT(sectorIDx-1) /
                     FLOAT(localnumofsecx) - 0.5 ) ;
    xmax := localxrange * ( FLOAT(sectorIDx) /
                    FLOAT(localnumofsecx) - 0.5 ) ;
    ymin := localyrange * ( FLOAT(sectorIDy-1) /
                    FLOAT(localnumofsecy) - 0.5 ) ;
    ymax := localyrange * ( FLOAT(sectorIDy) /
                    FLOAT(localnumofsecy) - 0.5 ) ;
    NEW(vertex, 1..4);
    vertex[1].x := xmin ; vertex[1].y := ymin ;
    vertex[2].x := xmax ; vertex[2].y := ymin ;
    vertex[3].x := xmax ; vertex[3].y := ymax ;
    vertex[4].x := xmin ; vertex[4].y := ymax ;
    NEW(Polygon);
    ASK Polygon TO SetPoints(vertex);
    ASK Polygon TO SetColor(Yellow);
    ASK Polygon TO SetStyle(HollowFill);
    ASK MasterGraphic TO AddGraphic(Polygon);


END METHOD;


ASK METHOD ChangeSectorStatusAbsent (IN Plane : PlaneObj);
VAR
 elem : PlaneObj;     {****is this method being used???****}
 BEGIN
             IF (ASK SELF Includes(Plane))

              ASK SELF RemoveThis(Plane);
              OUTPUT("Remove Plane ", ASK Plane ID);
{Added this in order to change the graphical sector status, Rajesh,3/3/92}
{***************}
       ASK Polygon TO SetColor(Yellow);
       ASK Polygon TO SetStyle(HollowFill);
     ASK Polygon TO Draw() ;


              ELSE
               OUTPUT("ERROR, In Remove plane But not a memeber of master
 sector");

              END IF;
END METHOD;

ASK METHOD ChangeSectorStatus ( IN Member : GeoObj ;
        IN Status : STRING; INOUT Indicator : STRING ) ;
  VAR
        localname: STRING;

  {VAR
        elem,elem1, : PlaneObj ;}

  BEGIN
  {the Status defines whether the plane should be in this sector obj's queue
     or not}

  IF ( Status = "ABSENT" )

    (**plane is in the permanent sector array and must be removed **)
 {I am changing this in order to accomodate for the static}
 {object on 04/07/92 ,Rajesh}
    {localmember := ASK SELF First();}



    IF (ASK SELF Includes(Member))
         localname := ASK Member NameID;
        IF (localname <> "Static")
           ASK SELF RemoveThis(Member);


     {if this sector is now empty--change the grapic color back to empty}
  {Added this in order to change the graphical sector status, Rajesh,3/3/92}
  {****************}
     IF ASK SELF numberIn = 0
         ASK Polygon TO SetColor(Yellow);
         ASK Polygon TO SetStyle(HollowFill);
       ASK Polygon TO Draw() ;
     END IF;
     END IF;

     {OUTPUT("From change sector status Remove Plane ", ASK Member ID);}

   ELSE
    OUTPUT("ERROR, In Remove plane But not a memeber of master sector");
 END IF;

ELSIF ( Status = "PRESENT" )
  ASK Polygon TO SetColor(Red) ;
  ASK Polygon TO SetStyle(SolidFill) ;
  ASK Polygon TO Draw() ;

  {?????}
  IF NOT (ASK SELF Includes(Member))

       ASK SELF TO Add(Member) ;
   END IF;

  ASK SELF MonitorSector(Member,Indicator);

 ELSIF ( Status = "ENVELOPE" )


   IF NOT (ASK SELF Includes(Member))

       ASK SELF TO Add(Member) ;
   END IF;

{added this IF statement to incorporate the Static}
    IF ASK SELF numberIn = 1

                ASK Polygon TO SetColor(Red) ;
                ASK Polygon TO SetStyle(NarrowDiagonalFill) ;
                ASK Polygon TO Draw() ;

        END IF;


    ASK SELF MonitorSector(Member,Indicator);

  ELSIF ( Status = "StationaryEnvelope" )

    ASK Polygon TO SetColor(Green) ;
    ASK Polygon TO SetStyle(SolidFill) ;
    ASK Polygon TO Draw() ;

    IF NOT (ASK SELF Includes(Member))

        ASK SELF TO Add(Member) ;
        ASK SELF MonitorSector(Member,Indicator);


     END IF;


    END IF;


   END METHOD;

   ASK METHOD LogicallyRemove ( IN  Plane:  PlaneObj );
   VAR
          elem : PlaneObj;
   BEGIN
          elem := ASK SELF First();
          WHILE ( elem <> NILOBJ )
                 IF ( elem = Plane )
              ASK SELF TO  RemoveThis ( elem );
               END IF;
                 elem := ASK SELF Next (elem) ;
          END WHILE;
   END METHOD;

   ASK METHOD MonitorSector(IN Member:GeoObj;INOUT Indicator : STRING);

   VAR

       Rock: GeoObj;

       Plane : PlaneObj;

   BEGIN
```

```
    Rock := ASK SELF First();


      {OUTPUT
     ("MONITOR SECTOR--is plane in sector: ", ASK SELF Includes (Member));}

      WHILE (Rock <> NILOBJ )

         IF  (Rock <> Member)

        ASK SpatialTemplet TO
        InformPotentialConf(SELF,Member,Rock,Indicator);

         END IF;

       Rock := ASK SELF Next(Rock);

     END WHILE;


END METHOD;

END OBJECT;

{***************** OBJECT SectorManagerObj   *********************}

OBJECT SectorManagerObj;

ASK METHOD InitEnvelope;

        VAR
        sx,sy:REAL;

        BEGIN

           NEW ( vertex, 1..4);
           NEW ( PlaneGraphic );
           NEW ( EnvelopePolygon );
           ASK EnvelopePolygon TO SetStyle(HollowFill);
           NEW ( PlaneIcon ) ;
           ASK GraphicLib TO ReadFromFile("AConIcon.lib");
           ASK PlaneIcon TO LoadFromLibrary(GraphicLib,"PlaneIcon");
        ASK PlaneGraphic TO AddGraphic(PlaneIcon);
        ASK PlaneGraphic TO AddGraphic ( EnvelopePolygon ) ;
        ASK MasterGraphic TO AddGraphic ( PlaneGraphic);
        ASK PlaneIcon Scaling(sx,sy);
        ASK PlaneIcon Scale(sx/300.0,sy/300.0);

END METHOD;

ASK METHOD InitSectorQueue;

    VAR
        k ,i , j : INTEGER ;
    BEGIN
      NEW ( SectorQueue );
       NEW( SectorEnvQueue);
      NEW ( SectorArray , 1 .. 10, 1 .. numofsecx , 1 .. numofsecy ) ;
      FOR k := 1 TO 10
      FOR j := 1 TO numofsecy
            FOR i := 1 TO numofsecx
                    SectorArray [ k ,i , j ] := "ABSENT" ;
            END FOR ;
      END FOR ;
      END FOR;
END METHOD;



     ASK METHOD UpdateEnvelope(IN Plane: PlaneObj);
     VAR

      localxpos,localypos,
    localxcomp,localycomp,
    localxdest,localydest,
    localpheading,
    localttoarrive    :REAL;
    {replace the number 2.0 with the variable: HalfWidthOfEnvelope}
      BEGIN



       localxpos := ASK Plane XPos;
       localypos := ASK Plane YPos;
       localxdest := ASK Plane XDest;
       localydest := ASK Plane YDest;
       localxcomp := ASK Plane XComp;
```

```
  localycomp := ASK Plane YComp;
  localpheading := ASK Plane Heading;

 {vertex is a Sector Manager attribute in this case SELF;
  is method can be called by the sector objects also
  through inheritance, but I didn't find any such calls;
  (if the sector calls it then vertex belong to the sector obj)
 IMPORTANT: sets vertex for this plane to be followed
        immediately by a call to Update Env Sectors
  }

 {Rajesh-- save these vertex calculations in the plane's vertex
    and call the plane's vertex in the UPdateEvnSectors method}

    vertex[1].x := localxpos - 2.0 * localycomp ;
    vertex[1].y := localypos + 2.0  * localxcomp ;
    vertex[2].x := localxpos + 2.0 * localycomp ;
    vertex[2].y := localypos - 2.0 * localxcomp ;
    vertex[3].x := localxdest + 2.0 * localycomp ;
    vertex[3].y := localydest - 2.0 * localxcomp ;
    {vertex[4].x := localxdest - 2.0 * localxcomp ;
    vertex[4].y := localydest + 2.0 * localxcomp ;}

IF (localxcomp <= 0.0)AND (localycomp >= 0.0)

      vertex[4].x := localxdest  - 2.0*localycomp ;
     vertex[4].y := localydest  + 2.0*localxcomp ;
  ELSIF (localxcomp >= 0.0)AND (localycomp <= 0.0)

      vertex[4].x := localxdest  - 2.0*localycomp ;
      vertex[4].y := localydest  + 2.0*localxcomp ;
    ELSIF (localxcomp <= 0.0)AND (localycomp <= 0.0)
      vertex[4].x := localxdest   - 2.0*localycomp ;
      vertex[4].y := localydest + 2.0*localxcomp ;
    ELSIF (localxcomp > 0.0) AND (localycomp > 0.0)
      vertex[4].x := localxdest  - 2.0*localycomp ;
      vertex[4].y := localydest  + 2.0*localxcomp ;

   ELSE
     vertex[4].x := localxdest  - 2.0*localxcomp ;
     vertex[4].y := localydest  + 2.0*localxcomp ;
   END IF;
```

```
        {EnvelopePolygon and PlaneIcon belong to the Sector Manager}

          ASK (ASK Plane PlaneEnvelopePolygon) TO SetPoints(vertex);
          ASK (ASK Plane PlanePlaneIcon) TO
                SetTranslation(localxpos, localypos);
          ASK (ASK Plane PlanePlaneIcon) TO
                Rotate(localpheading - Pi/2.0);

          ASK (ASK Plane PlanePlaneGraphic) TO Draw();

        END METHOD;


ASK METHOD UpdateEnvSectors(IN Plane:PlaneObj);

VAR
  queueOld, queueNew:  SectorQueueObj;
  loopsec, secA, secB, secAtemp:  SectorObj;
  secnum, loopsecnum: INTEGER;
  REMOVEDorNOT:  BOOLEAN;
  qx, qy, xstart, ystart, xstop, ystop: INTEGER;
  xscale, yscale:  REAL;
   xsector,ysector : INTEGER;
  xborder,yborder: REAL;
  m , b , yint,xint : REAL;
  NewSector : ARRAY INTEGER , INTEGER OF STRING ;
  elem: PlaneObj;
   ID:  INTEGER;
  i,j : INTEGER;
  Indicator : STRING;
  Member:PlaneObj;

 BEGIN

   {routine called by SectorManager in Report Position method}
   { of the spatial template}

   Member := Plane;

   ID := ASK Plane ID;

   Indicator :=  "NO";

   (***Initialize the NewSector array ***)
```

```
NEW ( NewSector , 1 .. numofsecx , 1 .. numofsecy ) ;

FOR j := 1 TO numofsecy
  FOR i := 1 TO numofsecx
      NewSector [ i , j ] := "ABSENT";
  END FOR ;
END FOR ;

(*** Determine the new list of sectors for the envelope of this plane***)

{repeat for 4 pairs of vertices of the evelope:
     V[1], V[2];  V[2], V[3]; V[3], V[4]; V[4], V[1]
     for x coor and then for y coor, where V is vertex array}

FOR i := 1 TO 4
  j := (i MOD 4) + 1;


  (*** Define m and b for the line equation y=mx+b only for non-vertical
        lines   ***)

  IF ( vertex[i].x <> vertex[j].x )
      m := ( vertex[j].y - vertex[i].y ) / ( vertex[j].x - vertex[i].x ) ;
      b := vertex[i].y - m*vertex[i].x ;

      {OUTPUT(" From Moving Object Update EnvelopeSector vertex[i]= ",
         vertex[i].x, " Vertex[i].y= ", vertex[i].y);}
  END IF;

  (*** first, add the "x pairs"  if line is not vertical ***)

  IF ( vertex[i].x <> vertex[j].x )

     FOR xsector := 2 TO numofsecx

       xborder := xrange * ( FLOAT(xsector-1)/FLOAT(numofsecx) - 0.5 );

       IF ((MINOF (vertex[i].x,vertex[j].x) < xborder) AND
           (MAXOF(vertex[i].x,vertex[j].x) > xborder))

               yint := m*xborder + b ;
               yborder := -yrange/2.0;
               ysector := 1;
```

```
            WHILE ( yborder < yint )
             ysector := ysector + 1 ;
             yborder := yrange *
                        ( FLOAT(ysector-1)/FLOAT(numofsecy) - 0.5 );
            END WHILE;

            qx := xsector-1;
            qy := ysector-1;

            NewSector [ qx , qy ] := "ENVELOPE" ;
            qx := xsector;
            NewSector [ qx , qy ] := "ENVELOPE" ;

        END IF;
     END FOR;

  END IF;


  (*** now add the "y pairs" ***)

  FOR ysector := 2 TO numofsecy

   yborder := yrange * (FLOAT (ysector-1)/FLOAT(numofsecy) - 0.5 );

   IF ( (MINOF(vertex[i].y, vertex[j].y) < yborder) AND
              (MAXOF(vertex[i].y, vertex[j].y) > yborder) )

      IF ( vertex[i].x <> vertex[j].x )

     xint := (yborder-b)/m ;
      ELSE
       xint := vertex[i].x;
    END IF;

      xborder := -xrange/2.0;

      xsector := 1;

      WHILE ( xborder < xint )
     xsector := xsector + 1;
     xborder := xrange *
                  ( FLOAT (xsector-1)/FLOAT(numofsecx) - 0.5 );
      END WHILE;
```

```
        qx := xsector -1;
        qy := ysector -1;

        NewSector [ qx , qy ] := "ENVELOPE" ;
        qy := ysector;
        NewSector [ qx , qy ] := "ENVELOPE" ;

    END IF;
  END FOR;

 END FOR;
{END repeat for 4 pairs of vertices of the evelope}

{NOW the NewSector 2-dim array has the word ENVELOPE at every
   point where the plane should be; each point is the ID of a sector
   else the array hold the word ABSENT}

Indicator := "YES";
OUTPUT("Before the dump sector plane is ", ASK Plane ID);

ASK SELF DumpSectors;

{SectorArray created and init by GenSectors which
   ask the SectorManager object to InitSectorQueue
   the init value is ABSENT}


{ while loop that stops if indicator <> yes}

{Indicator will be set to NO if a conflict is identified}

 j := 1;
 i := 1;
WHILE ( i <= numofsecx ) AND (Indicator = "YES")
 WHILE ( j <= numofsecy ) AND (Indicator = "YES")
   { WHILE ( i <= numofsecx ) AND (Indicator = "YES")}

    {}
      IF ( SectorArray [ ID, i , j ] <> NewSector [ i , j ] )
      AND ( SectorArray [ ID, i , j ] <> "PRESENT")

      {update the permanent array with the plane}
        SectorArray [ ID , i , j ] := NewSector [ i , j ] ;

      ASK MasterSectorArray [i,j] TO
```

```
                  ChangeSectorStatus (Plane, NewSector[i,j], Indicator) ;

     END IF;
    {i := i + 1;}j := j + 1;
   END WHILE;

   {i := 1;}   j := 1;
  { j := j + 1;}        i := i + 1;
  END WHILE;

  ASK ConflictResolver TO Doit( Plane, Indicator);


  {dispose of the created array}
  DISPOSE (NewSector);

END METHOD ;
ASK METHOD SetStationaryEnvSectors(IN Stationary:StationaryObj);

VAR
  queueOld, queueNew:  SectorQueueObj;
  loopsec, secA, secB, secAtemp:  SectorObj;
  secnum, loopsecnum: INTEGER;
  REMOVEDorNOT:  BOOLEAN;
  qx, qy, xstart, ystart, xstop, ystop: INTEGER;
  xscale, yscale:  REAL;
  xsector,ysector : INTEGER;
  xborder,yborder: REAL;
  m , b , yint,xint : REAL;
  NewSector : ARRAY INTEGER , INTEGER OF STRING ;
  elem: PlaneObj;
   ID:  INTEGER;
  i,j : INTEGER;
  Indicator : STRING;
  Member : StationaryObj;
  StationaryVertex:PointArrayType;

 BEGIN

        NEW(StationaryVertex, 1..4);


        StationaryVertex[1].x := ASK Stationary  StationaryVertexX1;
        StationaryVertex[1].y := ASK Stationary  StationaryVertexY1;
        StationaryVertex[2].x := ASK Stationary  StationaryVertexX2;
```

```
      StationaryVertex[2].y := ASK Stationary  StationaryVertexY2;
      StationaryVertex[3].x := ASK Stationary  StationaryVertexX3;
      StationaryVertex[3].y := ASK Stationary  StationaryVertexY3;
      StationaryVertex[4].x := ASK Stationary  StationaryVertexX4;
      StationaryVertex[4].y := ASK Stationary  StationaryVertexY4;



 Member := Stationary;

 {routine called by SectorManager in Report Position method}
 { of the spatial template}

 ID := ASK Stationary ID;


 (***Initialize the NewSector array ***)

  NEW ( NewSector , 1 .. numofsecx , 1 .. numofsecy ) ;

 FOR j := 1 TO numofsecy
   FOR i := 1 TO numofsecx
       NewSector [ i , j ] := "ABSENT";
   END FOR ;
 END FOR ;

(*** Determine the new list of sectors for the envelop of this plane***)

 {repeat for 4 pairs of vertices of the evelope:
      V[1], V[2];  V[2], V[3]; V[3], V[4]; V[4], V[1]
      for x coor and then for y coor, where V is vertex array}

 FOR i := 1 TO 4
   j := (i MOD 4) + 1;


   (*** Define m and b for the line equation y=mx+b only for non-vertical
          lines   ***)

   IF ( StationaryVertex[i].x <> StationaryVertex[j].x )
        m := ( StationaryVertex[j].y - StationaryVertex[i].y ) / ( StationaryVertex[j].x -
StationaryVertex[i].x ) ;
        b := StationaryVertex[i].y - m*StationaryVertex[i].x ;
      {OUTPUT(" From Moving Object Update EnvelopeSector StationaryVertex[i]=
",
        StationaryVertex[i].x, " Vertex[i].y= ", StationaryVertex[i].y);}
    END IF;

    (*** first, add the "x pairs"  if line is not vertical ***)

    IF ( StationaryVertex[i].x <> StationaryVertex[j].x )

      FOR xsector := 2 TO numofsecx

       xborder := xrange * ( FLOAT(xsector-1)/FLOAT(numofsecx) - 0.5 );

       IF ((MINOF (StationaryVertex[i].x,StationaryVertex[j].x) < xborder) AND
          (MAXOF(StationaryVertex[i].x,StationaryVertex[j].x) > xborder))

            yint := m*xborder + b ;
            yborder := -yrange/2.0;
            ysector := 1;

            WHILE ( yborder < yint )
             ysector := ysector + 1 ;
             yborder := yrange *
                          ( FLOAT(ysector-1)/FLOAT(numofsecy) - 0.5 );
            END WHILE;

            qx := xsector-1;
            qy := ysector-1;

            NewSector [ qx , qy ] := "StationaryEnvelope" ;
            qx := xsector;
            NewSector [ qx , qy ] := "StationaryEnvelope" ;

         END IF;
     END FOR;

   END IF;


   (*** now add the "y pairs" ***)

   FOR ysector := 2 TO numofsecy

    yborder := yrange * (FLOAT (ysector-1)/FLOAT(numofsecy) - 0.5 );
```

```
 IF ( (MINOF(StationaryVertex[i].y, StationaryVertex[j].y) < yborder) AND
           (MAXOF(StationaryVertex[i].y, StationaryVertex[j].y) > yborder) )

    IF ( StationaryVertex[i].x <> StationaryVertex[j].x )

   xint := (yborder-b)/m ;
    ELSE
     xint := vertex[i].x;
  END IF;

    xborder := -xrange/2.0;

    xsector := 1;

    WHILE ( xborder < xint )
   xsector := xsector + 1;
   xborder := xrange *
                  ( FLOAT (xsector-1)/FLOAT(numofsecx) - 0.5 );
    END WHILE;

    qx := xsector -1;
    qy := ysector -1;

    NewSector [ qx , qy ] := "StationaryEnvelope" ;
    qy := ysector;
    NewSector [ qx , qy ] := "StationaryEnvelope" ;

  END IF;
 END FOR;

END FOR;
{END repeat for 4 pairs of vertices of the evelope}

{NOW the NewSector 2-dim array has the word ENVELOPE at every
   point where the plane should be; each point is the ID of a sector
   else the array hold the word ABSENT}

Indicator := "NotMovable";
OUTPUT("Before the dump sector Stationary is ", ASK Stationary ID);

ASK SELF DumpSectors;

{SectorArray created and init by GenSectors which
   ask the SectorManager object to InitSectorQueue
   the init value is ABSENT}
```

```
{FUTURE: change to while loop that stops if indicator <> yes}

{Indicator will be set to NO if a conflict is identified}

 OUTPUT (" before update sector array; numofsec y and x: ",
       numofsecy, numofsecx);
 FOR j := 1 TO numofsecy
   FOR i := 1 TO numofsecx

     {this will be called for the static object only in those
     array positions of the NewSector array where it = StationaryEnvelope}

       IF ( SectorArray [ ID, i , j ] <> NewSector [ i , j ] )

       {update the permanent array with the static object}
         SectorArray [ ID , i , j ] := NewSector [ i , j ] ;


       ASK MasterSectorArray [i,j] TO
                        ChangeSectorStatus (Stationary, NewSector[i,j], Indicator) ;

       END IF;

        END FOR ;
  END FOR;

        {ASK MasterSectorArray[qx,qy] TO Add (Stationary);}
{****the way this stationary object is being added should be checked!!!!******}

  OUTPUT("DUMP After add to sectors; Stationary is ", ASK Stationary ID);

  ASK SELF DumpStationarySectors;


  {ASK ConflictResolver TO Doit( Plane, Indicator);}


  {dispose of the created array}
  DISPOSE (NewSector);

END METHOD ;
```

```
ASK METHOD DumpSectors;
VAR
        i,j: INTEGER;
        elem : GeoObj;

BEGIN
        OUTPUT("----------- D U M P   S E C T O R S -----------");
        FOR j := 1 TO numofsecy
        FOR i := 1 TO numofsecx
                elem := ASK MasterSectorArray [i,j] First();
                IF ( elem = NILOBJ ) OUTPUT("Sector [",i,",",j,"] is empty.");
                        ELSE
                        OUTPUT("Sector [",i,",",j,"] contains plane ",
                                ASK elem NameID);
                                elem := ASK MasterSectorArray[i,j] Next(elem);
                        END IF;
                WHILE (elem <> NILOBJ)
                        OUTPUT("              and ",ASK elem NameID);
                        elem := ASK MasterSectorArray[i,j] Next(elem);
                END WHILE;
        END FOR;
        END FOR;
END METHOD;


ASK METHOD DumpStationarySectors;
VAR
        i,j: INTEGER;
        elem : StationaryObj;

BEGIN
        OUTPUT("----------- D U M P   Stationary S E C T O R S -----------");
        FOR j := 1 TO numofsecy
        FOR i := 1 TO numofsecx
                elem := ASK MasterSectorArray [i,j] First();
                IF ( elem = NILOBJ ) OUTPUT("Sector [",i,",",j,"] is empty.");
                        ELSE
                        OUTPUT("Sector [",i,",",j,"] contains Stationary ",
                                ASK elem ID);
                                elem := ASK MasterSectorArray[i,j] Next(elem);
                        END IF;
                WHILE (elem <> NILOBJ)
                        OUTPUT("              and ",ASK elem ID);
```

```
                    elem := ASK MasterSectorArray[i,j] Next(elem);
                END WHILE;
            END FOR;
            END FOR;
        END METHOD;




            ASK METHOD UpdateSectors(IN Plane:PlaneObj);

            VAR
              queueOld, queueNew:  QueueObj;
              loopsec, secA, secB, secAtemp:  SectorObj;
              secnum, loopsecnum: INTEGER;
              REMOVEDorNOT:  BOOLEAN;
              qx, qy, xstart, ystart, xstop, ystop: INTEGER;
              xscale, yscale:  REAL;
              xpos,ypos,xcomp,ycomp,xdest,ydest:REAL;


            BEGIN
             xpos:= ASK Plane XPos;
             ypos:= ASK Plane YPos;

             (*** Set queueOld to be the previous list of sectors***)
              NEW ( queueOld );
            OUTPUT (" in update sectors; # of objects in SectorQ ",
                   ASK SectorQueue numberIn);
              secA := ASK SectorQueue First();
              WHILE ( secA <> NILOBJ)
                    ASK queueOld TO Add(secA);
                 secA := ASK SectorQueue Next(secA);
                 END WHILE;

              (*** Determine the new list of sectors ***)
              NEW ( queueNew );
                    OUTPUT("XPos,YPos = ",xpos,",",ypos);

              xscale := (xpos + xrange/2.0)/xrange*FLOAT(numofsecx);

              yscale := (ypos + yrange/2.0)/yrange*FLOAT(numofsecy);
              qx := INT(xscale)+1;
        qy := INT(yscale)+1;


        xstart := qx;
        xstop := qx;
        ystart := qy;
        ystop := qy;
        IF ( ( (xscale - FLOAT(INT(xscale))) <= 0.2 ) AND (qx <> 1) )
              xstart := xstart - 1;
              END IF;
        IF ( ( (yscale - FLOAT(INT(yscale))) <= 0.2 ) AND (qy <> 1) )
              ystart := ystart - 1;
              END IF;
        IF ( ( (xscale - FLOAT(INT(xscale))) >= 0.8 ) AND (qx <> numofsecx ) )
              xstop := xstop + 1;
              END IF;
        IF ( ( (yscale - FLOAT(INT(yscale))) >= 0.8 ) AND (qy <> numofsecy ) )
              ystop := ystop + 1;
              END IF;
        FOR qx := xstart TO xstop
              FOR qy := ystart TO ystop
                  ASK queueNew TO Add(MasterSectorArray[qx,qy]);
              END FOR;
        END FOR;


(*** Remove from both queueOld and queueNew those sectors which are in
     both queues.                                                ***)


         secA := ASK queueOld First();
         LOOP
         IF ( secA = NILOBJ ) EXIT;
             END IF;
         secB := ASK queueNew First();
         REMOVEDorNOT := FALSE;
          LOOP
               IF ( secB = NILOBJ ) EXIT;
           END IF;
               IF ( secA = secB )
           secAtemp := secA;
           secA := ASK queueOld Next(secA);
           ASK queueOld TO RemoveThis ( secAtemp ) ;
           ASK queueNew TO RemoveThis ( secB ) ;
```

```
            REMOVEDorNOT := TRUE;
            EXIT;
            END IF;
                secB := ASK queueNew Next(secB);
                END LOOP;
            IF (NOT REMOVEDorNOT) secA := ASK queueOld Next(secA);
                END IF;
          END LOOP;

  WHILE ( secA <> NILOBJ )
     secB := ASK queueNew First();
     WHILE ( secB <> NILOBJ)
           IF ( secA = secB )
              ASK queueOld TO RemoveThis ( secA ) ;
                ASK queueNew TO RemoveThis ( secB ) ;
                END IF ;
           secB := ASK queueNew Next(secB);
     END WHILE;
     secA := ASK queueOld Next(secA);
  END WHILE;


  (*** Inform any sectors remaining in queueOld that this plane is no longer                  in
its airspace. and remove the sector from the plane's SectorQueue.
                                                                        ***)


   secA := ASK queueOld First();
   WHILE ( secA <> NILOBJ )
       {  ASK secA TO RemoveThis ( Plane );}

      ASK SectorQueue TO RemoveThis ( secA);

         secA := ASK queueOld Next(secA);
    END WHILE;

   (*** Inform any sectors remaining in queueNew that this plane has entered
         its airspace,****)

   secB := ASK queueNew First();
   WHILE ( secB <> NILOBJ)
         ASK secB TO Add ( Plane);

         ASK SectorQueue TO Add ( secB );
         secB := ASK queueNew Next(secB);
   END WHILE;
END METHOD;

{The following method is being called from SPcheckupdate of the SpatialTempObj}
{and it appears it is called only when the plane is at its destination}

 ASK METHOD checkupdate(IN Plane: PlaneObj;
              IN TempInd: STRING);
 VAR
      j, i, qx, qy: INTEGER;
      ID: INTEGER;
    xscale, yscale : REAL;

 BEGIN

   ID:= ASK Plane ID;
   FOR j := 1 TO numofsecy
    FOR i := 1 TO numofsecx

       IF ( SectorArray [ID,i , j ] <> "ABSENT" )
          AND ( SectorArray [ ID, i , j ] <> "PRESENT")

         SectorArray[ID, i, j] := "ABSENT";
           ASK MasterSectorArray[i,j] TO
                ChangeSectorStatus (Plane, SectorArray[ID,i,j],TempInd );
          END IF;
    END FOR ;
  END FOR ;

  {do not remove the plane completely from these arrays, leave plane
    in it current position}

  IF ASK Plane Status = "ON HOLD"
       xscale := (ASK Plane XPos + xrange/2.0)/xrange*FLOAT(numofsecx);

          yscale := (ASK Plane YPos + yrange/2.0)/yrange*FLOAT(numofsecy);
          qx := INT(xscale)+1;
          qy := INT(yscale)+1;

        SectorArray[ID, qx, qy] := "PRESENT";
          ASK MasterSectorArray[qx,qy] TO Add (Plane);
  END IF;

   {####Rajesh dont do this here look for where lobna does this
```

```
    erase polygon but not plane icon...

      ASK PlaneIcon TO Erase();
        ASK EnvelopePolygon TO Erase();}

             {DISPOSE(PlaneIcon); }
             {DISPOSE(EnvelopePolygon); }
             {DISPOSE(PlaneGraphic);}




      END METHOD;
END OBJECT;

END MODULE.
```

```
DEFINITION MODULE CCSpatialTempObj;

            (***** Spatial Templet Object *****)


FROM GrpMod IMPORT QueueObj;
FROM GTypes IMPORT PointArrayType;
FROM ACMove IMPORT PlaneObj;
FROM CCGeoObj IMPORT GeoObj;
FROM CCStationaryObj IMPORT StationaryObj,Stationary;
FROM CCSectorObj IMPORT SectorObj, SectorManagerObj;

TYPE


SpatialTempletObj = OBJECT(QueueObj)
        Bxmin,Bymin,Bxmax,Bymax :REAL;
        Bvertex: PointArrayType;
        BsectorIDx, BsectorIDy: INTEGER;


        ASK METHOD SetFrame(IN xrange,yrange:REAL);
        ASK METHOD GenSectors;
        ASK METHOD SPcheckupdate(IN Plane:PlaneObj);
        ASK METHOD ReportPosition(IN Plane:PlaneObj);
        ASK METHOD ReportStationaryPosition(IN Stationary:StationaryObj);
        ASK METHOD Reportzerofindings(IN Plane:PlaneObj);
        ASK METHOD InformPotentialConf(IN commonsector:SectorObj; IN
Member,Rock:GeoObj;INOUT Indicator:STRING);




END OBJECT;{ SpatialTempletObj}

VAR
MasterSectorArray: ARRAY INTEGER, INTEGER OF SectorObj;
sector:SectorObj;
SpatialTemplet: SpatialTempletObj;
SectorManager: SectorManagerObj;
END MODULE.
```

```
IMPLEMENTATION MODULE CCSpatialTempObj;
FROM GTypes IMPORT PointArrayType;
FROM CCSectorObj IMPORT SectorObj, SectorManagerObj;
FROM CCGeoObj IMPORT GeoObj;
FROM ACMove IMPORT ModelInit,PlannerObj,PlaneObj;
FROM CCStationaryObj IMPORT StationaryObj,Stationary;
FROM CConflictIdentObj IMPORT ConflictIdentifierObj,ConflictIdentifier;
FROM CCGlobal IMPORT xrange, yrange, numofsecx,
numofsecy,numofhrs,ObjectNumber;

FROM Image IMPORT ImageObj;
FROM Fill IMPORT PolygonObj;
FROM Window IMPORT WindowObj;
FROM Animate IMPORT DynDClockObj;
FROM Graphic IMPORT GraphicLibObj;

OBJECT SpatialTempletObj;

 ASK METHOD SetFrame(IN  localxrange,localyrange:REAL);

VAR
        Bvertex: PointArrayType;
BEGIN

    Bxmin := localxrange * (-0.5);
    Bxmax := localxrange * (0.5);
    Bymin := localyrange * (-0.5);
    Bymax := localyrange * (0.5);
  NEW(Bvertex, 1..4);
  Bvertex[1].x := Bxmin; Bvertex[1].y := Bymin;
  Bvertex[2].x := Bxmax; Bvertex[2].y := Bymin;
  Bvertex[3].x := Bxmax; Bvertex[3].y := Bymax;
  Bvertex[4].x := Bxmin; Bvertex[4].y := Bymax;

 END METHOD;


   ASK METHOD GenSectors;
   VAR
     i,j : INTEGER;
     tempsector:SectorObj; {use tempsector instead of global var: sector
                      below someday}

   BEGIN
```

```
    {create the sectors and set their attributes;
     save the sector reference pointer in the MasterSectorArray}

    {sector and MasterSectorArray are variables that are global
     for all routines of the Spatial Template;
     It's definition is in DCCSpatialTempObj.mod}

    FOR j:= 1 TO numofsecy;
      FOR i:=1 TO numofsecx ;
              NEW ( sector );
              ASK sector TO SetobjectID(i,j);

         { set this sector's atributes: sectorIDx , sectorIDy,
                                   xmin,ymin,xmax,ymax,
                     and the inherited attributes: vertex,
                                        Polygon
          }

              ASK sector TO SetBorders(xrange,yrange,
numofsecx,numofsecy);
              ASK (ASK sector Polygon) TO Draw();

          {save the reference pointer to this sector}
              MasterSectorArray[i,j] := sector;

              END FOR;
    END FOR;

       NEW( SectorManager); {SectorManager is a variable that is global
                    for all routines of the Spatial Template;
                     It's definition is in DCCSpatialTempObj.mod}

     {create the SectorManager's vertex attribute (don't assign values)
      and do graphical icons: envelopepolygon, planeicon}

       ASK SectorManager InitEnvelope;

     {create more sector manager's attributes:
               SectorQueue, SectorEnvQueue,
          SectorArray (a 3-dim array
                with 1rd dim = "ABSENT"
                   2th and 3rd dim stand for the sector ID)
              each value in this array corresponds to a position
                in an
                individual sector object's queue (the queue
                 is called using the sector obj's reference
                 pointer name)
    }

   ASK SectorManager InitSectorQueue;

 END METHOD;

ASK METHOD ReportPosition(IN Plane:PlaneObj);
VAR
      planestatus: STRING;

BEGIN
      planestatus:= ASK Plane Status;

      OUTPUT(" From the SpatialTemplet the planestatus  of plane ",
          ASK Plane ID,  planestatus);

      IF (planestatus = "AT DESTINATION")

              ASK SELF SPcheckupdate(Plane);

      ELSIF (planestatus ="NEWPOSITION")

              {##### rajesh get rid InitEnvelope; InitSectorQueue;
          should be done only once and in GenSectors}

              {**ASK SectorManager TO InitEnvelope;
           ASK SectorManager TO InitSectorQueue;**}


              ASK SectorManager TO UpdateEnvelope(Plane);
              ASK SectorManager TO UpdateEnvSectors(Plane);

        {##### rajeshcan't figure this out will drop it for now:
              ASK SectorManager TO UpdateSectors(Plane);}



       ELSIF (planestatus ="INITIALIZED")
              {#### rajesh get rid InitEnvelope; InitSectorQueue;
          should be done only once and in GenSectors}

              {ASK SectorManager TO InitEnvelope;
      ASK SectorManager TO InitSectorQueue;**}


     {set SectorManager's vertex field and
       draws plane and sets polygon pts}

          ASK SectorManager TO UpdateEnvelope( Plane);

     {uses the sector manager's vertex fields to
      Determine the new list of sectors for the envelop
     of this planes and start the process}

          ASK SectorManager TO UpdateEnvSectors( Plane);

   {##### rajeshcan't figure this out will drop it for now:
          ASK SectorManager TO UpdateSectors(Plane);}

  ELSIF (planestatus ="NOT GIVEN PERMISSION")
          {#### rajesh get rid InitEnvelope; InitSectorQueue;
      should be done only once and in GenSectors}

          {ASK SectorManager TO InitEnvelope;
        ASK SectorManager TO InitSectorQueue;}

           ASK SectorManager TO UpdateEnvelope( Plane);
           ASK SectorManager TO UpdateEnvSectors( Plane);

    {##### rajeshcan't figure this out will drop it for now:}
           {ASK SectorManager TO UpdateSectors(Plane);}


 ELSE

   OUTPUT(" The plane is not Init, Dest, new");

 END IF;
   END METHOD;

ASK METHOD ReportStationaryPosition(IN Stationary:StationaryObj);

    BEGIN
            {ASK SectorManager TO SetStationaryEnvelope( Stationary);}
            ASK SectorManager TO SetStationaryEnvSectors( Stationary);

    END METHOD;
```

```
ASK METHOD SPcheckupdate(IN Plane:PlaneObj);
     VAR
      sp: STRING;
     BEGIN
     sp:= "YES";
     ASK SectorManager checkupdate( Plane, sp);

END METHOD;

 ASK METHOD Reportzerofindings(IN Plane:PlaneObj);
   VAR
     planestatus: STRING;
     BEGIN
      planestatus:= ASK Plane Status;
     IF (planestatus <> "GIVEN PERMISSION")

      ASK ConflictIdentifier TO Noconflict(Plane);
      END IF;
  END METHOD;



ASK METHOD InformPotentialConf(IN commonsector: SectorObj;
                          IN Member,Rock:GeoObj;
                          INOUT Indicator:STRING);

VAR
 SIDX, SIDY, P1ID,P2ID: INTEGER;
  plane1,plane2 : PlaneObj;

BEGIN

 {P1ID := ASK plane1 ID;
 P2ID := ASK plane2 ID;}

 {OUTPUT(" SECTOR  ",  ASK commonsector sectorIDx," ,
           ",ASK commonsector sectorIDy,"  Plane ", P1ID,
            " Plane ", P2ID);}
  ASK ConflictIdentifier TO ReportPC
                  (commonsector,Member,Rock, Indicator);
 END METHOD;
END OBJECT;

END MODULE.
```

```
DEFINITION MODULE CCStationaryObj;


FROM RandMod IMPORT RandomObj;
FROM CCGlobal IMPORT ObjectNumber;
FROM CCGeoObj IMPORT GeoObj;

FROM Fill IMPORT PolygonObj;
FROM Image IMPORT ImageObj;
FROM GTypes IMPORT FillStyleType(HollowFill,SolidFill,NarrowDiagonalFill,
        MediumDiagonalFill,WideDiagonalFill,NarrowCrosshatchFill,
        MediumCrosshatchFill,WideCrosshatchFill);
FROM GTypes IMPORT PointArrayType;


TYPE

StationaryObj = OBJECT(GeoObj)

        StationaryVertexX1,StationaryVertexX2,
        StationaryVertexX3,StationaryVertexX4,
        StationaryVertexY1,StationaryVertexY2,
        StationaryVertexY3,StationaryVertexY4 : REAL;

        StationaryVertex: PointArrayType ;
        StationaryPolygon : PolygonObj;



        Name : STRING;

        ASK METHOD InitStationaryFields (IN localname : STRING);


END OBJECT;

VAR

        Stationary : StationaryObj;



END MODULE.
```

```
IMPLEMENTATION MODULE CCStationaryObj;

FROM CCGlobal IMPORT xrange,yrange,ObjectNumber;
FROM RandMod IMPORT RandomObj;
FROM CCSpatialTempObj IMPORT SpatialTempletObj,SpatialTemplet;
FROM CCGeoObj IMPORT GeoObj;

FROM Fill IMPORT PolygonObj;
FROM Image IMPORT ImageObj;
FROM GTypes IMPORT FillStyleType(HollowFill,SolidFill,NarrowDiagonalFill,
        MediumDiagonalFill,WideDiagonalFill,NarrowCrosshatchFill,
        MediumCrosshatchFill,WideCrosshatchFill);
FROM GTypes IMPORT PointArrayType;


OBJECT StationaryObj;


ASK METHOD InitStationaryFields(IN localname : STRING);

VAR
        ranNumGen : RandomObj;



BEGIN
    {####Rajesh make sure you really want a separate RN generator here;
      if so you probably should set the seed}
    NEW (ranNumGen);

{       StationaryVertexX1 := (xrange- 2.0) * ((ASK ranNumGen Sample()) - 0.5);
        StationaryVertexY1 := (yrange- 2.0) * ((ASK ranNumGen Sample()) - 0.5);
        StationaryVertexX2 := (xrange- 5.0) * ((ASK ranNumGen Sample()) - 0.5);
        StationaryVertexY2 := (yrange- 5.0) * ((ASK ranNumGen Sample()) - 0.5);
        StationaryVertexX3 := (xrange- 2.0) * ((ASK ranNumGen Sample()) - 0.5);
        StationaryVertexY3 := (yrange- 2.0) * ((ASK ranNumGen Sample()) - 0.5);
        StationaryVertexX4 := (xrange- 5.0) * ((ASK ranNumGen Sample()) - 0.5);
        StationaryVertexY4 := (yrange- 5.0) * ((ASK ranNumGen Sample()) - 0.5);
}
        StationaryVertexX1 := 4.0;
        StationaryVertexY1 := 5.0;
        StationaryVertexX2 := 4.0;
        StationaryVertexY2 := 4.0;
        StationaryVertexX3 := 5.0;
        StationaryVertexY3 := 4.0;
        StationaryVertexX4 := 5.0;
        StationaryVertexY4 := 5.0;


        ASK SELF TO SetID(ObjectNumber);


        ASK SELF TO SetNameID(localname);

        ASK SpatialTemplet TO ReportStationaryPosition(Stationary);

    END METHOD;


    END OBJECT;

    END MODULE.
```

```
DEFINITION MODULE CSectQObj;

FROM GrpMod IMPORT QueueObj;

TYPE


SectorQueueObj = OBJECT(QueueObj)

        ASK METHOD LogicalAdd ( IN elem1: ANYOBJ );
END OBJECT;

END MODULE.
```

```
IMPLEMENTATION MODULE CSectQObj;

FROM GrpMod IMPORT QueueObj;


OBJECT SectorQueueObj;

        ASK METHOD LogicalAdd (IN elem1:  ANYOBJ);
        VAR
                elem2: ANYOBJ;
                status: STRING;
        BEGIN
                elem2 := ASK SELF First();
                status := "ABSENT";
                WHILE ( (elem2<>NILOBJ) AND (status="ABSENT") )
                        IF (elem2=elem1) status:="PRESENT";
                                END IF;
                        elem2 := ASK SELF Next(elem2);
                        END WHILE;
                IF (status="ABSENT") ASK SELF TO Add(elem1);
                        END IF;
        END METHOD;


 END OBJECT;
 END MODULE.
```

```
DEFINITION MODULE sendrec;
        PROCEDURE rajmain(INOUT status : STRING ): STRING;NONMODSIM;

END MODULE.
```

```
DEFINITION MODULE sendrec;
        PROCEDURE rajmain(INOUT status : STRING ): STRING;NONMODSIM;

END MODULE.
```

```
DEFINITION MODULE CConflictResolve;

 (***Conflict Resolver Object***)

FROM RandMod IMPORT RandomObj, FetchSeed, Random;
FROM GrpMod IMPORT QueueObj;
FROM GTypes IMPORT PointArrayType;

FROM AuxMathMod IMPORT DIST2D, INT;
FROM ACMove IMPORT ModelInit,PlannerObj,PlaneObj;
FROM CCGeoObj IMPORT GeoObj,Geo;



TYPE


ConflictResolverObj = OBJECT



ASK METHOD Resolve(IN plane1,plane2:GeoObj;
              INOUT Indicator:STRING;
              IN DesVar1,DesVar2:REAL );
ASK METHOD Doit(IN plane1:PlaneObj; INOUT Indicator:STRING);

ASK METHOD DoitStatic(IN plane1:GeoObj; INOUT Indicator:STRING);

{ASK METHOD Noconflict(IN plane1:PlaneObj);}

END OBJECT;

VAR
ConflictResolver: ConflictResolverObj;

END MODULE.
```

```
IMPLEMENTATION MODULE CConflictResolve;


FROM ACMove IMPORT PlannerObj,PlaneObj,ReportEstimatedTime;
FROM SimMod IMPORT PendingListDump;
FROM CCSpatialTempObj IMPORT SpatialTempletObj,SpatialTemplet;
FROM CCGeoObj IMPORT GeoObj,Geo;
FROM sendrec IMPORT rajmain;


OBJECT ConflictResolverObj;



{####***********RAJESH   INOUT indicator}
ASK METHOD Resolve (IN plane1,plane2:GeoObj;
               INOUT  Indicator:STRING;
               IN  DesVar1,DesVar2:REAL );



VAR
 planestatus1,planestatus2:STRING;
 Diff:REAL;


BEGIN

   {enter this routine with the absolute Diff < 5 and hence a conflict}

   Diff:=  DesVar1 - DesVar2;

   OUTPUT(" the DIfference in the times = ", Diff);

   planestatus1:= ASK plane1 Status;

   OUTPUT ("######## Potential Conflict  exist between plane######");

   OUTPUT(" Plane ", ASK plane1 ID, "and Plane ",
          ASK plane2 ID);

    {Currently plane1 is put on hold when there is a conflict;
     NOTE: this Indicator is used for plane1 only}

    Indicator:= "ON HOLD";
```

```
END METHOD;



ASK METHOD Doit(IN plane1:PlaneObj; INOUT Indicator:STRING);
 VAR

 planestatus1: STRING;
 TestMethod : STRING;
 status : STRING;
 BEGIN
    OUTPUT ("IN DOIT: plane1 id: ", ASK plane1 ID, " with indicator ",
           Indicator);
    planestatus1:= ASK plane1 Status;

    IF (Indicator= "YES")

      {set status}
      ASK plane1 ChangeStatus(plane1,"GIVEN PERMISSION");
        status := "GIVENPERMISSION";
        TestMethod := rajmain(status);

        IF (TestMethod = "NewMove")
              TELL plane1 NewMove(plane1);
        END IF;

    ELSIF (Indicator ="ON HOLD")
       ASK plane1 ChangeStatus(plane1,"NOT GIVEN PERMISSION");

       {this does the remove from the MasterSector and 3-dim SectorArray}
       ASK SpatialTemplet SPcheckupdate(plane1);

       OUTPUT("SHOULD REMOVE ENVELOPE");
       TELL plane1 Delaymove(plane1);

    ELSIF (Indicator = "ON STATIC")

        ASK SELF TO DoitStatic(plane1,Indicator);

         {do nothing}
    END IF;

    PendingListDump(TRUE);
```

```
  END METHOD;

ASK METHOD DoitStatic(IN plane1:GeoObj;INOUT Indicator:
                                                STRING );
 VAR

 planestatus1: STRING;

 BEGIN
   OUTPUT ("IN DOITSTATIC: plane1 id: ", ASK plane1 ID, " with indicator
",Indicator);

   planestatus1:= ASK plane1 Status;

   { IF(Indicator="YES")##########removing}

      {set status}
        TELL plane1 NewArrive1(Indicator);

   { END IF;}

END METHOD;



END OBJECT;
END MODULE.
```

```
/*****************************************************************/
/* UNIX test programs for two process communicating across  FIFOs
**
** 08AUG91 Dutch Guckenberger initial edit
** two way test
** 21AUG91 Dutch Guckenberger chg header info & fix bugs
** 25AUG91 Dutch Guckenberger mod sim version to go with
**        definition module of same name .mod
*/

#include <stdio.h>
#include <errno.h>
#include <fcntl.h>
#include <math.h>
#include "clips.h"
#define FALSE 0
#define TRUE 1
#define enum {FALSE,TRUE} BOOLEAN
#define MAXOPEN 7


static struct {
 long key;
 int fd;
 int time;
}fifos[MAXOPEN];

#define MAXTRIES 3
#define NAPTIME 5

syserr(str)
char *str;
{
 printf("error stub");
}

static char *fifoname (key) /* construct fifo name from key */
long key;
{

  static char fifo[20];

  sprintf(fifo, "/tmp/fifo%ld", key);
  return(fifo);
```

```
}

static int openfifo(key, flags) /* return fifo fd */
long key;
int flags;
{

  static struct {
   long key;
   int fd;
   int time;
  }fifos[MAXOPEN];
  static int clock;
  int i, avail, oldest, fd, tries;
  char *fifo;
  extern int errno;

  clock++;
  avail = -1;
  for (i = 0; i <MAXOPEN; i++) {
   if (fifos[i].key == key) {
    fifos[i].time = clock;
    return(fifos[i].fd);
   }
   if (fifos[i].key == 0 && avail == -1)
    avail = i;
  }
  if (avail == -1) { /* all fds in use; find oldest */
   oldest = -1;
   for (i=0; i<MAXOPEN; i++)
    if (oldest == -1 || fifos[i].time <oldest) {
     oldest = fifos[i].time;
     avail = i;
    }
   if (close(fifos[avail].fd) == -1 )
    return (-1);
  }
  fifo=fifoname(key);
  if (mkfifo(fifo) == -1 && errno != EEXIST)
   return(-1);
  for (tries = 1; tries <=MAXTRIES; tries++) { /* await writer */

   if ((fd = open(fifo, flags | O_NDELAY)) != -1)

    break;
   if (errno != ENXIO)
     return (-1);
   sleep(NAPTIME);
  }
  if (fd == -1) {
   errno =ENXIO; /* sleep may have messed it up */
   return (-1);
  }

  if (fcntl(fd,F_SETFL,flags) == 1 ) /*clear 0_NDELAY */
   return(-1);

  fifos[avail].key = key;
  fifos[avail].fd = fd;
  fifos[avail].time = clock;
  return(fd);
}

int send(dstkey, buf, nbytes) /* send message */
long dstkey;
char *buf;
int nbytes;
{
  int fd;

  if ((fd = openfifo(dstkey, O_WRONLY)) == -1)
    return(FALSE);
  return(write(fd, buf, nbytes) != -1);
}

int receive (srckey, buf, nbytes) /* receive message */
long srckey;
char *buf;
int nbytes;
{
  int fd,  nread;

  if ((fd = openfifo(srckey, O_RDONLY)) == -1)
    return(FALSE);
  while ((nread = read(fd, buf, nbytes)) == 0)
    sleep(NAPTIME);
  return(nread != -1);
}
```

```
void rmqueue(key) /* remove message queue fifo */
long key;
{
 int errno;

 if (unlink(fifoname(key)) == -1 && errno != ENOENT)
   syserr("unlink");
}

int mkfifo(path) /* make FIFO */
char *path;
{
 return(mknod(path,S_IFIFO | 0666,0));
}




/************* process adder client *********/
#include "addmsg.h"


/*float recmain()  adder client */
char * rajmain(status) /* adder client */

{
  char result[100];
  MESSAGE  m;
  int x,y;
  char Chumma[100];
  m.clientkey = getpid();
  for (x = 1; x <= 5; x++)
    for (y = 1; y <= 5; y++) {
      m.x = x;
      m.y = y;

      printf("\nRaj sending x= %d  and y=%d ",x,y);
      /*sprintf(Chumma,"%d",x);
      AssertString(Chumma);*/
      if (!send(ADDERKEY, &m, sizeof(m)))
        syserr("send");
      if (!receive(m.clientkey, &m, sizeof(m)))
        syserr("receive");
       printf("\nRaj received x= %d  and y=%d sum=%d ",m.x,m.y,m.sum);
           sprintf(Chumma,"%d",x);
        /*AssertString(Chumma);*/
    if (x + y != m.sum){
      printf("Addition error!\n");
      exit(1);
        }


   }

 rmqueue(m.clientkey);
 printf("%ld worked OK\n", m.clientkey);
 /*AssertString("Tats test");*/
 /*exit(0); */


}

/*****command line is adder&addclient&addclient&  ****/

float recmain()

  {
  /*AssertString("this is a fact");*/
  return(1.0);
  }

/*****command line is adder&addclient&addclient&  ****/
/*

 char* itos(strp,n)
      char* strp;
      int n;

      {
      sprintf(strp,"%d",n);
      return(strp);
      }

 */
```

```
/*******************************************************/
/*   "C" Language Integrated Production System    */
/*                                  */
/*          A Product Of The                */
/*        Software Technology Branch          */
/*        NASA - Johnson Space Center          */
/*                                  */
/*        CLIPS Version 5.00  11/19/90         */
/*                                  */
/*          CLIPS HEADER FILE              */
/*******************************************************/

/*************************************************************/
/* Purpose:                          */
/*                                  */
/* Principal Programmer(s):                  */
/*   Gary D. Riley                    */
/*                                  */
/* Contributing Programmer(s):                 */
/*                                  */
/* Revision History:                    */
/*                                  */
/*************************************************************/

#ifndef _H_clips
#define _H_clips

#include "setup.h"
#include "constant.h"
#include "clipsmem.h"
#include "symbol.h"
#include "router.h"
#include "sysdep.h"
#include "expressn.h"
#include "evaluatn.h"
#include "facts.h"
#include "constrct.h"
#include "utility.h"

#include "intrfile.h"

#if DEFRULE_CONSTRUCT
#include "defrule.h"
#include "engine.h"
#include "drive.h"
```

```
#endif

#if DEFFACTS_CONSTRUCT
#include "deffacts.h"
#endif

#if DEFTEMPLATE_CONSTRUCT
#include "deftempl.h"
#endif

#if DEFGLOBAL_CONSTRUCT
#include "defglobl.h"
#endif

#if DEFFUNCTION_CONSTRUCT
#include "deffnctn.h"
#endif

#if DEFGENERIC_CONSTRUCT
#include "genrccom.h"
#include "genrcfun.h"
#endif

#if OBJECT_SYSTEM
#include "extobj.h"
#endif

/*******************************/
/* OTHER FUNCTION PROTOTYPES */
/*******************************/

#if ANSI_COMPILER
#if (BLOAD || BLOAD_ONLY || BLOAD_AND_BSAVE)
  int                   Bload(char *);
#endif
#if BLOAD_AND_BSAVE
  int                   Bsave(char *);
#endif
  int                   LoadFacts(char *);
  int                   SaveFacts(char *);
  int                   SetAutoFloatDividend(int);
  int                   GetAutoFloatDividend(void);
  VOID                  InitializeCLIPS(void);
#if DEFRULE_CONSTRUCT
  BOOLEAN               PPDefrule(char *,char *);
  VOID                    ListDefrules(void);
#endif
#else
#if (BLOAD || BLOAD_ONLY || BLOAD_AND_BSAVE)
  int                   Bload();
#endif
#if BLOAD_AND_BSAVE
  int                   Bsave();
#endif
  int                   LoadFacts();
  int                   SaveFacts();
  int                   SetAutoFloatDividend();
  int                   GetAutoFloatDividend();
  VOID                    InitializeCLIPS();

#if DEFRULE_CONSTRUCT
  BOOLEAN                    PPDefrule();
  VOID                  ListDefrules();

#endif
#endif

#endif
```