TSAFE: Building a Trusted Computing Base for Air Traffic Control Software

by

Gregory D. Dennis

Bachelor of Science in Computer Science and Engineering

Submitted to the
Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 4, 2003

Author ....
Department of Electrical Engineering and Computer Science
February 4, 2003

Certified by ....
Daniel N. Jackson
Associate Professor
Thesis Supervisor

Accepted by ....
Arthur C. Smith
Chairman, Department Committee on Graduate Students

TSAFE: Building a Trusted Computing Base for Air Traffic Control Software

by

Gregory D. Dennis

Submitted to the
Department of Electrical Engineering and Computer Science

February 4, 2003

In partial fulfillment of the requirements for the degree of
Master of Engineering in Computer Science and Engineering

Abstract

The Tactical Separation Assisted Flight Environment, or TSAFE, is a new tool designed to aid air traffic controllers in detecting and resolving short-term conflicts between aircraft. Conceived at NASA Ames Research Center, TSAFE was proposed as a principle component of a larger Automated Airspace Computing System, whose goal it was to shift the burden of maintaining aircraft separation from controllers to computers. This paper retraces the history of TSAFE, examines the motivation for such a tool, and fully discusses the design and implementation of a TSAFE prototype. In doing so, it seeks to illustrate how one can turn a potentially complex system into a simple and intuitive Trusted Computing Base. Lastly, it proposes future avenues for research involving TSAFE.

Thesis Supervisor: Daniel N. Jackson
Title: Associate Professor

Acknowledgments

This thesis is the culmination of nearly two years of research. It would not have happened if it were not for the help, guidance, and friendship of a number of individuals.

This project began in the summer of 2001 with a trip to NASA Ames Research Center in Moffett Field, California. It was there I met Heinz Erzberger, the inventor of TSAFE, to whom I owe special thanks for his initial conception of the tool and his efforts to help us understand it. Thanks, too, to Russ Paielli, Michelle Eschow, and others at NASA Ames for answering all our questions about TSAFE and air traffic control.

I would like to ackowledge the efforts of Cristian Cadar, Nathan Doble, Tudor Leu, and Professors Martin Rinard and John Hansman for their collaboration on the analysis of TSAFE's requirements. Special thanks to Cristian and Tudor for their help with the early implementation of the prototype.

For his development of FIG, and for always letting me bounce ideas off him, thanks to Roshan Gupta, my congenial officemate. Roshan also developed the TSAFE Configuration Panel.

Professor Leslie Kaelbling and her graduate student Yu-Han Chang made a number of useful suggestions regarding the applicability of machine learning techniques to TSAFE. I have included a number of those within.

And special thanks to Tom Reynolds for his insights into conformance monitoring and for allowing me to bug him with endless questions. The conformance monitoring techniques employed by TSAFE are largely the results of his research.

Lastly, I owe a great deal of gratitude to Professor Daniel Jackson, the supervisor of this thesis. I thank him for his invaluable comments, support, and guidance on this project and on all my efforts, both research-oriented and academic. I look to forward to participating in future research projects in his Software Design Group.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The Tactical Separation Assisted Flight Environment, or TSAFE, is a new tool designed to aid air traffic controllers in detecting and resolving short-term conflicts between aircraft. Conceived by Heinz Erzberger, Chief Scientist for air traffic control at NASA Ames Research Center, TSAFE was proposed as a principle component of a larger Automated Airspace Computing System (AACS), whose goal it was to shift the burden of maintaining aircraft separation from controllers to computers [3].

Air traffic control software systems are generally highly complex systems of machinery. Having been developed experimentally, with the goal of emulating controller inference, they typically consist of complex heuristics and intricate algorithms, many of which were added and tweaked in a very ad hoc manner. As a result, ATC systems are notoriously large and messy and intractable in terms of analysis and verification. I began this project intending to take TSAFE in the opposite direction. That is, I hoped to turn it into a small, trusted, and verifiable component–a Trusted Computing Base.

This paper retraces the history of TSAFE, examines the motivation for such a tool, and fully discusses the design and implementation of a TSAFE prototype. In doing so, it intends to illustrate how one can build a potentially complex system as a simple and intuitive Trusted Computing Base. Much of this work, especially the early pre-prototype work, was done in conjunction with the NASA Ames Research Center and the MIT International Center for Air Transportation.

The prototype I have built of TSAFE performs two primary functions: *conformance monitoring* and *trajectory synthesis*. By "conformance monitoring," we mean the process of determining the degree to which a flight is conforming, or not conforming, to its instructions. The term "trajectory synthesis" refers to the prediction and construction of the aircraft's future trajectory, or 4-dimensional path. To perform these functions, TSAFE analyzes flight plan and radar data received from either a live feed or from recorded files. It displays both the observed and computed flight data on an interactive graphical user interface.

The chapters, which have been arranged more or less chronologically, recount the research and prototyping of TSAFE that I and others have conducted to date. Chapters 2 through 4 discuss the background information and analysis of TSAFE done in preparation for the prototype. In Chapter 2, I retrace the history and motivation for building TSAFE and detail each of its components as they were initially proposed. In Chapter 3, I discuss our efforts to more fully understand the requirements of TSAFE through various diagrams and models. And Chapter 4 recounts the decisions I and others made as we sorted out the logistics and scope of the prototype we intended to build.

Chapters 5 through 7 detail the TSAFE prototype itself. Chapter 4 describes the algorithms TSAFE employs for conformance monitoring and trajectory synthesis. In Chapter 6, I relate the details of the software design and architecture of TSAFE. Chapter 7 concludes the discussion of the prototype with a thorough description of the graphical user interface through which a user operates TSAFE.

The final chapters, 9 and 10, talk about post-prototype work and potential research. In Chapter 9, I discuss some preliminary data analysis performed with the TSAFE prototype. Chapter 10 brings the thesis to a close with suggestions for future avenues of research.

The reader who is unfamiliar with basic air traffic control terminology may find it helpful to read the glossary in Appendix A on page 61 before continuing on to Chapter 2.

Chapter 2

History of TSAFE

The Tactical Separation Assisted Flight Environment, or TSAFE, is a proposed new decision support tool for air traffic controllers. Its function is to monitor aircraft for potential violations of separation in the short-term and issue clearances in the form of short conflict-free trajectories. It was initially proposed by Dr. Heinz Erzberger, Senior Scientist for Air Traffic Management at the NASA Ames Research Center as an integral part of his Automated Airspace Concept.

2.1 Motivation for TSAFE

In today's Air Traffic Control (ATC) system, air traffic controllers are primarily responsible for maintaining aircraft separation. Controllers accomplish this by surveilling radar data for potential conflicts and issuing clearances to pilots to alter their trajectories accordingly. Ground-based automated tools play only a supportive role in this process [3].

Under this current system, the airspace within the United States operates at only half its potential capacity. Experience has shown controllers' workload limits to be the fundamental limiting factor in the current model. Despite attempts at resectorization–dividing the airspace into smaller sectors to dilute the workload–and development of enhanced ATC decision support tools, researchers have failed to circumvent these fundamental workload limits. Erzberger claims that exploiting the full airspace capacity requires a new paradigm. He refers to this new paradigm as the *Automated Airspace Concept* (AAC) [3].

Under the AAC framework, automated mechanisms would play a primary role in maintaining aircraft separation. Aircraft would remain in direct connection with a ground-based automated system, which would transmit conflict alerts and air traffic control clearances via a persistent two-way data link. (The concept includes a fail-safe mode in which controller-pilot communication takes over in emergencies and other special circumstances.) By shifting much of the responsibility of aircraft separation from controllers to automated systems, AAC will allow controllers to focus more on long-term strategic traffic management, and thereby allow for a safe increase in the volume of aircraft per sector.

The role of TSAFE is as an independent monitor of this AAC Computing System. It is to act as a reliable safety net—a last line of defense against inevitable imperfections in the AAC model. Its job is to probe for short-term conflicts and issue avoidance maneuvers accordingly. As an independent monitor, it must sit outside of the primary system, on separate hardware, and on a separate software process, yet be privy to the same data as the AAC.

TSAFE differs in purpose and functionality from the existing conflict avoidance systems CTAS and TCAS. Whereas CTAS performs long-term conflict prediction on the order of 20-40 minutes ahead, and whereas TCAS detects conflicts only seconds away, TSAFE is intended to detect conflicts somewhere between 3 and 7 minutes in the future. Because TCAS operates on the order of seconds, it only considers aircraft *state information*-velocities, headings, altitudes, etc. TSAFE and CTAS, on the other hand, must also take *intent information* into account, including flight routes, cruise altitudes, and cruise speeds. But due to TSAFE's shorter time horizon, its algorithms must be simpler and less computationally intensive than those of CTAS.

2.2 TSAFE Components

In the initial proposal, TSAFE was to consist of five main components: (1) a Conformance Monitor, (2) a Trajectory Synthesizer, (3) a Conflict Detector, (4) a Conflict Avoidance Module, and (5) a Controller Interface. An overview of these four components follows.

2.2.1 Conformance Monitor

Conformance monitoring is the process of determining the degree to which a flight is following its instructions. In other words, conformance monitoring ask the question: "Is a flight where it's supposed to be?" If the answer is yes, the flight is said to be *conforming*. If the answer is no, the flight is said to be *not conforming*, or *blundering*.

To determine whether a flight is conforming, the flight's current state is measured against a *conformance basis*, a model of the ideal state of the flight. In the simplest of models, the conformance basis is just derived from the flight plan. More sophisticated models allow for several conformance bases at once. In such a system, if a flight is found to be not conforming to a particular basis, the conformance monitor may dynamically hypothesize a new conformance basis and perform a new comparison. The new conformance basis may try to account for controller clearances and directive that were given, but never actually entered into the ATC system.

TSAFE falls somewhere in between these two approaches. To construct a conformance basis, TSAFE makes use of the flight plan, flight plan amendments, cruising altitude, controller clearances, and other intent information. Drawing on this data, it generates a 3-dimensional volume, which we will call the *planned path*, which represents the space in which the aircraft should ideally be located. The function of TSAFE's conformance monitor is to compare each aircraft's current position and heading against its planned path.

A conformance monitor will assign a flight a 'conformance status' based on the degree to which the flight is adhering to the conformance basis. Depending on the particular monitor, the range of possible statuses can vary greatly. For some conformance monitors, the status may be boolean, i.e. each flight is either labeled as "conforming" or "blundering." Other conformance monitors may utilize a large multi-dimensional spectrum of judgments.

In the case of TSAFE, horizontal and vertical conformance statuses are computed for each flight. There are eight separate horizontal and eight separate vertical conformance statuses, each indicating a different degree of flight conformance. As a result, TSAFE may place a flight into one of sixty-four different conformance status combinations at any given time.

2.2.2 Trajectory Synthesizer

TSAFE must be able to synthesize two different types of trajectories, *nominal trajectories* and *dead reckoning trajectories.* A nominal trajectory assumes the flight will eventually rejoin its planned path, regardless of its current state. The dead reckoning trajectory, on the other hand, assumes the flight will stay "on course." This may simply mean that the flight retains its current velocity and heading, though the exact meaning may differ between implementations.

2.2.3 Conflict Detector

The Conflict Detector, or Conflict Probe, will then probe along one or both of the nominal and dead reckoning trajectories, searching for points at which two flights break legal separation. Precisely which trajectories are used will vary according to the conformance statuses of the flights in question. The Detector will provide timely and reliable warnings to controllers should any imminent loss of separation be detected.

2.2.4 Conflict Avoidance Module

In response to high-risk conflict warnings, the Conflict Avoidance Module calculates appropriate avoidance maneuvers. The maneuver is not intended to deliver a long-term conflict-free trajectory, but rather a brief controller clearance that should steer the aircraft out of danger for about three minutes. Nevertheless, the modules would need to factor in nearly all flight state, flight intent, weather, airspace geometry, and relevant geography data available in order to make this calculation.

2.2.5 Controller Interface

The avoidance maneuvers and conflict warnings are relayed to the controller in the form of visual and aural signals via the Controller Interface. Later enhancements to the interface may offer voice recognition technology so as to limit the number of keyboard commands the controller must execute.

Chapter 3

Problem Analysis

To develop a better understand of TSAFE's requirements, we began our analysis by constructing a series of diagrams and models of its structural and computational properties. These artifacts helped us to conceptualize the interaction between the components of TSAFE and brought to light a number of questions we had yet to answer.

3.1 Data Flow Diagram

We began by constructing an **Data Flow Diagram**, to give us a better grasp of how data flows between modules in TSAFE. This diagram can be found in Appendix B on page 63.

The flow of data begins on the right-hand side of the diagram at the *Static and Dynamic Databases.* The Static Database contains information we expect not to change throughout TSAFE's operation. As the diagram illustrates, this includes the locations of navaids and fixes, as well as aircraft performance models and data regarding airspace geometry. The information in the dynamic database, on the other hand, is expected to change continually. This database would store information such as to the positions and velocities of flights, flight plans and routes, and weather data.

The *Planned Path Synthesizer* uses the aircraft state, intent, and performance models, along with weather information provided by the databases to construct a three-dimensional volume, or *planned path*, of the flight's expected locations. The *Conformance Monitor* compares the flight's actual state information to the planned path to determine to what degree, if any, the flight is deviating from where it should be. Based on the degree and type of deviation, the flight is assigned a particular conformance status, which is then outputted by the Conformance Monitor.

Given the conformance status, the *Probing Strategy Function* chooses a conflict probing strategy for the flight. The strategy specifies the types of trajectories–nominal, dead reckoning, or both–along which conflict detection will be performed and the look ahead times to use for each trajectory, as well as whether or not to conduct a vertical airspace search. The *Conflict Probe* follows the probing strategy dictated by the Probing Strategy Function, invoking the *Nominal* and *Dead Reckoning Trajectory Synthesizers* to generate the nominal and dead reckoning trajectories as needed. The Trajectory Synthesizers draw information from the Databases for their calculations.

Once the conflicts have been determined, TSAFE must notify the controllers of them and, depending on the severity of the conflict, possibly suggest avoidance maneuvers for the conflicting flights. If the conflict low-risk, i.e. it is not scheduled to occur for some sufficiently long period of time, the conflict is sent only to the *Alerting Mechanism*, which generates an appropriate level of alert and notifies the controller via the *Controller Interface.*

If the conflict is high risk, it is sent to both to the Alerting Mechanism and the *Conflict Avoidance Module.* The Conflict Avoidance Module must pull nearly all available information from the databases and from previous calculations in order to calculate and appropriate avoidance maneuver. In this case, the Alerting Mechanism would generate a high-level an alert. Both the avoidance maneuver and the alert are relayed to the controller via the Controller Interface.

3.2 Computation Graph

The Data Flow Diagram forced us to precisely enumerate the conceptual components of TSAFE, but we felt that it too strongly implied a specific architecture. The diagram also brings the modules to the forefront, when it was the required data and the composite structures of data that we were primarily interested in. We concluded that creating the dual graph of the DF Diagram–which would put the data objects (positions, flight plans, conflicts, etc) at the nodes and move the computations (trajectory synthesis, conformance monitoring, etc) to the edges–would serve as a better aid in analyzing the nature of TSAFE.

This decision prompted the creation of the TSAFE **Computation Graph**, which can be found in Appendix C on page 65. As noted, this graph is essentially the dual of the Data Flow Diagram, with improvements. It links each data component to the data components that are necessary for its derivation. Fittingly, it shows the conflict to be the final result, or purpose, of all prior derivations, and shows all the intermediary computations that TSAFE must perform along the way. (At this point in our study of TSAFE, we had decided to simplify the problem and not look at conflict avoidance or alerting, so they are not included in this diagram).

3.3 Abstract Alloy Model

Clearly, much effort has been invested into understanding the workings of TSAFE. But how do we know that at a fundamental level, the design of TSAFE even allows for accurate conflict prediction? The logic inherent in TSAFE's structure may in fact prohibit this.

This is what I intended to investigate in building the **Abstract Alloy Model** of TSAFE. The model I built abstracted the computational complexities of trajectory prediction and conflict detection into a black box. The model boiled the problem domain down to just flights, trajectories, locations, and points of time, and sought to investigate one hypothesis:

All other things being equal, if flights are following the same trajectories in a given time instance that they were following in the previous instance, the predicted conflicts in both instances should be the same.

The Alloy Analyzer could find no counterexamples to this assertion up to a scope of four, a sufficient scope it seemed for this model. This gave me confidence that at least TSAFE was not fundamentally flawed in this regard. The full Alloy model can be found in Appendix D on page 67.

Chapter 4

Planning the Prototype

At this point in our work with TSAFE, we felt we had exhausted the potential for the type of high-level problem analysis we had been conducting. There were only so many issues we could flush out at the black-box level before engaging in some sort of implementation. So we decided to build a toy model of TSAFE–a prototype–that we would use to more fully explore this tool's potential. But before doing so, there were a number of issues we would first have to resolve.

4.1 ETMS Data Source

One of the initial hurdles we had to surmount was how to obtain access to up-to-date information on flight positions, flight plans, flight plan amendments, etc. across the U.S. We managed to gain access to this data by connecting to an Enhanced Traffic Management System (ETMS) feed. ETMS is used by the FAA to "to predict, on national and local scales, traffic surges, gaps, and volume based on current and anticipated airborne aircraft" [4]. It is not typically used for conflict detection and resolution purposes because it obtains updated information at a relatively slow rate (at approximately 1 minute intervals); however we believed it would prove adequate for our purposes.

4.2 Simplifications

Before beginning the prototype, we made a series of deliberate simplifications and approximations of the initially proposed design in order to give ourselves a more manageable problem with which to being our attempt at a prototype.

4.2.1 Focus on Conformance Monitoring

Our first and most major modification was to focus our development primarily on the Conformance Monitor. Though the prototype would still include a trajectory synthesis module, we decided to leave out the conflict detection and avoidance modules from our first version, yet build the tool with knowledge that they may be added later.

The decision to focus primarily on conformance monitoring made sense for a number of reasons. First and foremost, we wanted to tackle a smaller problem first, rather than try to build all of TSAFE at once. Moreover, a brief look at the Data Flow Diagram, Appendix B (p63), and Computation Graph, Appendix C (p65), shows the Conformance Monitor to be the appropriate starting point in terms of the data flow. That is, the calculation of a flight's conformance status is necessary for trajectory synthesis; and naturally the synthesized trajectories are required before TSAFE can probe them for conflicts; and of course, the conflicts must be computed before they can be resolved.

The decision to focus on conformance monitoring also made sense given the possibilities we had for collaboration with other researchers. Most notably, conformance monitoring happened to be the research focus of Tom Reynolds, a PhD in the Aero-Astro department with whom we were already collaborating on this project. It also offered the greatest potential for applicability of machine learning techniques, an area in which we could benefit immensely from our additional collaboration with AI Professor Leslie Kaelbling.

As discussed below, we also made some significant simplifications to the originally proposed methods for conformance monitoring and trajectory synthesis.

4.2.2 Conformance Monitoring Simplifications

Our principle simplification with regard to conformance monitoring was to use flight plans and their amendments as the only conformance bases for flights. This differs from Erzberger's initial design in which he proposes continually changing conformance bases. Under his proposal, if a flight is deemed to not be conforming to the issued flight plan, TSAFE could dynamically hypothesize a new conformance basis against which the flight could be compared.

We do not allow for this dynamic derivation of alternate conformance bases. In our model, the operation of the Conformance Monitor is reduced to performing a static comparison of a flight's position and heading against its flight plan. Note that this means the conformance monitor does not make use of any trajectories, and therefore has no dependencies on the Trajectory Synthesizer. In addition, we do not provide for a multitude of conformance statuses. Instead, our conformance status is just boolean: either the flight is conforming or it is blundering.

4.2.3 Trajectory Synthesis Simplifications

We have also simplified the prediction and synthesis of trajectories from TSAFE's proposed strategy. In contrast to the proposed algorithm, if our prototype deems a flight to be conforming, we will not look along either the dead reckoning or the nominal trajectory. Instead, we will probe a trajectory that runs along the flight's assigned route, at the its assigned speed and altitude. We will refer to this as the *planned trajectory*. If a flight is blundering, we will use the dead reckoning trajectory, which we will define to be a straight line projection along the flight's velocity vector.

Note that this greatly simplifies trajectory synthesis by eliminating the notion of a nominal trajectory, which would have otherwise required us to implement intensely complex calculations. We make a further simplification to trajectory synthesis by ignoring vertical intent information.

4.3 Desired Features

Before beginning the implementation phase, we enumerated a number of specific features, outside of the Conformance Monitor and Trajectory Synthesizer, that we wanted the prototype to include.

First, we wanted the ability to visually display flights, fixes, blunders, routes, and trajectories on a user-friendly graphical display. Secondly, we wanted the prototype to be general enough to fit a number of different conformance monitoring and trajectory synthesis. Indeed, a prototype architecture that is not adaptable to other calculation schemes will not be useful for making a claim about air traffic control systems generally. Thus, our prototype should allow us to "plug-in" different conformance monitoring and trajectory synthesis modules without altering its overall architecture.

Another feature we deemed necessary was the ability to record and playback the data feed. Our thought was that a user should be able to record a feed and play it back on several occasions, with a possibly different set of settings or parameters each time. This was an essential feature to incorporate because it would allow the user to learn which settings worked best, and would enable me, as the developer, to replay data over which the system is known to fail.

Chapter 5

Algorithms

Having decided that the prototype would be limited to conformance monitoring and trajectory synthesis, I set out to devise algorithms for these functions. Given that TSAFE was intended to operate on a 3-7 minute time horizon, it was important that these algorithms be simple, so that the logic would cascade quickly.

One interesting feature of the algorithms, and one which contributes to their simplicity, is that none of them rely on historical data. That is, the calculation of a flight's conformance status and the prediction of its trajectory are based only on the flight's most recently observed state, and not on any past state. The algorithms are discussed below.

5.1 Conformance Monitoring

Our conformance monitoring algorithm draws heavily on the work of Tom Reynolds. It is essentially an implementation of the approach to monitoring that he advocates in his research.

5.1.1 Conformance Monitoring as a Model-Based Fault Detection Problem

Reynolds approaches the task of conformance monitoring as a model-based fault detection problem. In a model-based fault detection problem, one attempts to detect faults in a "real world" system by comparing it against a model of the expected behavior of the system [7]. The diagrams in this subsection were either taken directly from, or adapted from, Reynold's work.

The first step in this detection process is known as *residual generation*. The *residual* is a measure of the dissimilarity between the observed state of the real world system and the expected state as provided by the model; residual generation is the process of calculating the residual. Following residual generation comes a process known as *decision making*, in which the residual is analyzed to determine if it indicates a fault in the real world system. (In the most simplest case, the residual is simply a scalar and the decision-making process is just a threshold test.) If a fault is detected, a process known as *fault isolation* may be performed to determine the reason for its occurrence. This is typically done by assigning the real world state at the time of the fault to some category of known fault-inducing scenarios [7]. Figure 5-1 shows a conceptual structure of the model-based fault detection concept.

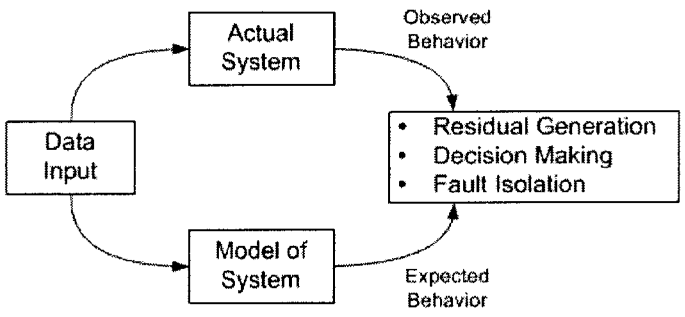


Figure 5-1: Model-Based Fault Detection Concept

As Reynolds notes, this model-based fault detection technique is readily translated to the air traffic control domain. In the ATC domain, the "real world" system becomes the real world airspace, i.e. the flight locations, speeds, altitudes, headings, accelerations, bank angles, etc. The model of expected behavior, which we refer to as the Conformance Monitoring Model (CMM), is essentially the entirety of instructions given to the flights, including the filed flight plans (flight routes, assigned altitudes, and assigned speeds), flight plan amendments, and any and all clearances and directives given to the flights by air traffic control, as well as any reasonable extrapolations thereof. A fault is deemed to have occurred, therefore, when a flight's actual state deviates significantly from the expectation for that flight as determined by the CMM. To detect such a fault, one can use the same techniques of residual generation and decision making described above. In the air traffic control domain, the process identified above as fault isolation falls under the term *intent inferencing*, which is the task of determining the intent of a pilot whose flight is off course, i.e. the reason for the apparent blunder [7]. Figure 5-2 shows the model-based fault detection concept translated to the ATC domain.

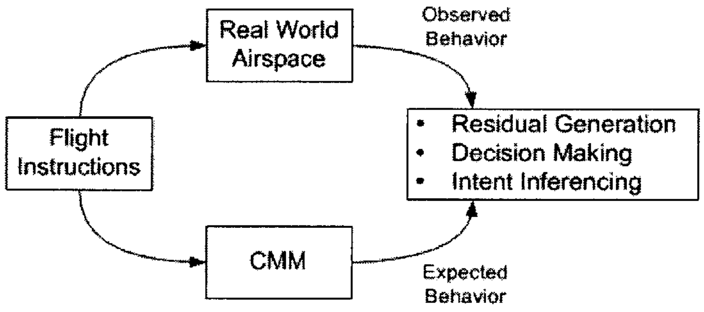


Figure 5-2: Model-Based Fault Detection Concept in the ATC Domain

We have integrated this model-based fault detection approach to conformance monitoring into TSAFE's normal execution. The data we receive over the ASDI feed provides us with the speed, position, and altitude of every civilian aircraft in the airspace. Although, it does not supply any flight headings, TSAFE approximates the heading once it has received two positional data points for the flight. As for intent data, the feed delivers filed flight plans, each of which consists of a flight route, assigned altitude, and assigned speed, as well as all subsequent amendments to the flight plan. From the feed data, TSAFE constructs a state vector of observable values–latitude/longitude position, altitude, speed, and heading–for each flight. It is now the task of TSAFE's CMM to construct a state vector of expected values that can be compared to the observed values. TSAFE's conformance monitor does not currently perform any intent inferencing.

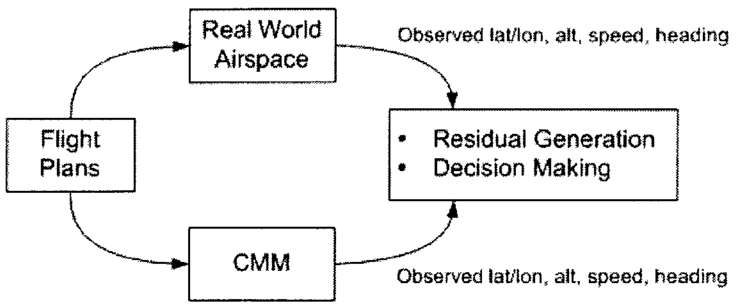

Real World Airspace
Observed lat/lon, alt, speed, heading
Flight Plans
• Residual Generation
• Decision Making
CMM
Observed lat/lon, alt, speed, heading

Figure 5-3: Model-Based Fault Detection Concept in TSAFE

5.1.2 Extracting the Expected State

As Figure 5-3 illustrates, to calculate the residual we must first be able to extract the expected state of the aircraft from Conformance Monitoring Model. The expected speed and the expected altitude are trivial to obtain, as they are copied directly from the assigned speed and assigned altitude given in the flight plan and flight plan amendments.

Calculating the expected position and heading, on the other hand, is not a trivial matter. Given that the feed provides TSAFE with each flight's filed route, assigned speed, and subsequent amendments to the route and speed, it is conceivable that a CMM could project along the route at the assigned speed to determine each flight's expected position and heading. However, because of the notorious inaccuracy of speed data provided by air traffic control systems, this is typically not the approach taken.

The more common approach, and the approach we have taken with TSAFE, is to "snap" the flight from its current position back to the flight route. This *snap-back point* is taken to be the flight's expected position. The heading of the route segment along which the snap-back point is located is taken to be the flight's expected heading. (If the snap-back point is at a joint in the flight route, the expected heading is taken to be the heading that is tangent to the joint.) Note that using the snap-back approach has the interesting property that it requires knowledge of where the flight is, before determining where it should be.

5.1.3 Calculating the "Snap-back" point

To determine the location of the snap-back point, we simply find the closest point on the flight route to the flight. This entails finding the snap-back point for each of the individual route segments and then picking the point from this set of snap-back points that is closest to the flight.

Geometry tells us that the point on a route segment from (x_1, y_1) to (x_2, y_2) that is closest to a flight at point p will either be (1) (x_1, y_1), (2) (x_2, y_2), or (3) the intersection of the line segment with a line orthogonal to the segment that intersects p. These three scenarios are illustrated below in Figure 5-4.

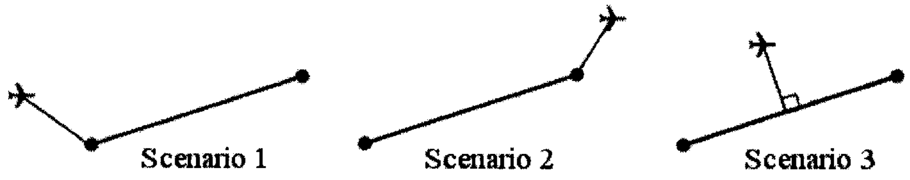


Figure 5-4: Closest point on a route segment to a flight.

We begin every snap-back computation by assuming that the route segment and flight match the third scenario, and then correct later if our assumption turns out to be false. The line l to which a line segment from (x_1, y_1) to (x_2, y_2) is coincident is defined as follows

$$y = \left(\frac{x_1 - x_2}{y_1 - y_2}\right)(x - x_1) + y_1 \tag{5.1}$$

We know the orthogonal line m will have a slope that is the negative reciprocal of the slope of l. And seeing as we know it intersects (x_p, y_p), the equation for m must be

$$y = -(\frac{y_1 - y_2}{x_1 - x_2})(x - x_p) + y_p \qquad (5.2)$$

With a little algebra, we can now find the point (x_i, y_i) at which the lines l and m intersect.

$$x_i = \frac{(mx_1 - y_1) - (nx_p - y_p)}{m - n} \qquad (5.3)$$

$$y_i = m(x_i - x_1) + y_1 \qquad (5.4)$$

$$\text{where } m = \frac{y_1 - y_2}{x_1 - x_2} \text{ and } n = \frac{1}{m}$$

If the point of intersection (x_i, y_i) lies on the route segment, then we know it must be the closest point on the route segment to the flight and we are done. However, if it lies outside the line segment, we know the arrangement much match either scenario 1 or scenario 2 (see Figure 5-4). In this case, we calculate the Euclidean distance from each of the endpoints to p and choose the endpoint that is closest as the snap-back point.

In sum, TSAFE's CMM obtains the expected altitude and speed directly from the flight plan and its amendments and calculates the expected position and heading using the snap-back approach, thereby constructing a vector of expected states to be compared against the observed.

5.1.4 Residual Generation and Decision Making

Having obtained both an observed state vector and expected state vector, TSAFE now has the necessary arguments for residual generation. The calculation of the residual involves at most four properties of aircraft: (1) latitude/longitude position, (2) altitude, (3) heading, and (4) speed. For each property, the algorithm calculates a deviation of the observed value from the expected value. For lat/long position this deviation is a Euclidean distance between the observed position and expected position. For altitude, heading, and speed it is simply the absolute value of the difference between the observed and expected. Each deviation is then multiplied by a weighting factor, the resulting products are added together, and the sum is then divided by the total number of properties used in the calculation. The resulting quotient is the residual. (If zero properties are used in the calculation, we define the residual to be zero). For n properties, the residual generation formula is

$$residual = \frac{1}{n}\sum_{i=1}^{n}\omega_i||x_{obs_i} - x_{exp_i}|| \tag{5.5}$$

TSAFE's decision-making process is a simple boolean test of whether this residual exceeds some preset threshold. If it does, the flight is deemed to be conforming; if not, it is deemed to be blundering.

5.2 Trajectory Synthesis

According to the decision discussed in 4.2.3 on page 29, if a flight is determined to be blundering, TSAFE synthesizes a dead reckoning trajectory for the flight. Otherwise, TSAFE will synthesize a planned trajectory, which follows the flight's assigned flight route at its assigned cruise speed and altitude.

Because a dead reckoning trajectory is a straight line, it can be defined by two points, the first of which will be the current 4-dimensional location of the flight in question. We will label this current position of the flight (x, y, z, t). If the synthesis uses a look-ahead time τ, and the flight has a speed σ, a flight can be expected to travel $\tau\sigma$ during the time horizon. So a flight with heading θ can expected to move $\tau\sigma cos(\theta)$ in the x direction and $\tau\sigma sin(\theta))$ in the y direction. Assuming the flight maintains its current altitude, the terminal point of the dead reckoning trajectory is

$$(x + \tau\sigma cos(\theta), y + \tau\sigma sin(\theta), z, t + \tau) \tag{5.6}$$

The synthesis of the planned trajectory is a little more involved. The first point in a flight's planned trajectory we take to be the snap-back point (which by this time TSAFE has already calculated in determining the flight's conformance status). We will label the 4-dimensional snap-back point (x_0, y_0, z_0, t_0). The algorithm then calculates the distance, δ_{01}, between this snap-back point and the next fix $(x_1, y1)$ in the flight route. Dividing this distance by the flight's speed, σ, yields the expected time t_{01} that it will take for the flight to reach that fix. If this time is less than the look-ahead time τ, then the point $(x_1, y_1, z_0, t_0 + t_{01})$ is added to the planned trajectory.

The algorithm then calculates the expected time t_{12} to the next fix located at (x_2, y_2). If $t_{01} + t_{12} < \tau$, then the flight should be able to reach this next fix within the time horizon, so the algorithm adds $(x_2, y_2, z_0, t_0 + t_{01} + t_{12})$ to the planned trajectory. This process continues until there is not enough time left to reach the next fix in the flight route. At this point, the algorithm dead reckons for the remaining time along the heading to the next fix and adds the terminal point of this dead reckoning trajectory to the planned trajectory.

Chapter 6

Software Design

TSAFE is divided into four main software modules: (1) the database, (2) the parser, (3) the engine, and (4) the client. Information regarding flights, flight plans, routes, trajectories, is stored in the form of simple data types and passed between modules. For a detailed understanding of the structure of the software structure, a Code Object Model and Modular Dependency Diagram are provided in appendices E (p71) and F (p73) , respectively. The graphic in Figure 6-1 illustrates the high-level architecture of TSAFE in terms of these four modules.

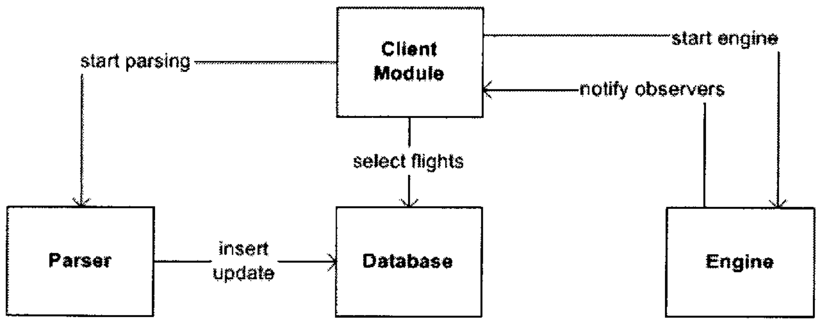


Figure 6-1: Module level architecture of TSAFE

6.1 Database

The database stores all the airspace data and observed flight data utilized by TSAFE. The airspace data remain unchanged throughout TSAFE's operation. They include the locations of fixes, navaids, airports, jet and victor routes, STARs, and SIDs, and they are used primarily to parse the text descriptions of flight routes into actual 2-dimensional paths. Unlike the airspace data, the flight data stored in the database change constantly. They consist of flight IDs, flight routes, flight latitude/longitude positions, headings, speeds, and altitudes, and they are continually updated by the feed parser and extracted by the client for display and analysis.

The relationship between the abstract class `TsafeDatabase` and its subclasses is an example of the "Template Method" design pattern. Template Method defines the skeleton of an algorithm, deferring certain *hook* methods for implementation by subclasses [5]. Consider, for instance, an algorithm for processing transactions on a bank account. This procedure, or *template method*, could invoke methods for calculating the interest earned, the amount of reserve credit, the minimum balance, the penalty for overdrawing, all of which could be deferred to subclasses for implementation. The specific implementation of the hook methods would vary depending on the type of account.

The `RuntimeDatabase` provided by the prototype stores its data in runtime data structures. Abiding by Template Method, `RuntimeDatabase` subclasses the abstract class `TsafeDatabase` and implements all the necessary hook methods listed in the superclass. These include methods like `insertFlight`, `selectFlight`, and so on. To amend the system to access another database, one would need to write another subclass of `TsafeDatabase` with an appropriate implementation of all the hook methods.

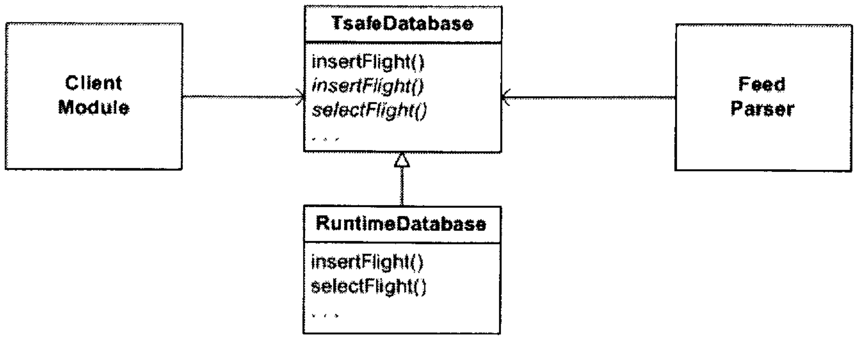


Figure 6-2: Template Method design pattern in the Database module.

6.2 Feed Parser

The feed parser reads radar and flight plan data from a feed source and continually updates the dynamic flight information in the database accordingly. Upon launch, the parser enters a loop, which it does not exit unless the feed terminates, it encounters an error reading the feed, or the client cues it to stop parsing.

The work of parsing the feed is divided amongst many classes. The main parsing loop is found in the `FeedParser` abstract class. Each time through the loop, `FeedParser` calls an abstract hook method `executeUpdate`, whose function it is to read a single message from the feed and update the database accordingly. The nature of reading the message and extracting its meaning is left to subclasses of `FeedParser` to implement. This is another example of the Template Method design pattern.

The prototype provides a single subclass of `FeedParser` called `ASDIParser` for parsing an ASDI ETMS data feed. Per the Template Method, one could write a parser for another feed by subclassing `FeedParser` and implementing `executeUpdate` in a manner specific to that source. In its `executeUpdate` method, `ASDIParser` reads a single text message from the feed and constructs a `Message` out of it. The constructor of `Message` parses the fields of the message that are common to all messages (time of the message, facility of origin, and type of message). This message is then passed to a `MessageExtractor` which fully parses the message and updates the database accordingly.

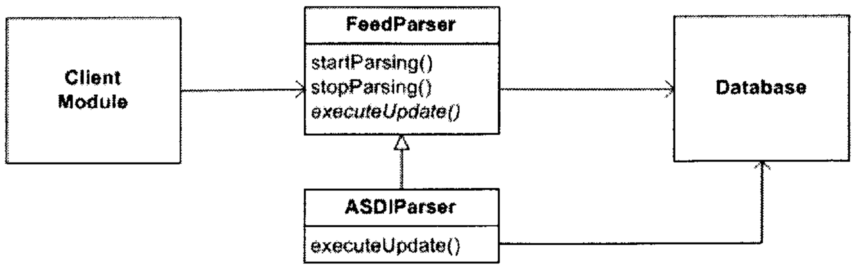


Figure 6-3: Template Method design pattern in the Feed Parser module.

6.3 The Engine

The Engine module is home to the algorithmic guts of TSAFE. It calculates the "snap-back" point, determines each flight's conformance status, and synthesizes predicted trajectories for flights. The relationship between the client and engine is that of observer and subject, ergo the "Observer" design pattern [5]. The client feeds the engine a set of flights and then launches the engine on a new thread. The engine cycles through each flight, and for each assigns a conformance status and synthesizes a trajectory. When finished, the engine notifies the client, sending the conformance and trajectory results in the notification.

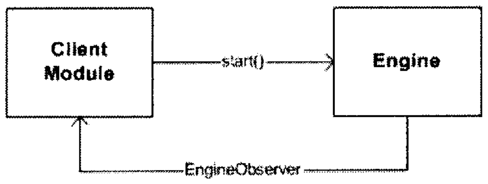


Figure 6-4: Observer pattern relationship between Client and Engine modules.

The structure of the engine module follows the "Mediator" design pattern. In Mediator, one component acts as a hub, mediating the interaction between several spokes [5]. In the case of the TSAFE engine module, the `TsafeEngine` class acts as a mediator between three spokes: `RouteTracker`, `ConformanceMonitor`, and `Trajectory Synthesizer`. Iterating through the set of flights it was given by the client, `TsafeEngine` sends data between these spokes to obtain the conformance status and a synthesized trajectory for each flight. Given a flight by the `TsafeEngine`, the `RouteTracker` returns the flight's snap-back point. Given the snap-back point and the flight, the `ConformanceMonitor` returns the flight conformance status. And based on its conformance status, `TsafeEngine` asks the `TrajectorySynthesizer` to synthesize the appropriate trajectory. When new algorithmic components, such as a conflict detector, are added to TSAFE, they can be incorporated quite easily as new spokes radiating from the `TsafeEngine` hub.

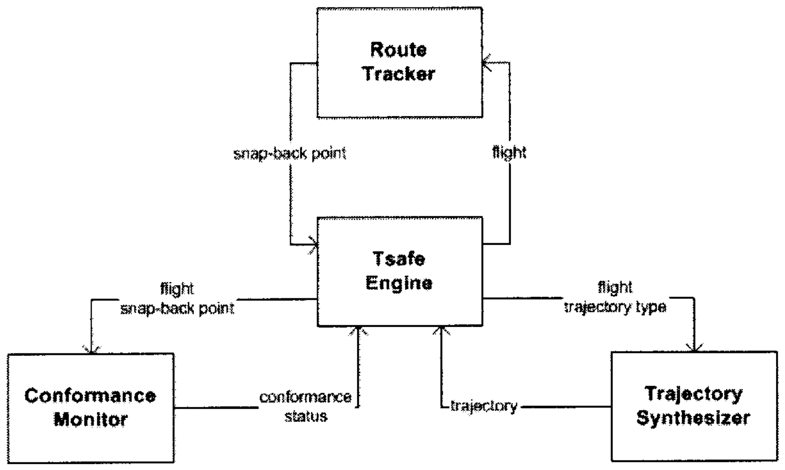


Figure 6-5: Mediator pattern in the Engine module.

6.4 The Client

The client is in charge of displaying flights, flight routes, trajectories, and conformance statuses of flights graphically on the screen. As the parser continually updates the dynamic flight data in the database, the client, operating in an asynchronous thread, periodically takes a snapshot of that data. It then passes this information on to the engine to calculate the conformance status and trajectory for each flight. It then draws both the observed and computed data on a graphical user-interface with which the user can interact. For more information on how the user can interact with the client's user interface, see Section 7.2.

6.5 Critique of Dependences

In the Modular Dependence Diagram found in Appendix F on page 73, an arrow is drawn from module P to module Q with the label A if P assumes Q fulfills specification A [6]. A glimpse at the diagram reveals the dependence structure to be largely hierarchical.

At the top of the hierarchy sits the client module. Though it depends on every other module, these dependences are essential to the client's coordination of inter-module interaction. The one regrettable dependence in the client module is the unnecessary coupling of `ParametersDialogue` to `EngineParameters`. This was done as a matter of convenience, so changes to the engine's parameters via the GUI would be automatically visible to the components of the engine module. However, this violates the property that all dependences from the Client module on the engine module originate at `TsafeClient`.

The only back dependence in the system is from `TsafeEngine` to `TsafeClient`. As noted above, this is a necessary consequence of using the Observer pattern. In an alternative design, the client could continually probe the engine until it found the engine computation to be complete. But for performance reasons, such a design would be highly undesirable.

As illustrated in the MDD, both the engine module and the ASDI submodule have dependences on a class called `SimpleCalculator` via the `EngineCalculator` specification. `SimpleCalculator` performs simple x/y to lat/long conversion and computes Euclidean distances between lat/long points. The original reason for the separate `EngineCalculator` interface was to enable others to code their own, possibly more complex, calculators and easily plug them into the system. Looking back, however, this was probably unnecessary, as one can just as easily rewrite the provided calculator methods, and there's no need for multiple calculator implementations to coexist.

Because the Feed Parser's task is to continually update the database, the parser module has an inevitable dependence on `TsafeDatabase`. `TsafeDatabase` has no dependences. And every module depends on the available data types: `Flight`, `FlightPlan`, `Trajectory`, etc. The dependences on these data types are essential, as instances of these classes carry the information the larger modules use to communicate with one another.

Chapter 7

Graphical User Interface

The TSAFE prototype provides two separate, but related, Graphical User Interfaces (GUIs). The first, entitled the *TSAFE Configuration Panel*, is used to configure the TSAFE client module for launch. The second, the *TSAFE Flight Console*, is provided by the client module, and as alluded to in the previous chapter, displays the flights, flight routes, trajectories, and conformance statuses of all the flights within the specified airspace.

7.1 TSAFE Configuration Panel

The TSAFE Configuration Panel is not actually a part of TSAFE, per se; rather, it functions as a separate tool whose purpose it is to configure TSAFE for launch. It consists of two tabs named "Data Sources" and "Map," which provide the functionality to set up the data sources TSAFE has access to and the geographic region over which TSAFE will display and analyze the data.

7.1.1 "Data Sources" Tab

In the upper portion of the "Data Sources" tab, the panel displays the location of selected data files. There are six such files listed, one each for fixes, airports, navaids, airways, SIDs, and STARs. With the exception of a few minor symbols, such as

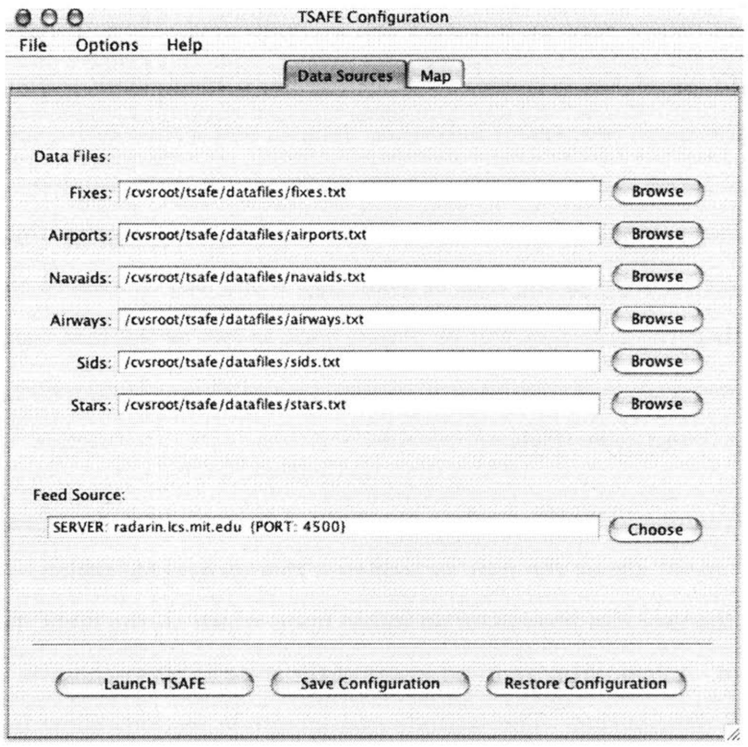
"XXX" to identify an unknown fix, these comprise the entire universe of tokens that can appear in flight routes received over an ETMS feed. These files are used by TSAFE to parse each text description of a flight route into a series of 2-dimensional straight line segments along the Earth's surface.

One important property of the files is that the names of the route elements should be unique across all the files. More specifically, no fix, navaid, airport, airway, SID, or STAR should share a name with any other fix, navaid, airport, airway, SID, or STAR. If multiple elements do have the same name, TSAFE makes no guarantees as to which one will be used when parsing the routes, or that the same one will be used in all circumstances. Thankfully the case of duplicate names, especially within a relatively small geographic area, is very rare. When the case does arise, the best option is generally to choose the route element that occurs most often and leave the other(s) out. This is the approach that was taken in the construction of the default data files provided in the TSAFE distribution.

At the bottom of the "Data Sources" tab, the user can select the feed source from which TSAFE will read the flight plan and radar data. The user has the option of choosing from pre-defined feed sources, editing the existing source, or creating new sources. There are currently two types of feed sources available, *server sources* and *FIG file sources*. A server source represents a direct connection to a server with access to the ETMS feed. To create a new server source, one must only supply the server name and port number. A screenshot of the "Data Sources" tab is shown in Figure 7-1 and the construction of a FIG file source is discussed in Section 7.1.2.

7.1.2 Configuring a FIG File Source

A Feed Input Generator (FIG) File source is an archive of feed data pre-recorded by FIG, a TSAFE companion tool written by Roshan Gupta. (For more information on FIG see http://sdg.lcs.mit.edu/atc/FIG/) A FIG file source can play back the messages in a single FIG file or in multiple FIG files. Playing feed data from multiple files entails playing all the messages from the first file, followed by all the messages from the second, and so on.



Data Files:

Fixes: /cvsroot/tsafe/datafiles/fixes.txt

Airports: /cvsroot/tsafe/datafiles/airports.txt

Navaids: /cvsroot/tsafe/datafiles/navaids.txt

Airways: /cvsroot/tsafe/datafiles/airways.txt

Sids: /cvsroot/tsafe/datafiles/sids.txt

Stars: /cvsroot/tsafe/datafiles/stars.txt

Feed Source:

SERVER: radarin.lcs.mit.edu {PORT: 4500}

Figure 7-1: TSAFE Configuration Panel "Data Sources" Tab.

As an added feature, FIG File Sources can be configured to filter out specific messages from the files and/or to play back them back at different speeds. The messages can be filtered according to the following criteria: (1) the airline of the flight the message is regarding, (2) the facility from which the message originated, and (3) the type of the message (flight position update, flight plan amendment, etc). In addition, the user can choose to automatically filter out badly formatted messages. A badly formatted message is one that has an incorrect number of fields for its given message type.

Though the message filtering is handy, arguably the most useful feature is the ability to change the speed at which messages are played back. When configuring a FIG file source, the user is given the option of scaling the delay between successive messages, setting a constant delay between messages, or eliminating the delay altogether. To scale the delay, the user must only provide a scaling factor by which all the recorded delays between messages will be multiplied. Thus, a scale factor of 1 plays the FIG file at the originally recorded speed; a scale factor of 0.5 plays the file at twice the speed (because it halves the delays); and a scale factor of 2 plays the file at half the speed (because it doubles the delay). If the user stipulates a constant delay, every message will be separated from its successive message by this delay regardless of the actual delay at the time of recording. If the user chooses to eliminate the delay altogether, the messages will be played back as fast as the host machine will allow.

7.1.3 "Map" Tab

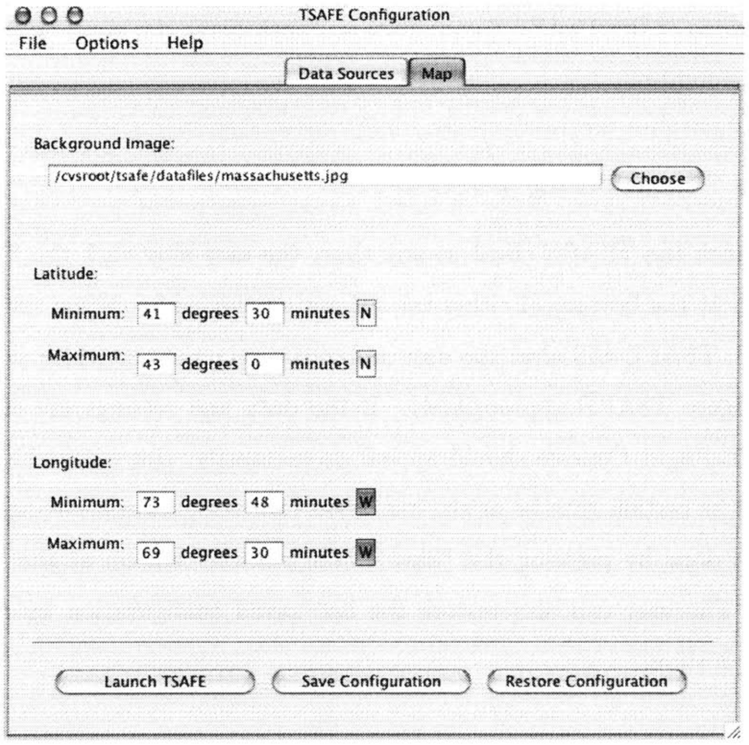
When running, TSAFE displays the flight data over a fixed geographic region. The "Map" panel allows the user to configure this geographic region. At the top of the panel, the user can browse for an image upon which all the flight data will be superimposed. A default background image of the state of Massachusetts is provided in the TSAFE distribution. The latitude and longitude sections allow the user to specify the bounds of the geographic region. A screenshot of the "Map" Tab can be found in Figure 7-2.

7.1.4 The Options Menu

The "Options" menu provides the ability to adjust settings on the Configuration Panel GUI, add and subtract default feed sources, edit the default FIG file content filter, and edit the list of airlines, facilities, and message types.

Selecting "GUI" under the "Options" menu brings up a dialogue containing a series of GUI-related preferences. One checkbox allows the user to enable or disable the display of the splash screen when the Configuration Panel is launched. The user can also toggle on and off two other features, one labeled "Remember Window Sizes" and the other "Remember Window Locations." When toggled on, these features will trigger the Configuration Panel to remember the sizes and locations of all the windows and dialogues and use these same window settings the next time the Panel is launched.

The "Options" menu also provides the ability to manage the list of default feed



Background Image: /cvsroot/tsafe/datafiles/massachusetts.jpg

Latitude:

Minimum: 41 degrees 30 minutes N

Maximum: 43 degrees 0 minutes N

Longitude:

Minimum: 73 degrees 48 minutes W

Maximum: 69 degrees 30 minutes W

Figure 7-2: TSAFE Configuration Panel "Map" Tab.

sources and to configure the default FIG file content filter. The default feed sources are those that are made available for quick selection when the user opts to change the source of the feed. To add or remove sources from this list, the user selects "Feed Sources" from the "Options" menu. The default content filter is simply the filter that is selected by default when a new FIG file source is created. The user can change the default filer by selecting "Default Filter" from the "Content Filter" submenu in the "Options" menu.

Recall that the messages from FIG file sources can be filtered according to the airline, facility, and message type. Using the "Options" menu, the user can manage the exact list of airlines, facilities, and message types that are available when constructing a filter. The user can access this functionality by selecting "Airlines", "Facilities", or "Message Types" from the "Content Filter" submenu in the "Options" menu.

If the user wants any of the settings discussed to be in effect for subsequent executions of the TSAFE Configuration Panel, he or she must be sure to save his or her preferences.

7.1.5 Launching TSAFE

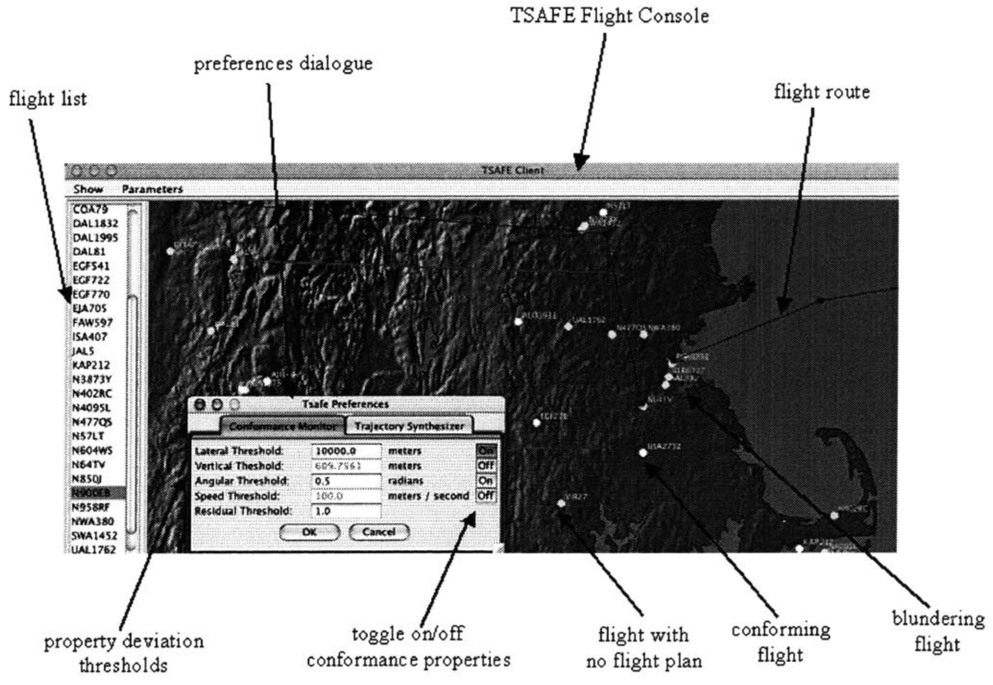
To launch the TSAFE client at any time, the user may click the "Launch TSAFE" button at the bottom of either tab. If any of the configuration data or settings are invalid, TSAFE will alert the user and will not launch, at which point the user and reconfigure TSAFE appropriately. If the data and settings are entirely valid, the TSAFE Flight Console should appear momentarily. But before launching, the user should be certain that he or she has saved the configuration settings if desired. This can be done by pressing the "Save Configuration" button at the bottom of either tabs. The user can also restore the last saved configuration by clicking "Restore Configuration."

7.2 TSAFE Flight Console

The TSAFE Flight Console displays flight data over the geographic region previously configured in the Configuration Panel. Every flight on the console is colored either red, white, or yellow, depending on its conformance status. A flight is colored white if TSAFE determines the flight to be conforming to its flight plan, red if it is determined to be not conforming, and yellow if TSAFE has not received a complete flight plan for that flight. Since flight plans are generally received at least a half-hour prior to take off, TSAFE typically needs to be up and running at least that long before it displays any non-yellow flights.

By selecting "Conformance Monitoring . . ." from the "Parameters" menu, a window appears allowing the user to fully configure the parameters used in the calculation of the conformance status. These adjustable settings include the selection of the properties involved, the value of property deviation thresholds, and the value of a residual threshold. The reciprocal of the deviation thresholds are taken to be the weighting factors in the residual generation formula. If the residual is equal to or greater than the residual threshold, TSAFE will deem the flight to not be conforming.

The other data displayed by the Console is color-coded as well. Flight routes are displayed in blue, fixes in green, and trajectories in purple. The lookahead time for trajectory synthesis can be adjusted by selecting "Trajectory Synthesis" from the "Parameters" menu. Naturally, the greater the lookahead time, the longer the trajectories will appear on screen. The scrollable list along the left hand side of the map includes all the flights currently within the bounds of the map. Depending on the current display settings, the flights that are selected in this list may or may not be displayed differently on the flight panel. This list enables multiple selection, so multiple flights may be selected at once. The panel has the ability to display various combinations of flights, routes, trajectories, and fixes. For instance, the panel can be configured to display only those flights which are selected or only the trajectories of flights which are deemed conforming to their flight plans. Every combination of display options are available to the user via the "Show" menu.



Lateral Threshold:	10000.0	meters	On
Vertical Theshold:	609.7561	meters	Off
Angular Threshold:	0.5	radians	On
Speed Threshold:	100.0	meters / second	Off
Residual Threshold:	1.0		

Figure 7-3: TSAFE Flight Console.

Chapter 8

Data Analysis

Having completed the TSAFE prototype, I decided to use TSAFE to perform data analysis on the ETMS feed, and thereby gauge the accuracy of the data we have been utilizing. On advice from Tom Reynolds, I decided to estimate the percentage of flight plans that are "correct," i.e. they accurately reflect the pilot's intent.

So how do we determine if a flight plan is correct? Naturally, if a flight is strictly conforming to its flight plan, one would assume the plan to be correct. However, even if a flight is blundering with respect to a plan, one would still assume the plan to be correct, so long as the blundering is not excessive. That is, a flight would have to be "way off" its flight plan before one would deem the flight plan to simply be an inaccurate reflection of the pilot's intent.

I will now specifically quantify, somewhat arbitrarily, what I mean by "way off." In the x-y plane, this will mean a distance greater than ten nautical miles from the flight route. In the vertical dimension, it will mean a straying greater than 2000 feet from the assigned altitude. Since a deviation outside these limits would constitute such an egregious deviation from the flight plan, my assumption is that these do not represent blunders, but are indicative of inaccurate flight plans.

In the analysis, I examined 2000 flights over the the state of Massachusetts. Of these, I marked which ones deviated more than ten nautical miles from their flight route, which ones deviated more than 2000 feet from their assigned altitude, and which did both. The results appear in the table below.

	10nm Laterally	2000ft Vertically	Both
Deviated $\leq$	1057	117	74
Deviated $>$	943	1883	1926
Total	2000	2000	2000
% Accuracy	52.85%	5.85%	3.70%

Table 8.1: Extreme Flight Deviations from the Flight Plan.

The results were informative and somewhat disturbing. Only a slight majority of flights were found to always be within the wide lateral margins of deviation and nearly all were strayed from their assigned altitude. It is difficult to pinpoint the exact cause of these divergences. Perhaps many incorrect flight plans were filed, or maybe the controllers cleared flights for flight plan amendments that were never entered into the host computer, or perhaps a number of pilots decided to simply ignore parts of their plans. Separate analysis of commercial and non-commercial flights yielded nearly identical percentages. It is unclear whether another feed source other than ETMS would produce the same results.

Perhaps one conclusion we could draw is that the probability of an extreme lateral deviation and the probability of an extreme vertical deviation are not independent. If they were, we would expect the percentage of flights that do both to be 52.85% $\times$ 5.85% = 3.09%. But the percentage we empirically found was 3.70%. This would tend to indicate that a flight plan that is inaccurate in the lateral dimension is more likely to be inaccurate in the vertical dimension, and vice versa.

Though it is hard to draw concrete conclusions from the results, it is clear that further data analysis would be instructive. This is one of the several areas of future research discussed in the Chapter 9.

Chapter 9

Future Work

Constructing a working version of TSAFE has opened up many avenues for future research. In particular, there is much potential for applicability of TSAFE in the fields of software design, artificial intelligence, and of course, air traffic control.

9.1 Software Design Potential

Although the TSAFE prototype is a simple and well-structured piece of software, there is still much work to be done to verify it as a Trusted Computing Base. To do so, one would need to construct an argument that TSAFE meets its specifications, or at least guarantee it satisfies some minimal set of safety-critical properties. This I believe to be a serious software design challenge.

The modeling language Alloy, which was discussed briefly in Section 3.3, could prove useful in making such an argument. One could build an abstract Alloy model of the TSAFE code and check assertions that the code fulfills certain desired properties. The success of such an endeavor would have widespread, positive repercussions for the field of code conformance. I will try to make some headway on this problem in the near future.

9.2 Artificial Intelligence Potential

The greatest potential for applicability of AI techniques with TSAFE seems to lie in the task of conformance monitoring. It seems quite plausible that one could employ methods of machine learning to "learn" the way flights follow their flight plans (or how they stray from them) and to "teach" a conformance monitor what it means for a flight to be blundering.

For example, density estimation could be employed to model the distribution of flights according to the size and nature of their deviations from the flight plans. This model could then be analyzed to detect clusters of flights with common deviations. And perhaps further investigation could reveal that certain deviations are correlated with particular causes. For example, one might discover that all flights within a particular cluster have all had inaccurate flight plans filed on their behalf, or that most of those in another cluster were bypassing a fix in their flight route without authorization. If all flights could be classified as belonging to a particular cluster or clusters of behavior, we would have an intelligent and robust framework for intent inference and blunder detection.

Another, more classical, idea involves conventional classification techniques. Theoretically, if one could build a large training set of flights, each of which is accurately marked as conforming or blundering, then one could teach a conformance monitor the optimal weighting factors for residual generation, and thereby train it to correctly identify conforming and nonconforming aircraft. But seeing as its often difficult for even humans to determine a flight's conformance state, it is the construction of this training set that may prove the most difficult part of this exercise. Perhaps classification techniques could also be employed to learn how flights make turns and level off.

9.3 Air Traffic Control Potential

There are a number of ways air traffic control researchers could benefit from future work with TSAFE. For one, they could use the tool to experiment with different parameters for residual generation and perform extensive data analysis to find the optimal settings for this calculation. They could also test the viability of TSAFE's current algorithms by putting them to the test in various real-world scenarios. Lastly, they could conceive and propose conflict detection and avoidance algorithms to be added as components to the TSAFE engine module.

Appendix A

Glossary

Airway. A predefined sequence of fixes typically found in the middle of a flight route.

Conflict. A predicted violation of separation, an instance in which two flights will come within a minimum regulatory distance from one another.

Controller. A person responsible for maintaining a safe and efficient flow of aircraft.

Fix. A two-dimensional geographic position on the Earth's surface.

Flight Plan. Information about a flight and the aircraft that will fly it. It includes a flight route, cruise altitude, cruise speed, as well as the type and equipage of the aircraft.

Flight Route. A sequence of fixes, which when connected, form a two-dimensional path from a point of departure to a point of arrival.

Navaid. Navigational aid. A device located on the Earth's surface in relation to which a flight's position may be described.

Sector. A 3-dimensional region of airspace, the aircraft in which are managed by a team of air traffic controllers.

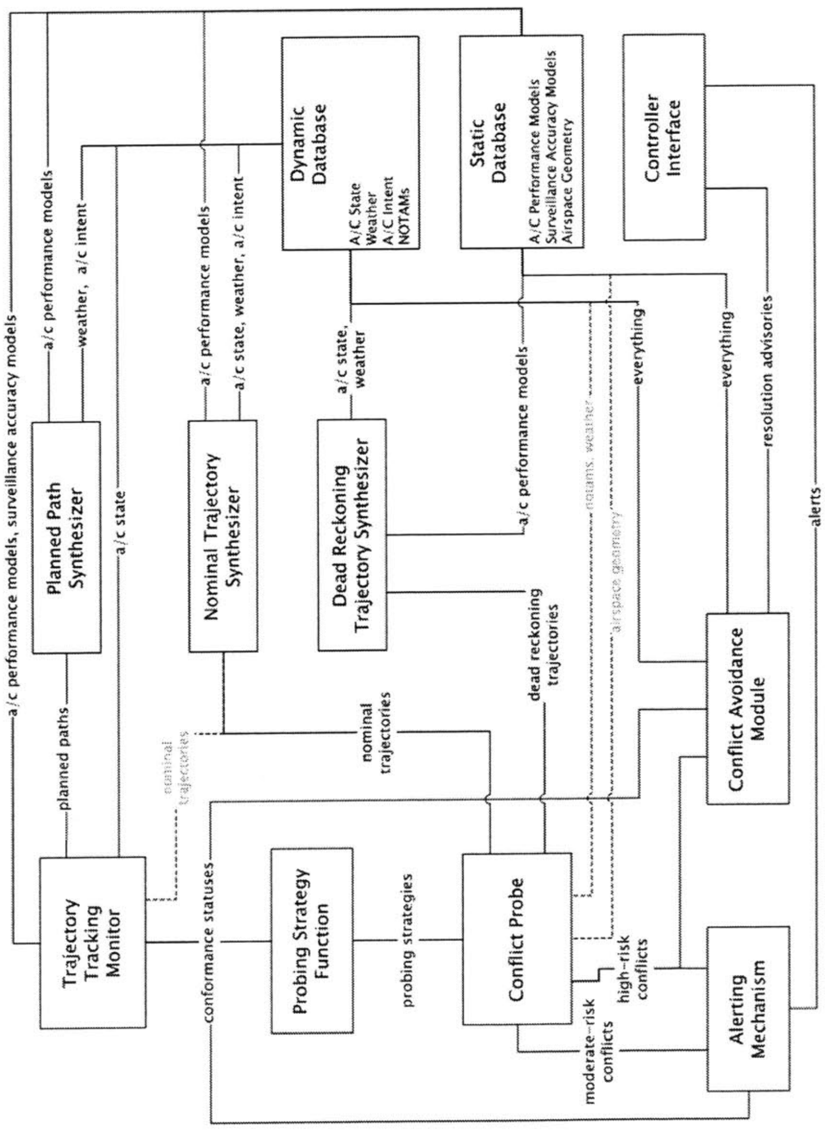
SID. Standard Instrument Departure. A predefined route for departing an airport, typically found at the beginning of a flight route.

STAR. Standard Terminal Arrival Route. A predefined route for approaching an airport, typically found at the end of a flight route.

Trajectory. A four-dimensional path (latitude, longitude, altitude, and time components) along which an aircraft may fly.

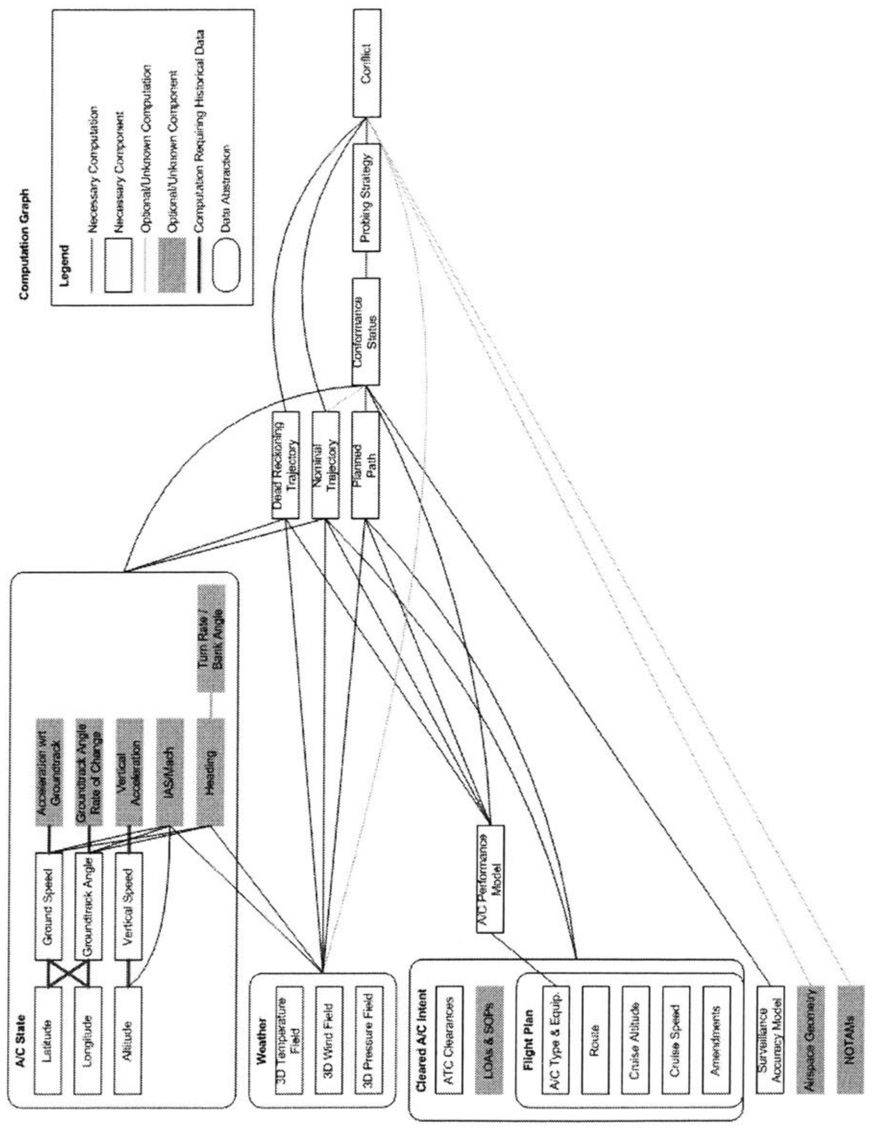
Appendix B

Data Flow Diagram



Appendix C

Computation Graph



Appendix D

Abstract Allow Model

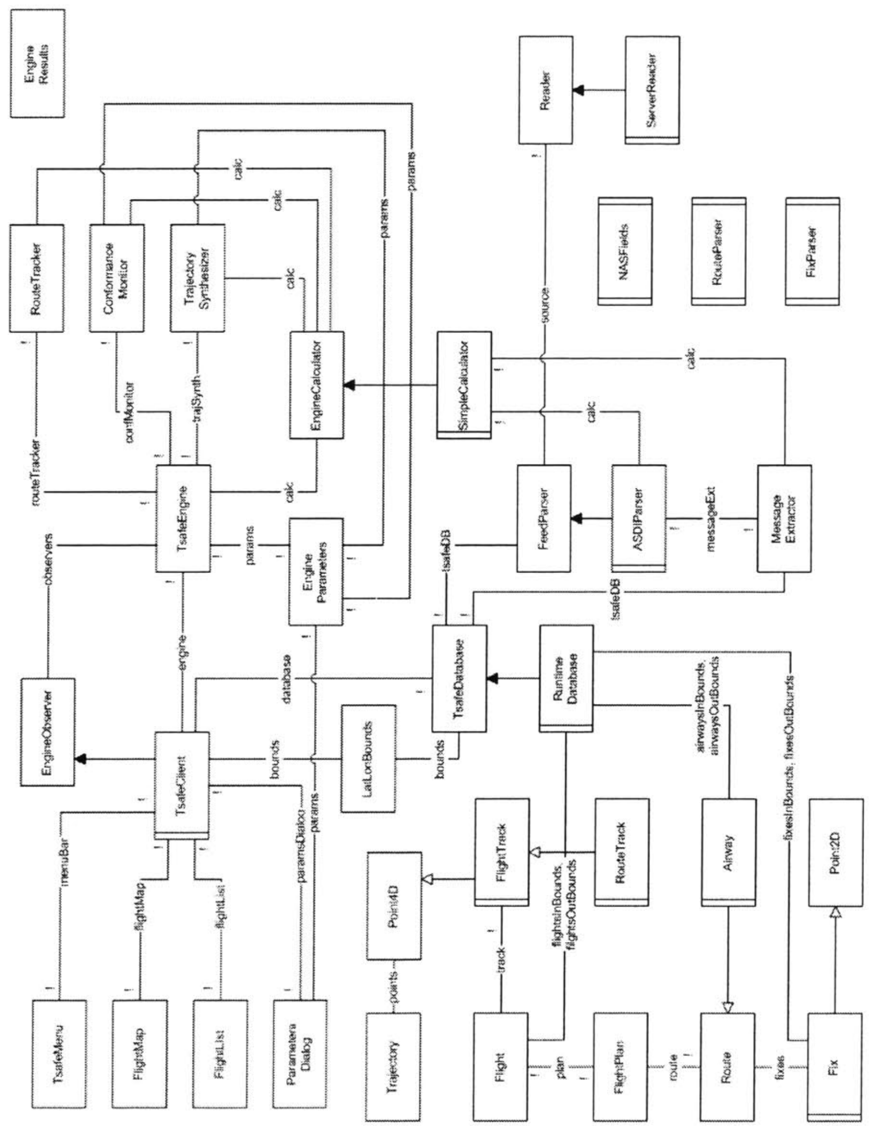
```
module tsafe

\\ The 'ord' module provides linear ordering of elements
open models/ord
fun OrdNexts_ [t] (elem: t): set t { result = elem.*(Ord[t].next) }
fun OrdPrevs_ [t] (elem: t): set t { result = elem.*(Ord[t].prev) }

\\ Construct a linear ordering of instances in time
\\ TimeMin is the first element in this ordering
\\ TimeMax is the last element in this ordering
sig Time {}
static sig TimeMin extends Time {}{this = Ord[Time].first}
static sig TimeMax extends Time {}{this = Ord[Time].last}


\\ A 3-dimensional point in space
sig Point {}

\\ Trajectories map instances in time to 3-D points in space
sig Trajectory {
    location: Time ->! Point
}

\\ Each heading defines a chain (a totally ordered subset)
\\ over the set of points
sig Heading {
    points: set Point,
    next: points ?->? points,
    prev: points ?->? points
}{
\\ next is the inverse of prev
    all p1, p2: Point | p1->p2 in next <=> p2->p1 in prev
    \\ all the points in the heading are reachable
    some p: points | p.*next = points
    \\ heading is not cyclic
    no p: points | p in p.^next
}

\\ Two headings are either coincident in the same direction, coincident
\\ in the opposite direction, or they cross at at most one point
fact {
    all h1, h2: Heading |
        h1.next = h2.next || h1.next = h2.prev ||
        (sole h1.points & h2.points)
}

\\ A flight in the airspace
sig Flight {}

\\ A state of the airspace.
sig State {
\\ the time at which this state occurs
    time0: Time,
    \\ position of each flight in the airspace
    position: Flight ->! Point,
    \\ heading of each flight in the airspace
    heading: Flight ->! Heading,
    \\ predicted trajectory for each flight
    probingTraj: Flight ->! Trajectory,
    \\ conflicts in the airspace
    conflicts: Flight -> Flight
}{
\\ Synthesize a dead reckoning trajectory for each flight
    all f: Flight | this..drTrajSynth(f)
    \\ Probe for all conflicts in this state
    this..conflictProbe(conflicts)
}

\\ Returns true if tMin is the minimum Time in the set times
\\ in which t1 and t2 cross. Returns false otherwise
fun minTimeToCross (tMin: Time, t1, t2: Trajectory, times: set Time) {
    t1.location[tMin] = t2.location[tMin]
    all m: times | t1.location[m] = t2.location[m] => OrdLE(tMin, m)
}

\\ Synthesizes a dead reckoning trajectory for the flight
fun State::drTrajSynth (f: Flight) {
    f.this::probingTraj.location[this.time0] = f.this::position
```

```
  all m: OrdNexts_(this.time0) |
        (f.this::probingTraj.location[m]).(f.this::heading.next) =
            f.this::probingTraj.location[OrdNext(m)]
}

\\ Probes for all conflicts amongst flights in this state
fun State::conflictProbe (conflicts: Flight -> Flight) {
    all f1, f2: Flight | f2 in f1.conflicts <=>
        {f1 != f2 &&
         some tMin: Time |
            minTimeToCross(tMin, f1.this::probingTraj,
                           f2.this::probingTraj, OrdNexts_(this.time0))
        }
}

\\ Computes the next state assuming flights stay on their current heading
fun nextState(s, s': State) {
    s'.time0 = OrdNext(s.time0)
    all f: Flight | f.s'::position =
        (f.s::position).(f.s::heading.next)
}

\\ Asserts that if flight1 is in conflict with flight2,
\\ then flight2 is in conflict with flight1
assert conflictsCommunitive {
    all s: State | all f1, f2: Flight |
        f1 in f2.s::conflicts <=> f2 in f1.s::conflicts
}

\\ Asserts that the conflicts in this state are the same as the
\\ conflicts in the next state
assert conflictsSame {
    all s, s': State |
        nextState(s, s') => s.conflicts = s'.conflicts
}

\\ Asserts that the conflicts in this state are the same as the
\\ conflicts in the next state given that a flight's trajectory
\\ remains the same
assert onProbTrajConflictsSame {
    all s, s': State |
        s'.probingTraj = s.probingTraj && nextState(s, s') =>
            s.conflicts = s'.conflicts
}
```

```
\\ This assertion fails because flights are allowed
\\ to move off their current trajectory
check conflictsSame for 5

\\ This assertion holds because flights are required
\\ to continue on their current trajectory
check onProbTrajConflictsSame for 5
```

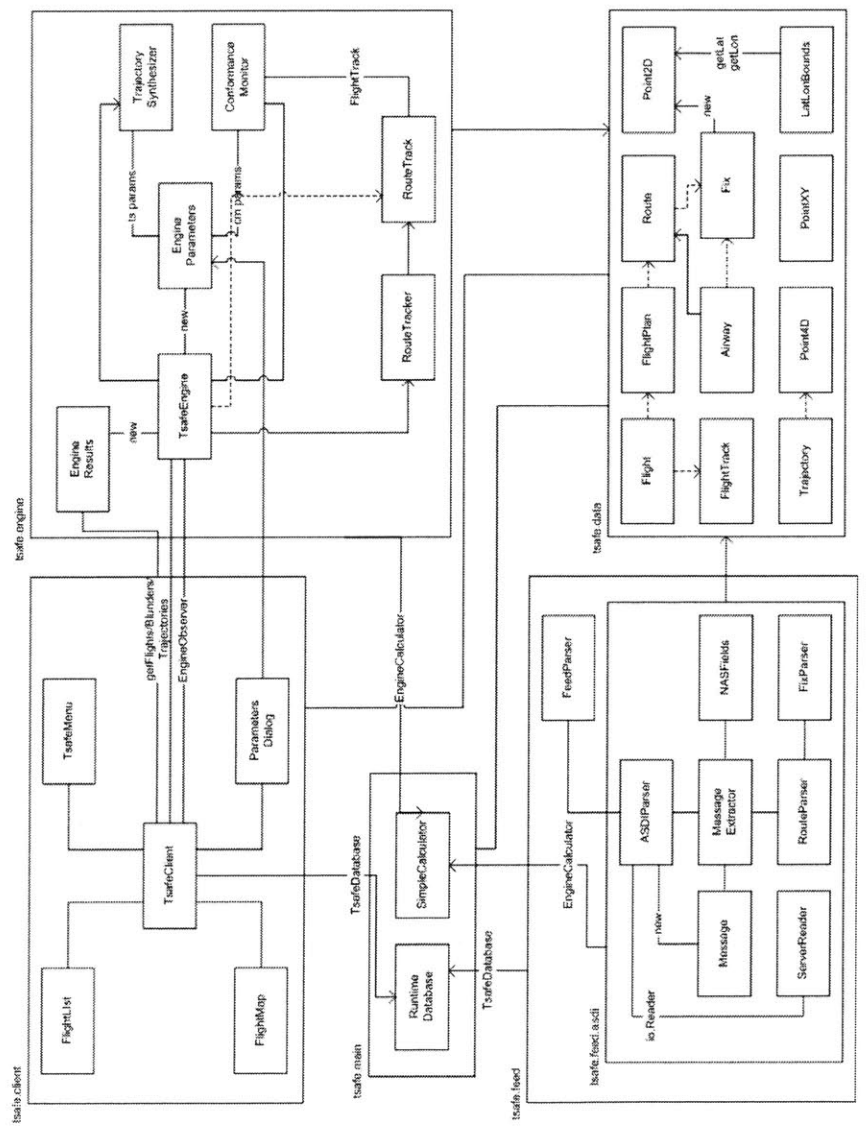
Appendix E

Code Object Model



Appendix F

Modular Dependency Diagram



Bibliography

[1] Route conversion and posting. National airspace system en route configuration management document, Operational Support, National En Route Automation Division, AOS300, Federal Aviation Administration, November 1997. NAS–MD-312.

[2] Aircraft situation display to industry: Functional description and interface control document. Technical report, Volpe Center, August 2002.

[3] Heinz Erzberger. The automated airspace concept. *4th USA/Europe Air Traffic Management R&D Seminar*, December 2001.

[4] Enhanced traffic management system. Available at http://www.fly.faa.gov/Information/EMTS/etms.html.

[5] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley, Reading, Massachusetts, 1995.

[6] Daniel N. Jackson. Dependences and decoupling. *6.170 Lecture Notes*, September 2002.

[7] Tom G. Reynolds and R. John Hansman. Conformance monitoring approaches in current and future air traffic control environments. *21st Digital Avionics Systems Conference*, October 2002.

[8] David C. Zhang. Collaborative arrival planner: Its design and analysis using object modelling. Master's thesis, Massachusetts Institute of Technology, May 2000.