DELFT UNIVERSITY OF TECHNOLOGY
Faculty of Electrical Engineering
Telecommunications and Traffic
Control Systems Group

Navigation Simulation (NAVSIM) for the Basic Research Simulator (BARESIM)

PART I - NAVSIM: towards a theoretical block diagram model
PART II - NAVSIM: developing a test-environment for the model

Thierry O.A. Larenas Linnemann

Graduation Thesis, final version
76 pages
29 September, 1993

Professor: Dr. Ir. D. van Willigen
Supervisor: Ir^2. E. Theunissen
Assignment No.: A-455
Period: September 1992 - July 1993

Abstract

With the increase of air traffic density, especially in the airport approach area, the need for aircraft navigation systems with a higher accuracy grows. Cost-effective test-facilities, like the Basic Research Simulator (BARESIM), are required for evaluation of these new and already-existing systems. In this report a navigation simulation (NAVSIM) model has been developed, able to fulfil the needs for simple and complex navigation system testing. The NAVSIM model consists of various modules suited to be implemented in an Object-Oriented Programming (OOP) test-environment. This paves the way for future programmers in charge of the implementation of a specific module.

Key words: Basic Research Simulator, Navigation Simulation, module data flow, block diagram model, test-environment, Object-Oriented Programming

Preface

The last 10 months of my studies at Delft University of Technology, I spent at the Telecommunications and Traffic Control Systems Group of the Faculty of Electrical Engineering. During this period I was privileged to be a part of, what undoubtfully must be labelled, the brightest and yet craziest bunch of fellow-students and supervisors. All these guys must be thanked for both their physical (I lived the unforgettable experiences of two removals) and mental support. I owe a special 'thank-you' to: David for teaching me the right plural form of antenna: antennae (though I overruled stubbornly with antennas) and checking this report for syntax and spelling erors; Dignus for teaching me to work till late(r) and providing me with Bitter Lemon; Guus for encouraging my special interest in Cindy C; Robert for all his exciting bed-time stories (or should I say trips?); Arjen for installing Windows and censuring my pretty babes; Peter: yes, despite my childish behaviour, I'm feeling fine; Arnoud: is there any feature of MS-DOS you don't know about? ; Gino for his support during the beginning of my work; Bas for correcting me, even if I was right; Lambert for his clear helicopter view of things; Remco Kroes, for being such a great GIF supplier (I won't get the grade you want me to); Joan Heijns, for making me such great diners; Dr. Eric Tetterroo for keeping me up-to-date in the computer world; Ir. Edward: we had so much fun with Coreldraw, didn't we? Should I scan some more?; Ir. Richard for his useful comments on my intermediate presentation; Dr. Ir. Mike Braasch: who said GPS signal modelling was any different from NAVSIM modelling?; Ir. Tycho for his help during my first C++ experiences; Ir. Han, the real Disk Doctor; Jaap, for lending me your C++ books; Cor, eventually I'll break your TETRIS record;

A very special thanks to my supervisor Ir. Erik Theunissen for the interesting assignment and all the free-space he gave me, and of course teaching me about the COBRA manoeuvre (who said aircraft freaks like us are psyched?). Last but not least I am very grateful to my professor Dr. Ir. van Willigen, whose dedication to all of our work is incomparable to anything alike: thanks for the encouragement.

The author.

Summary

With the exponential growth of air traffic, especially in the airport approach area, the need for new and more accurate aircraft navigation systems increases. Development and extensive testing of these systems is only possible by means of cost-effective test facilities. This is where flight simulators come in.

The Basic Research Simulator (BARESIM) is a six-degrees-of freedom flight simulator, currently developed at Delft University of Technology. The planned system configuration is such that no navigation errors are taken into account yet. That is, the *true aircraft state* (as generated by the host-computer) is shown on the pilot's displays, whereas in reality only a *best-estimated state* is available. This estimation is calculated by the on-board navigation systems. So, for a more realistic simulation, the navigation system's restrictions should be modeled. A simple solution might be the addition of (gaussian) noise to the true aircraft state; however, for the validation of future complex navigation systems a better method is required.

A modular approach towards Navigation Simulation (NAVSIM) is suggested. A block diagram model has been developed, in which each 'activity' module represents a specific part of the whole navigation system. The input, output, control, and mechanism data of each module has been defined, represented by arrow connections in the block diagram. The overall result is a flexible and yet detailed model, suitable for simple and complex navigation simulations. This has been proved by an evaluation of the NAVSIM diagram model with two existing models: a reflection model and a Differential Global Positioning (DGPS) signal model. Evaluation was successful: both models were covered by NAVSIM.

The theoretical NAVSIM model has been implemented in an Object-Oriented Programming (OOP) environment. Each module corresponds to a self-supporting Class in the OOP-language C++. The resulting program should be a base for future yet-to-be-assigned programmers, in charge of further implementation of NAVSIM. The NAVSIM test-environment should be extended with a User Friendly Interface, facilitating simulations under varying test conditions.

Contents

Abbreviations

a/c	aircraft
ADF	Automatic Direction Finder
AFCS	Aircraft Flight Control System
ASCII	American Standard Code for Information Interchange
ATIS	Automatic Terminal Information Service
BARESIM	Basic Research Simulator
C/A	Coarse Acquisition
CAS	Collision Avoidance System
CAS	Calibrated Air Speed
CDU	Control Display Unit
CRT	Cathode Ray Tube
DADC	Digital Air Data Computer
DGPS	Differential Global Positioning System
DME	Distance Measurement equipment
DOD	Department Of Defense
DPSK	Differential Phase Shift Keying
e	humidity
ED	Emission Delay
EFIS	Electronic Flight Instrument System
EFISCP	Electronic Flight Instrument System Control Panel
EICAS	Engine Indication & Crew Alert System
ETA	Estimated Time of Arrival
f_{cr}	critical frequency
FMS	Flight Management System
ft	foot (1 ft $\approx$ 30.5 cm)
GPS	Global Positioning System
GRI	Group Repetition Interval
ICAO	International Civil Aviation Organisation
ILS	Instrument Landing System
INS	Inertial Navigation System

IRS	Inertial Reference System
IRU	Inertial Reference Unit
LF	Low Frequency
LNAV	Lateral Navigation
LORAN-C	Long Range Navigation
MF	Medium Frequency
MLS	Microwave Landing System
MPP	Most Probable Position
N	Refraction Index
N(e)	Electron Density
navaid	navigation aid
NAVSIM	Navigation Simulation
ND	Navigation Display
NDB	Non Directional Beacon
NM	Nautical Miles (1 NM $\approx$ 1852 m)
P	Precision
p	air pressure
PFD	Primary Flight Display
PRN	Pseudo Random Noise
SA	Selective Availability
SHF	Super High Frequency
STA	Scheduled Time of Arrival
T	Static Air Temperature
TAS	True Air Speed
TDI	Track Deviation Indicator
TIGA	Texas Instruments Graphic Adapter
TMCS	Thrust Management Computer System
UFI	User-Friendly Interface
UHF	Ultra High Frequency
UTC	Universal Time Coordinated
VHF	Very High Frequency
VOR	VHF Omnidirectional Range

1 Introduction

In this introduction both background and goal of the graduation research are discussed, and the structure of this report is clarified.

1.1 Background

Ground-based flight simulators are of increasing importance to aerospace industry. They provide both a cost-effective tool for flight crew training and an invaluable facility for aerospace research and development, including (future) navigation systems. The need for aircraft navigation system research increases with the exponential growth of air traffic, especially in the airport approach area. Some serious incidents have been reported involving navigation systems causing false acquisition to approach courses . A typical example of navigation system failure, are the problems encountered with a brand new

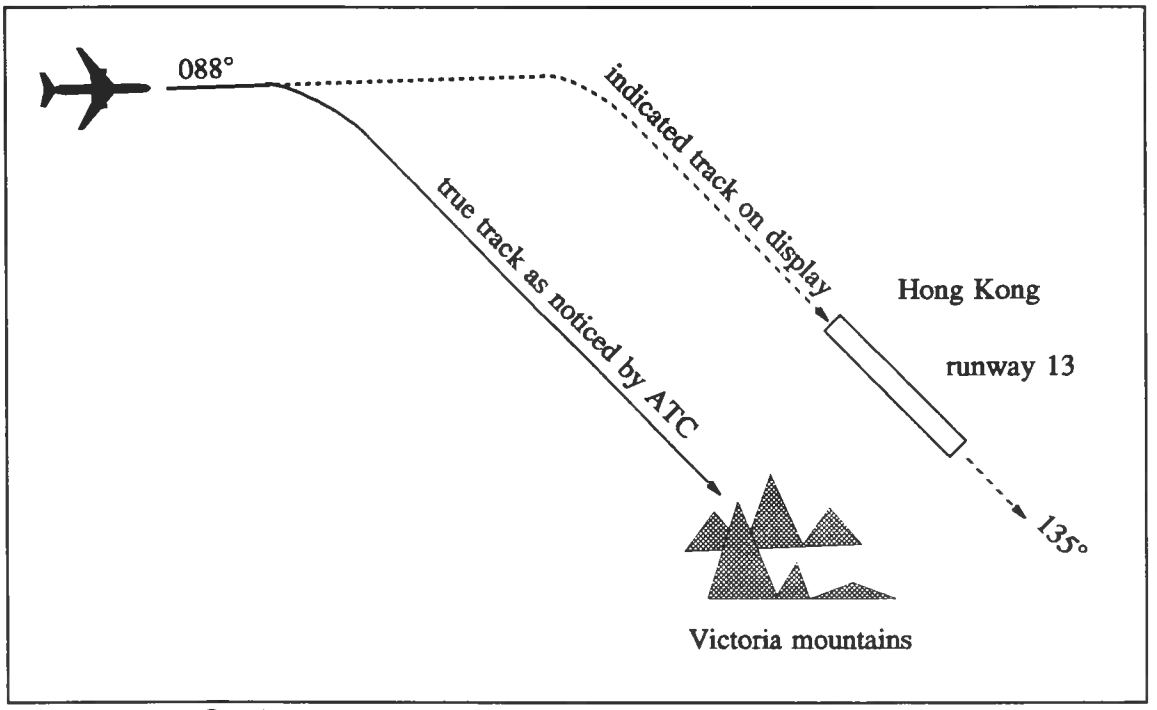

Figure 1.1 Symbolic presentation of the hazardous situation in Hong Kong: the pilot assumes he is on the right track.

Boeing 747/400 on a flight from Osaka to Hongkong (February 1990, [1]). The aircraft position shown on one of the pilot's navigation displays differed from the actual position

by more than 2 NM, due to a wrong DME[1]-update after passing the initial approach fix. As a result, the final turn was initiated 2 NM early. This fact was not detected by the flight crew because the navigation display showed an on-track-situation. Without intervention from air traffic control, glideslope (vertical guidance aid) interception might have occurred, and the aircraft might have followed the glideslope without being established on the localizer (horizontal guidance aid). Figure 1.1 shows the hazardous situation that occurred. This incident demonstrates the undeniable importance of extensive research into, and testing of already-existing and future complex navigation systems.

The Basic Research Simulator (BARESIM) is a six-degrees-of-freedom flight simulator, being developed as a joint effort within the Delft University of Technology by an inter-faculty working group. This group involves the faculties of Aerospace Engineering, Mechanical and Maritime Engineering, and Electrical Engineering. The three basic research areas are [2]:

- The dynamic behaviour and control of motion platforms
- The performance of vehicle simulations in a ground-based environment
- The human factors aspects of vehicle operation

This report focuses on the second area, especially on aircraft systems research. A simulation model of the aircraft navigation systems is investigated. The NAVigation SIMulation (NAVSIM) model will be implemented in the flight simulator. The current BARESIM configuration is given in Figure 1.2. BARESIM consists of a host computer and various subsystems. The subsystems contain their own process related computers.
A short description of the various (sub)systems is given below:

Host computer

The host computer controls the simulator environment and generates the actual aircraft state variables and instrument readings using pilot controls as input. The aircraft state variables include the exact aircraft position, velocity and attitude at a certain simulation

1 DME stands for Distance Measurement Equipment; DME is a ground-based navigation aid; in the next chapter an explanation of its function is given.

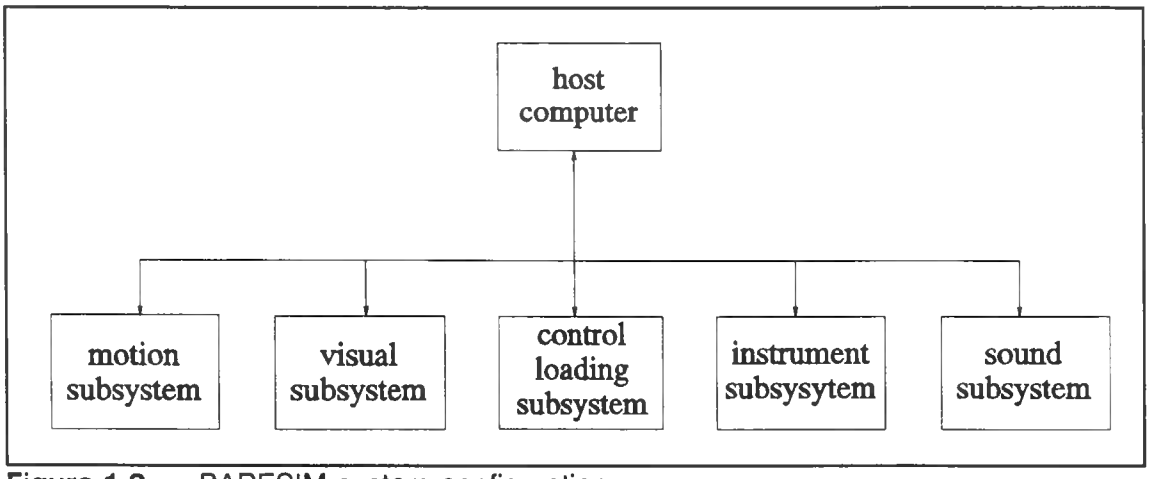

Figure 1.2 BARESIM system configuration

time. The aircraft model is generated between 30 and 60 times per second.

Motion subsystem

The six-degrees-of-freedom motion subsystem is driven by six linear hydraulic actuators and their related hardware. The six hydraulic actuators allow platform movement in x, y and z direction and rotation around x, y and z-axis (pitch, roll and yaw).

Visual subsystem

The multi-channel visual subsystem generates (by means of image generating computers) an out-of-the-window view using platform-based projectors. Probably a 75 degrees horizontal and 30 degrees vertical field of view will be simulated.

Control loading subsystem

The digital hydraulic loading subsystem provides a realistic force feel of the primary controls. There are four active channels in each pilot station : the ailerons, elevator, rudder pedals and brakes.

Instrument subsystem

All instrument readings will be presented on the Electronic Flight Instrument System (EFIS) in the simulator's cockpit. The system consists of i80X86 based PC boards, equipped with Texas Instruments Graphic Adapter (TIGA) graphic cards. A hardware

independent description has been developed by Theunissen [3] to allow for efficient aircraft display development.

Sound subsystem

This system introduces a realistic sound environment. A 16-channel digital synthesizer and sampler, controlled by a MIDI interface, generate real-life engine, wind, taxiing and instrument warning sounds.

System interconnection

A connection based on the Ethernet (IEEE 802.3) protocol will provide the high-speed environment needed for real-time operation of the simulator [4].

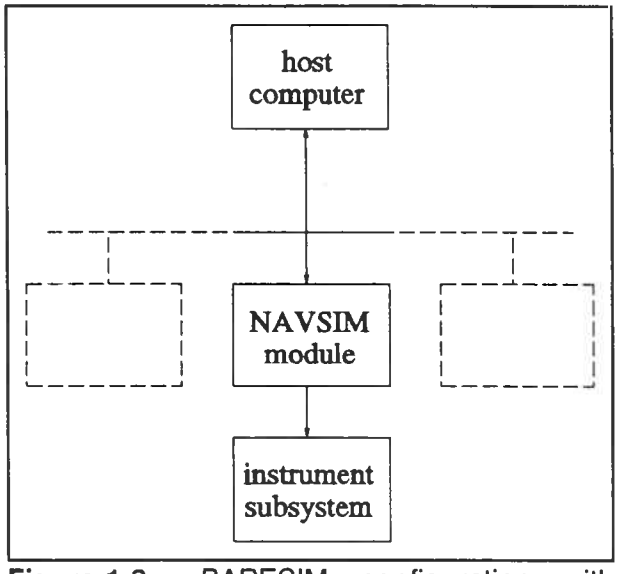

Figure 1.3 BARESIM configuration with NAVSIM module

As stated before, the host computer fulfils a key role in the system configuration. Aircraft state variables are generated in the host computer and transferred via the Ethernet system interconnection to the various subsystems. This means the real position, velocity and attitude (from now on referred to as *true aircraft state*) are fed directly to the instrument subsystem. In reality however, aircraft state variables are processed by the Flight Management System using the on-board navigation systems. In other words: not the *true aircraft state* is used by the EFIS but the *estimated aircraft state* based on the outcomes of the navigation systems. Figure 1.3 shows the NAVSIM module blocking the direct link between host computer and instrument system. The function of the NAVSIM module is comparable to the introduction of a noise generator. However, one should bear in mind that modelling of NAVSIM by just adding some (gaussian distributed) noise will not do: a more complex method is required to allow for accurate simulation of both today's 'simple' navigation systems and future, complex integrated navigation systems.

1.2 Goal

This report is divided into two parts:

The main goal of the research, described in the first part of this report, is the development of a clearly structured NAVSIM model, i.e. dividing the navigation system into smaller parts, called modules (Chapter 2), and describing the data flow between these modules (Chapter 3) by means of defining the inputs and outputs of each module (Section 3.2). Once this is done a structure diagram of NAVSIM is made, also referred to as the NAVSIM model (Section 3.2). A software engineer can develop the exact contents of a specific module of the NAVSIM model by only taking into account its in- and output variables (and the later-discussed control data). This way, he does not have to understand the whole system; only a part of it is his concern. The newly developed theoretical NAVSIM model is compared to two existing models: a multipath reflection model and a GPS (Global Positioning System) signal model (Section 3.3 and 3.4). Both models should be embodied by the NAVSIM model. In Section 3.5 the complexity and related simulation depth of the model are discussed. The first part is ended with conclusions and recommendations on the theoretical NAVSIM model (Chapter 4).

The second part of this report (Chapter 5 & 6) deals with the implementation of the NAVSIM model. The modular NAVSIM approach is suited to use with an OOP (Object-Oriented Programming) language (Section 5.1). After a general description of the basics of OOP (Section 5.2), a test-environment is suggested for NAVSIM in Borland's Turbo C++ (Section 5.3). The base structure of the NAVSIM program is discussed (Section 6.1 & 6.3), together with some sneaky C++ techniques (Section 6.2). Although these tricks can be found in C++ documentation, a great deal of effort was needed to find the correct syntax and commands for the NAVSIM program. In order to prevent future programmers from "re-inventing the wheel", example C++ code has been included in the text of Section 6.2. In a similar manner to the first part, the second part is finished with conclusions and recommendations, this time on the implementation of the NAVSIM model in the OOP-language C++ (Chapter 7).

PART I - NAVSIM: towards a theoretical block diagram model

2 Dividing the navigation system into modules

When describing a complex system, such as the navigation system of an aircraft, it is not recommendable to try to explain the system as a whole at once. A better approach is dividing the system in smaller (digestible) parts. One possible method is the definition of modules to describe the navigation system. Each module represents a part of the system and can be developed adequately by a specific software engineer.

2.1 Introduction

The navigation modules will be defined during the following short analysis of the positioning and navigation process of an aircraft. The module names are underlined.

An electro-magnetic signal is transmitted by a navigation aid (navaid). These transmitters include both 'classic' NDB, VOR/DME, ILS stations and any future navaids, like MLS and GPS. The signal is distorted on its way to the aircraft due to atmospheric conditions (e.g. raindrops). The distorted signal reaches the aircraft antenna accompanied by multipath signals. Multipath signals consist of reflected and diffracted waves and are caused by the environment (e.g. mountains, rivers, and buildings) in the vicinity of the aircraft. The composed navaid signal (direct + multipath) is either amplified or attenuated by the aircraft antenna, as a function of the antenna position, orientation and radiation pattern. The raw antenna signal is processed by the software receiver, resulting in an estimated aircraft position. This position can be compared to position parameters generated by other state-determination systems, such as the Inertial Navigation System (INS). The Digital Air Data Computer (DADC) provides air data, like air speed and barometric height coming from the aircraft sensors (e.g. pitot static tube). This air data is needed in the positioning process (e.g. to calculate ground range out of slant range).

The best estimated aircraft position is transferred from the software receiver to the 4D-navigation and guidance system. This position is compared to the desired flight path as defined in the flight plan, inserted by the pilot before taking off. The results are steering

commands either by the pilot or the Flight Management System (FMS, auto-pilot), to direct the aircraft along the desired path.

A navigation data base contains files with so-called static data, loaded at the beginning of a flight. This data includes navaid information on positions and frequencies needed for the positioning process, as well as 'rough' map background data for the navigation displays (position of waypoints, airports etc.). All air, aircraft and map background data are gathered and prepared for presentation in the 4D-navigation and guidance system. This system is directly linked to the presentation part of the navigation system in the cockpit. Presentation data is shown on the EFIS displays such as the Primary Flight Display (PFD) and Navigation Display (ND).

The modules involved are:

- transmitters
- atmospheric conditions
- environment
- aircraft antennas
- software receiver
- other state-determination systems
- Digital Air Data Computer (DADC)
- flight plan
- 4D-navigation and guidance system
- navigation data base
- presentation

One additional module will be specified. It cannot be found in the above description because it has to do with the actual simulation of the real world. Therefore we define it as:

- simulation of real-world

In the next section the function of this module will become clear.
Next, each module is briefly described by means of a summary of its characteristics.

2.2 Transmitters

The transmitters are the radio-navigation aids used by the airborne navigation systems to determine the position[1] of the aircraft. These transmitters can either be ground-based (e.g. VOR/DME, ILS, MLS) or space-based (GPS). In the next sections the most common and widely used navaids are briefly discussed (see ref.[5], [6], [7] and [8]).

2.2.1 Non Directional Beacon (NDB)

The NDB is a ground-based station with an omni-directional antenna. Frequencies used are within the 200-1750 kHz band. The NDB is always used in combination with the airborne Automatic Direction Finder (ADF). The ADF determines the angle between the aircraft's vertical axis and the radial towards the NDB. Nowadays the *locator* is the only NDB widely used for approach navigation up to 15 NM. Accuracy is between 2° and 5°[2].

2.2.2 VHF Omnidirectional Range (VOR) and Distance Measuring Equipment (DME)

VOR stations are ground-based and transmit in the 108-118 MHz band. Actually two signals are transmitted: an azimuth independent reference signal and an azimuth dependent signal. Comparison of the phases of these two signals gives the angle between North and the radial between aircraft and transmitter. VOR stations are of multiple use:

- a pilot can use the direction of the beacon for positioning purposes.
- a pilot can fly to or from the beacon via a radial; the Track Deviation Indicator (TDI) assists the pilot by means of the follow-the-needle principle.
- Automatic Terminal Information Service (ATIS) data can be broadcasted by means of a VOR data-link.

1 In the future the aircraft's velocity and attitude (GPS research project OHIO University) may be calculated with the help of navaid signals.

2 In general a 95% interval (2σ) is used for accuracy specification. This during 5% of the time inaccuracies larger than the indicated values are met.

Accuracies of 2° are usually met. VOR stations provide the aircraft with bearing information.

DME stations are used for distance measuring. A carrier of approximately 1 GHz is modulated with AM bursts. The DME system consists of a ground-based transponder and an airborne interrogator. The interrogator transmits pairs of AM bursts and the ground-based transponder returns the bursts after a specified delay time. The burst delay time contains information about the distance (slant-range) between transponder and interrogator. Errors of 0.5 NM are common.

Both VOR and DME stations use frequencies with line-of-sight wave properties. That is, these navigation aids can only be used within 250 NM for en-route purposes and within 20 NM for the approach.

2.2.3 Instrument Landing System (ILS)

The ILS navigation aid is the most widely used navaid during the final approach phase of the flight. The system consists of a localizer, a glide slope and markers.

The localizer provides lateral guidance for approaching aircraft (see Figure 2.1). A 90 Hz and 150 Hz signal are modulated on a 110 MHz carrier. The 90 Hz signal's main lobe is slightly to the left of the centre-line of the runway (seen from the approaching aircraft). The 150 Hz signal has its main lobe to the right of the runway centre-line. The airborne ILS receiver compares the modulation depth of both signals. The relative amplitude difference between both signals indicates whether the aircraft is either too much to the left (stronger 90 Hz signal) or to the right (stronger 150 Hz signal). The localizer is positioned at the end of the runway.

The glide slope antenna provides the pilot with information about a 3° glide slope path in the vertical plane (see Figure 2.2). The basic idea is the same as with the localizer: the 150 Hz signal has its main lobe under the 3° glide path; the 90 Hz signal above the 3° glide path. On the glide path both signals are equally strong. The glide slope antenna is situated at the beginning of the runway.

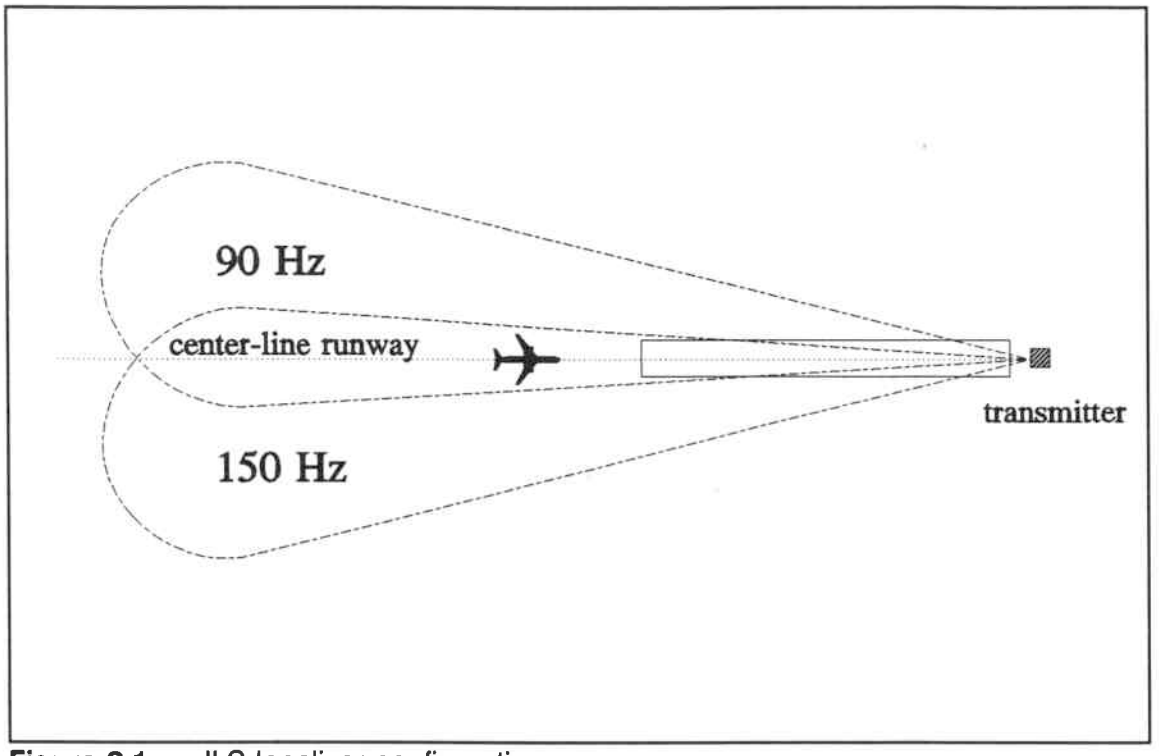

Figure 2.1 ILS localizer configuration

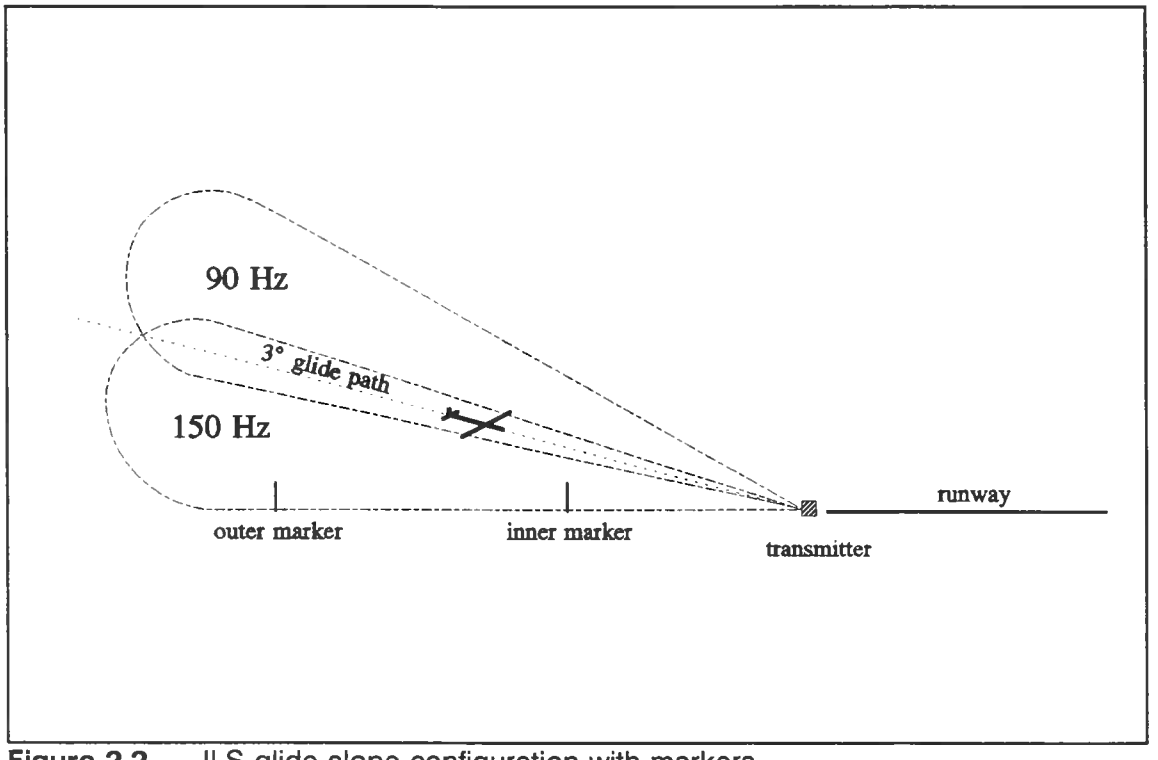

Figure 2.2 ILS glide slope configuration with markers

The outer, middle and inner marker (at respectively 4 NM, 3500 ft, and 100-150 ft from the runway) provide the third dimension needed for approach navigation. The pilot is

notified the moment he passes a marker by an illuminating coloured lamp in the cockpit. This enables him to estimate his distance to the runway. Each marker has its specific modulation frequency and (morse code) identification. A carrier frequency of 75 MHz is used.
DME can be used to give additional information on the distance-to-go to the runway.

2.2.4 Microwave Landing System (MLS)

The MLS is expected to be the most important approach guidance system in the near future. Unlike the ILS, MLS allows for a infinite number of approach paths. The carrier frequency used is around 5 GHz. The coverage area of MLS has a fan-shaped form (see Figure 2.3). Two scan beams can be distinguished:

An azimuth scan beam is used for lateral guidance. This beam is narrow in the horizontal plain (0.5°-3°) and wide in vertical plain (15°). The azimuth area between -40° and 40° is scanned. An aircraft is twice hit: once during the so-called TO-scan and a second time during the FRO-scan. The time difference between the moments the aircraft is hit by the beam is a linear function of the angle between the aircraft path and the runway centre-line.

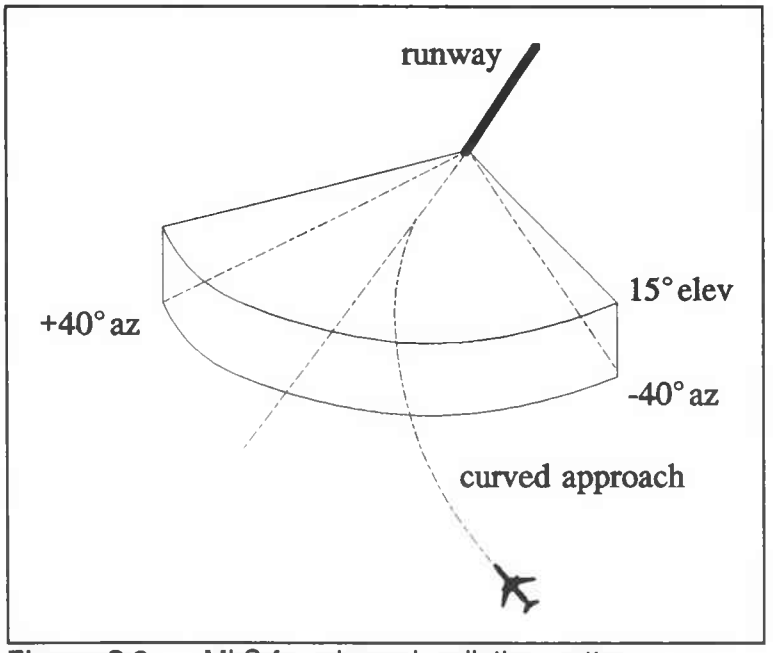

Figure 2.3 MLS fan-shaped radiation pattern.

The elevation scan beam operates in the same way as the azimuth. This beam is wide in the horizontal plane (80°) and narrow (0.5°-3°) in the vertical plane. An area between 0 and 15° is scanned during a TO- and FRO-scan.
MLS is normally used in combination with DME. Two types of DME are in use: DME/N for normal

distance measuring and DME/P for precision purposes.

The last tool needed for a minimal MLS configuration is data-link equipment. One distinguishes data directly involved in the positioning process (basic data words), like runway status or the exact position of the MLS elements, and auxiliary data (auxiliary data words), like meteorological data. Data is modulated on the carrier frequency by means of Differential Phase Shift Keying (DPSK). The technique used to distinguish between the various signals for positioning purposes and the basic/auxiliary data, is time division multiplexing: position information (azimuth and elevation) and data are transmitted during different fixed time slots. Pre-ambles are added to each time slot to inform the aircraft's receiver about the exact nature of the information to be expected during the next slot.

2.2.5 Global Positioning System (GPS)

In the near future GPS is expected to be one of the most important systems both for en-route and approach aircraft navigation[1]. It will probably replace most of the existing navaids (NDB, VOR/DME, LORAN-C). The general GPS configuration is given in Figure 2.4.
GPS consists of three so-called system segments [9]:

space segment

The space segment contains 21 satellites (+ 3 spares). The constellation is such that 5 or 6 satellites can be spotted from any place on earth. The satellites orbit at a height of 22.240 km around the earth. A complete orbit is accomplished in approximately 12 hours. Each satellite transmits a 1.58 GHz (L_1-band) and a 1.23 GHz (L_2-band) carrier signal. These signals are modulated by a Pseudo Random Noise (PRN) code and Data information. PRN codes differ from satellite to satellite and therefore allow for satellite identification. The complexity of the PRN depends on the user of the system: for military purposes a Precision (P) PRN code is available; civil customers can only use the Coarse Acquisition (C/A) Code.

1 Actually, GPS is suited for a variety of other navigation applications, including harbour precision approaches (DGPS) for (large) ships and travel pilot facilities for trucks and cars.

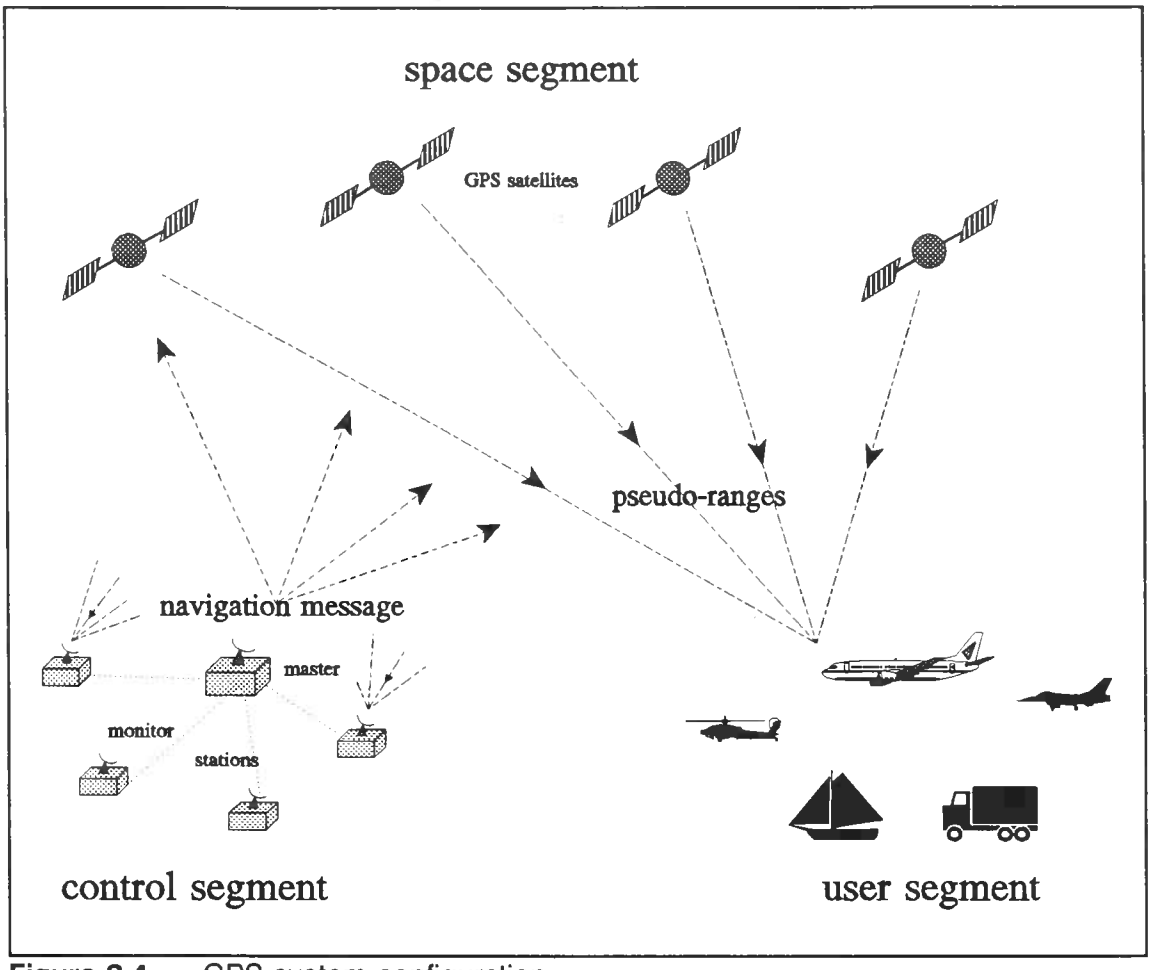

Figure 2.4 GPS system configuration

control segment

The control segment consists of 4 monitor stations and 1 master control station, all ground-based. Data concerning satellite ephemeris, satellite clock drift, ionospheric errors etc. is gathered and transmitted to the satellites. The satellites process this information and add it to the data sent to the user. Hence the control segment determines the navigation message broadcasted by the satellites.

user segment

The user segment consists of receivers containing the same PRN codes used by the transmitters. The PRN code of the receiver is shifted in time until it perfectly matches the incoming satellite PRN-code. The time shift is an indication of the delay time of the signal from satellite to receiver. Multiplication of this delay time by the propagation velocity of the signal gives a pseudo-range. The qualification pseudo is added because it is not the

exact range that is determined: a "constant" distance error is made. This is due to the constant clock synchronisation error between the receiver and the GPS satellites time. A user needs at least 4 satellites to get a position fix. Three satellites are needed to determine a three dimensional position (with the earth center as reference). The fourth satellite is needed to correct for the afore-mentioned time error made by the receiver's (relatively cheap) clock.
The PRN code is used to determine the pseudo-range toward a satellite. Once this code is extracted from the carrier signal, the remaining navigation message data becomes available. A frame of 1500 bits is broadcasted by each satellite at a rate of 50 bps. Each frame incorporates 5 subframes of 300 bits.

The subframes contain:

- clock parameters: used for synchronisation of the satellites clocks
- ephemeris data: Kepler parameters of the satellite position, used for the actual positioning process
- message data: all kinds of useful information, e.g. an ionospheric error model and Universal Time Coordinated (UTC) data
- almanac: estimated ephemeris data of all 24 satellites and health condition of each satellite.

During the so-called lock-in phase of the receiver most of this message data is decoded at least once. Part of the message data (e.g. ephemeris data) is only valid for a restricted period of time. This data therefore needs to be updated and re-decoded by the receiver every once in a while (every 8 hours new ephemeris data is uploaded).

The whole GPS system has been developed by the U.S. Department of Defense (DOD). The earlier mentioned PRN P code is only available for military purposes and allows for accuracies up to 3 m (95% interval). However, for civil use only the PRN C/A code is available. What is more, the GPS signals for civil use are distorted on purpose for civil use by the U.S. DOD: this is called Selective Availability (SA). This implies that an accuracy of 100 m is assured for 95% of the time. It goes without saying that, due to this restriction, GPS is inappropriate for use as sole means for radio navigation.

A reduction of the errors resulting from SA and the (rough) C/A code, can be obtained by the introduction of a ground-based reference station. The exact position of the reference station (known beforehand) is compared to the position calculated by the reference station's GPS receiver. The resulting error information can be added to the control segment navigation data sent to one of the GPS satellites. The satellites process the incoming data and transmit the navigation message to the user (e.g. aircraft receiver)[1,2]. Aircraft in the vicinity of the reference station can utilise the reference error information to reduce their own GPS positioning error. The principle is known as **differential** positioning (DGPS).

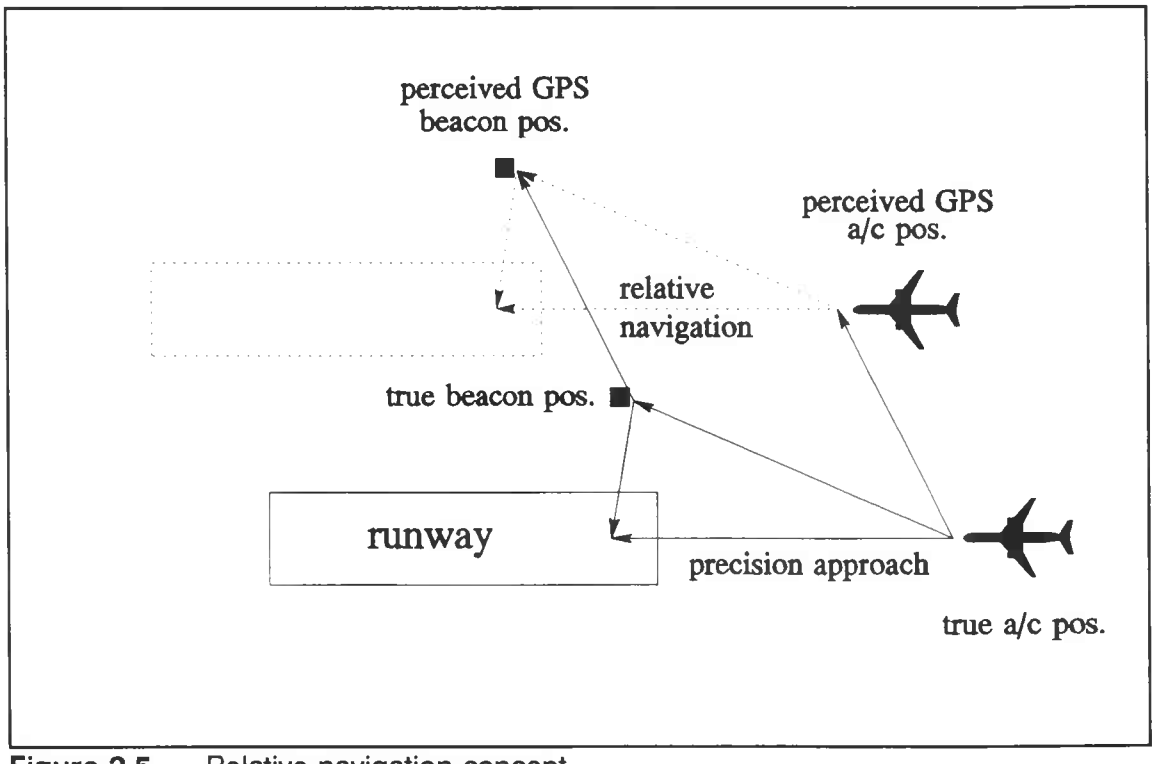

Figure 2.5 Relative navigation concept

Another technique used to reduce the errors made by normal GPS is called **relative navigation** [10]. The position calculated by the aircraft's GPS receiver is compared

1 In fact any alternative data-link facility can be utilised to transmit the error information to the end-user; The MIAS (MLS Integrated Approach System) project at Delft University of Technology makes use of the MLS ADW-channel to send DGPS information from reference station to aircraft.

2 EUROFIX is a proposed integrated navigation system of DGPS and Loran-C at Delft University of Technology: DGPS corrections are sent via a Loran-C data-link channel.

to the position calculated by a ground-based GPS receiver. Both estimated GPS positions must be based on the same satellite signals. This results in two estimated positions with highly correlated error vectors. Therefore, the difference vector between both estimated positions equals the real difference vector.

The distance between the reference station and a runway in its vicinity is accurately calculated beforehand (e.g. by means of laser measurements). The sum of the aircraft-/reference-station vector and the reference-station/runway vector equals the precision approach vector to the runway. The great advantage of this method is the fact that no absolute position needs to be calculated. Figure 2.5 clarifies the principle.

2.2.6 Long Range Navigation C (LORAN-C)

Loran C is a hyperbolic long range navigation system [6]. Large parts of the Pacific and Atlantic ocean are covered by Loran-C. The system is based on time difference measurements between the pulses coming from 1 master and 3 slave transmitters (together referred to as a Loran-C chain). These pulses are modulated on a 100 kHz carrier. A master station transmits 8 bursts. The first slave responds after a certain Emission Delay (ED), depending on the distance between master and slave. The next slave responds after receiving the signal of the first slave, and so on. This process is repeated every t seconds, defining a so-called Group Repetition Interval (GRI). The receiver compares the arrival times of the bursts coming from the master station and its slaves. The position of the various Loran-C chains and their transmitters must be known beforehand to the receiver. The different Loran chains are distinguished by the receiver by means of their different GRI's. The accuracy of Loran-C positioning is strongly dependent on the distance between user and Loran-C station; besides, atmospheric conditions seriously influence the Loran-C signals. In general accuracies between 130 m and 550 m are met for distances between respectively 350 NM and 1000 NM from a transmitter. One of the expected functions of Loran-C is to serve as back-up system for GPS navigation.

2.3 Atmospheric conditions

The atmosphere can be modelled as consisting of different layers, two of which are of crucial importance for radio wave propagation: the troposphere and the ionosphere [11].

2.3.1 Troposphere

The troposphere is the part of the atmosphere up to 11 km. In general the influence of this layer is equal for all frequencies: an additional time delay (compared to a wave propagating through a vacuum) is introduced. Waves propagating through the troposphere are refracted. The refraction index (N) is a function of air pressure (p), air temperature (T) and humidity (e). Consequently the refraction index decreases as a function of altitude. Radio waves with frequencies below 6 MHz are refracted in such way that their propagation paths follow the earth surface. These waves are called *ground waves*. Frequencies above 6 MHz are not bent along the earth surface. Instead, these waves are characterized by *line-of-sight* propagation (straight lines). Other mechanisms, like duct forming (as a result of special weather conditions), do influence high-frequency waves. Radio waves with frequencies in the Super High Frequency (SHF) range band are even more affected by weather conditions. For these frequencies the wave length is of the same order as the raindrop dimensions. As a result radio-waves can be absorbed and severely refracted by the raindrops.

2.3.2 Ionosphere

The ionosphere itself has a layered structure. We distinguish a D (50-90 km), E (90-125 km), F_1 (200 km) and F_2 (200-400 km) layer. Each layer is characterized by its molecule density (N) and its electron density N(e). Both variables can be regarded as functions of the time of day, the time of year, location (earth surface as reference) and solar activity. The ionized molecules result in reflection of radio waves below a certain critical frequency (f_{cr}). Consequently, long length radio waves with low frequencies (up to 300 kHz) are completely captured between the earth and the ionosphere. Propagation characteristics of these waves are usually described by means of *sky waves* (reflected wave

against the ionosphere) and the earlier mentioned *ground waves*. Waves with high frequencies (above 30 MHz) are not reflected. These waves can cross the ionosphere freely, although they are distorted in comparison with free-space wave propagation by: additional group delay time, diffraction from a straight line, and phase shift.

Table 1.1 Radio-wave properties of navigation aids

NAVAID	*FREQUENCY*	*BAND*	*RADIO-WAVE PROPERTIES*
LORAN-C	100 kHz	LF	ground + sky wave, distance < ≈100 km ground wave dominates, minor meteorological influences on phase and amplitude.
NDB	200-1750 kHz	MF	
VOR	108-118 MHz	VHF	direct wave (line-of-sight), reflections against earth surface and (large) obstacles, influence of weather conditions (duct forming).
ILS localizer	108-112 MHz	VHF	
ILS glide slope	330-335 MHz	VHF	
GPS	≈1.5 GHz	UHF	direct wave, (low) absorbtion by water damp and fog, reflections against small obstacles and tropospheric layers, ionospheric distortion (group delay, diffraction, Doppler shift).
MLS	5 GHz	SHF	direct wave, high absorbtion by rain and water-vapour (peak at 1.33 cm), reflections: even against small objects.

After this general description, we focus on the transmitters described earlier. Table 1.1 summarizes the wave properties for frequencies used by the avionic navigation aids. Some properties (influence of obstacles) relate to the module environment, described in the next section.

2.4 Environment

The module environment represents all obstacles that influence the radio-signals coming from the transmitters. Obstacles can be divided in four categories:

1. Natural, non-moving obstacles: mountains, rivers, oceans, etc. (earth surface).
2. Natural, moving obstacles: (large) animals.
3. Man-made, non-moving obstacles: buildings, hangars.
4. Man-made, moving obstacles: aircraft, cars, trucks.

In general, obstacles with dimensions matching the signal (carrier) wavelength influence the propagation of these waves. For low frequency signals (NDB, LORAN-C) relatively large obstacles should be taken into account; high frequency signals are even distorted by water-vapour and small raindrops. Obstacles can influence a radiated signal in three ways [12]:

- shadowing: a wave is completely blocked by an obstacle.
- multipath: the received signal is a composition of a direct wave, reflected waves and diffracted waves. A wave is reflected by the surfaces and diffracted by the edges (discontinuities) of an object. Reflected and diffracted waves can either increase the received signal strength (in-phase; amplification) or decrease this signal strength (out-phase; fading). Multipath wave parameters are: amplitude, time delay, phase and phase rate-of-change (all relative to the direct wave properties)[1].
- absorption: a part of the radiated wave power is completely absorbed by an obstacle. The amount of radiated and absorbed power depends on the surface structure, represented by the parameters σ (conductivity), μ (permeability), ϵ (permittivity) and roughness.

A possible method of the simulation of the module environment is a representation of obstacles by so-called polygon meshes. A polygon mesh is a collection of polygons,

1 Sky waves can be regarded as a special kind of reflected multipath signals.

edges and vertices. Polygons consists of surfaces characterized by a normal vector and the earlier mentioned surface parameters σ, μ, ϵ and roughness.

2.5 Simulation of real-world

In this module the outputs of the modules *transmitters*, *atmospheric conditions* and *environment* are combined to produce distorted navaid signal. To be more specific: the unpolluted navaid signal from the transmitter module is distorted due to atmospheric conditions (additional time delay, phase shift etc.). Multipath propagation is estimated using the true aircraft state, the true navaid states and the outcomes of the module environment. The result is a composed (direct wave and reflected waves) distorted navaid signal. This resultant signal is used as an input for the module *aircraft antennas*.

2.6 Aircraft antennas

The aircraft antennas are characterized by a radiation pattern [13]. Two important radiation parameters are the directive gain (D_g) and directivity (D_0) of the antenna. Directive gain is defined as the ratio of the radiation density in a specified direction to the radiation intensity of a reference antenna (usually an isotropic source). Directivity is the maximum directive gain in a certain direction. The directive gain and directivity result in an amplification or attenuation of the received navaid signal. This property can result in complete suppression of multipath signals. From this, it is clear that the position and orientation of the antennas on the aircraft are of crucial importance. Navaid signals are sometimes completely lost, due to the fact that a certain part of the aircraft (wing, fuselage etc.) is blocking the line-of-sight path between transmitter antenna and receiver antenna. This problem is often solved by the use of two antennas instead of one. If both antennas receive the same navaid signal (no shadowing), additional information about the attitude of the aircraft can be obtained (by means of a comparison of the phases of both signals). Currently the aircraft antennas for a certain navaid system are not actively tuned by the FMCS[1]: e.g. all VOR navaid signals are received with the same antenna

1 In literature the expressions FMS and FMCS are applied confusingly: the term FMS is often used to refer to all on-board aircraft navigation electronics (=avionics); while FMCS refers to the computational heart of the FMS, and consists of two FMC's.

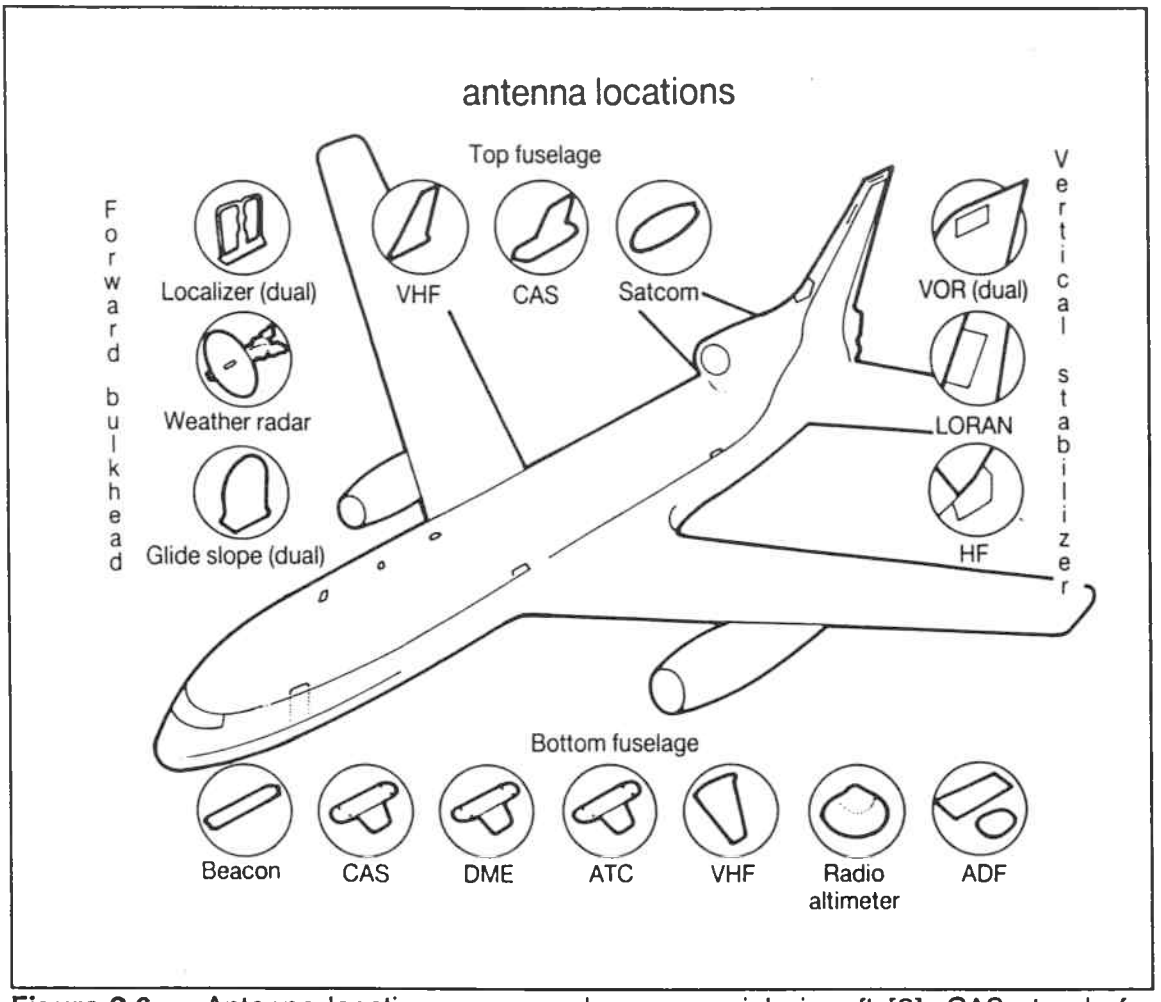

Figure 2.6 Antenna locations on a modern comercial aircraft [8]; CAS stands for Colission Avoidance System.

configuration. In the future, the FMCS can be expected to (fine) tune the antennas and to direct them automatically to the transmitting navaids. Figure 2.6 shows the locations of the aircraft antennas as they are mounted on today's modern commercial aircraft.

2.7 Software receiver

The software receiver can be regarded as the heart of the on-board aircraft navigation system. The software in this module is a part of the FMCS. All of the navaid signals received by the various aircraft antennas are gathered and processed in the software receiver, resulting in a best-estimated 3-D position (and in the future possibly the velocity and attitude) of the aircraft. In order to calculate this position, the position, velocity and orientation (=state) of the navaids involved in the positioning process need to be known

as accurately as possible. The state of a navaid can either be extracted from the navigation data base or de-modulated from the transmitted navaid signal (part of navigation message). Navaids with fixed positions (no velocity), like NDB, VOR/DME, ILS and LORAN-C have their state recorded in the navigation data base, where as a non-static system, like GPS, adds its exact position to the transmitted signal. This state data only becomes available once every T seconds. In between two updates, state parameters are calculated using pre-recorded ephemeris data from the navigation data base.
The MLS system can be regarded as a fixed-position-transmitter system with the navaid states stored in the navigation data base. Additionally, the exact position of the MLS elements (runway as reference) is transmitted by MLS itself as a part of the auxiliary data block. The stored transmitter state can be corrected with this accurate position data.

The decision to use or disregard certain signals depends on the 'health' status of these signals. 'Health' data is often modulated on the navaid transmitted radio signal. There are two methods to mix the data of two different navaid systems [6]:

1. Integration: position-solutions of both systems are calculated and compared to each other. The result is a weighted Most Probable Position (MPP).
2. Hybridization: information of a single navaid is not sufficient for a proper position fix (e.g. only three GPS signals are received; one is shadowed). Pseudo-ranges of other systems are added to get sufficient information for a position fix.

An example of a navaid selection algorithm is the one used in the FMS of a Boeing 747-400 [14]. The algorithm is based on integration of radio-navigation systems and the on-board Inertial Reference System (IRS) (discussed in the next section). Every two minutes the FMS software searches the navigation data base and generates a list of all navaids within 300 NM of the aircraft's computed position. Positions are calculated according to one of the four different lateral navigation (LNAV) modes :

mode 1 ILS localizer deviation data + IRS position (only in final approach area)

mode 2 rho-rho[1] positioning using 2 DME stations + IRS position

mode 3 rho-theta[2] positioning using 1 DME and 1 VOR station + IRS position

mode 4 position determined by IRS only

The mode selected depends on the availability, reliability and accuracy of the navaid signals. Of course, only the active mode is displayed on the ND by the instrument subsystem. Additional information (e.g. barometric height, air speed) obtained via the DADO, is combined with the radio-navigation and IRS computed position and velocity (e.g. DME slant range is converted into ground range with the help of the barometric height). Other variables computed by the FMCS include the true track angle, ground speed, vertical flight path angle, drift angle and wind vector.

2.8 Other state-determination systems

Radio-positioning is not the only way to determine aircraft state variables. The on-board IRS [15] produces attitude and acceleration information using up to 3 Inertial Reference Units (IRU). Integration and double integration of the acceleration information results in respectively velocity and position of the aircraft with respect to its initial state. This initial state (position, velocity, and attitude[3]) must be inserted into the FMCS at the start of a flight. Future systems involve infrared techniques to determine vehicle position.

Positions calculated by the alternative positioning systems are compared to the radio-navigation positions (integration). Once again this results in a MPP.

1 With rho-rho positioning, two distances are determined towards two independent (DME) beacons. This results in two independent circles the aircraft can be flying on. The intersection points (2) of both circles give the estimated aircraft positions. Ambiguity is solved by other-state-determination input (e.g. INS).

2 Rho-theta positioning is a combination of distance measurements (DME) and angle information (VOR).

3 The initial a/c attitude includes the magnetic compass heading.

2.9 Digital Air Data Computer (DADC)

The DADC [7] collects information from the aircraft sensors, such as the pitot static tube. This information includes the static pressure (p), the total pressure (P_t), the static temperature (T) and the angle of attack (α). The DADC calculates from these variables the True Air Speed (TAS), Calibrated Air Speed (CAS), barometric height (H) etc. The outputs of the DADC are electric signals representing the air data. Air data can either directly be shown on the ND or PFD of the instrument subsystem, or it can be used by the FMCS to calculate other variables (e.g. ground range out of slant range).

2.10 Flight plan

So far this report mainly has dealt with radio-**positioning**. However, radio-**navigation** involves comparing the estimated aircraft position with a desired flight path and controlling the aircraft according to a flight plan. A flight plan is defined and inserted in the FMS by the pilot before taking off, using a Control Display Unit (CDU). Information from the navigation data base is used by the pilot to create the flight plan. Another imminent method of inserting a flight plan into the FMS is by means of a data-link, with the flight plan being transmitted by an air company or by ATC.
The flight plan can be shown on the ND by the instrument subsystem. Selection of the navaids to be used by the FMCS (software receiver) depends on distance between aircraft and waypoints. With the Boeing 747/400 the 5 closest navaids are selected. An alternative selection method rejects waypoints not defined in the flight plan (even if they are at close range), and thus giving priority to flight-plan-defined-waypoints.

2.11 4D-Navigation and guidance system

The 4D-navigation and guidance system gathers and processes all kinds of information possibly needed to be shown on the CRT's of the presentation module. This includes the outcomes of the module *software receiver*: the best estimated aircraft state and the selected navaids/non-radio positioning systems used for this estimation. Other data, coming from the modules *navigation data base* and *flight plan* is often referred to as map background (e.g. waypoints positions, flight path). Air speed and height, calculated by

the DADC, are also collected by the navigation and guidance system and can either directly be transferred to the presentation module or used for guidance purposes.

The guidance function of the system compares the aircraft's 4D-position (best estimated 3D-position + time), generated by the software of the receiver, and DADC data, with the desired flight path defined in the flight plan. Steering commands are generated to control the aircraft along the flight path. Additional thrust commands are produced in order to fulfil the time demands of the flight plan. In other words: the Estimated Time of Arrival (ETA) at a certain waypoint is adjusted until it approximately equals the Scheduled Time of Arrival (STA) defined in the flight plan.

2.12 Navigation data base

The navigation data base contains a number of files. These files contain the so-called static data: during a flight the contents of the files are not altered. The files included are[1]:

- Navaid file
- Waypoint file
- Airway file
- Airport file (runway, gate and airport procedure information)
- Customer file
- Holding pattern file
- Company route file

The contents of the files are used for multiple purposes[2]. A pilot creates a flight plan according to information about the existing waypoints, airways and company routes. The FMCS automatically selects the frequencies of ground-based navaids in the vicinity of the aircraft (e.g. 5 different frequency channels can be used in the Boeing 747-400). Navaid frequency and state information are extracted from the navaid file.

1 source: Boeing 747-400 FMS manual

2 The data base files (supplied by Jeppesen) are updated every 28 days.

2.13 Presentation

The last module of the navigation system is involved with the presentation of the estimated aircraft state. This module is actually a part of the instrument subsystem of the BARESIM configuration (see Figure 1.2). In today's modern aircraft CRT's are widely used for the presentation of aircraft data. The two most important displays in the cockpit are the Primary Flight Display (PFD) and the Navigation Display (ND). The term EFIS is often used to refer to both displays and to distinguish between flight instruments displays and Engine Indication & Crew Alert System (EICAS) displays.
Information shown on the PFD includes [14]:

- aircraft position, velocity, heading, attitude and slip
- autopilot info
- ILS info
- guidance information

The ND contains information about:

- The aircraft position, velocity, track, heading and course
- Map background information about:
  - The flight-plan
  - The navigation aids
  - The Weather Radar

The kind of information shown on the EFIS displays is selected by the pilot via the EFIS Control Panel (EFISCP). For example: the ND can operate in one of four different modes: MAP, APPROACH, VCR and PLAN DISPLAY. The pilot chooses the most appropriate mode (depending on the current flight phase) by turning a 4-position knob on the EFISCP.

3 Information flow between the modules

In the previous chapter the contents of each module were briefly discussed. The focus will now be on the interaction between the modules. Each module uses information coming from another module or from an external source (e.g. from another subsystem of the BARESIM configuration). It then processes the information and outputs data that can be used by yet a third module. In this chapter a block diagram representation (NAVSIM model) is developed to describe the data flow between the modules.

3.1 Activity block representation

There are two different methods of block representation [16]. The first method describes a model by means of data block representation. Each data block contains a certain type of data. Arrows between the blocks define the action needed to get from data in block A to the data in block B. The second method is reciprocal to the first method; it is called the activity block representation. A block embodies a certain activity. The arrows linking the blocks represent data. Figure 3.1 shows both methods.

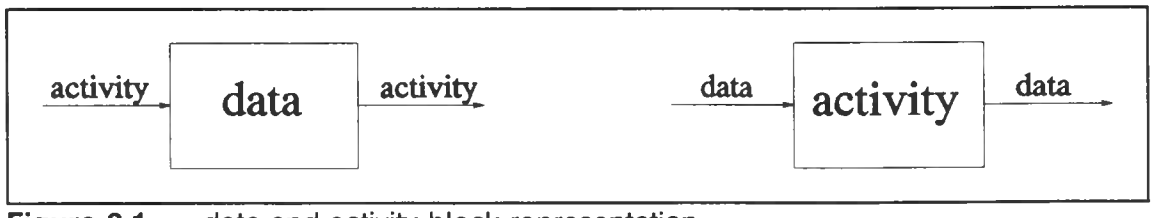

Figure 3.1 data and activity block representation

The navigation modules described in the previous chapter are clear examples of activity blocks: e.g. navaid signals (data) are polluted by a process in the *simulation of real-world module* (activity); the result is a distorted signal (data). Obviously input data is needed, processed and output data is the result. The amount of output can be controlled by the control data. The activity of the block is characterized by mechanism parameters. Figure 3.2 shows the basic activity block concept. The module NAVSIM as a whole can be represented using the same activity block representation (Figure 3.3). Note that the host computer both provides input data (the aircraft model, as a part of the host com-

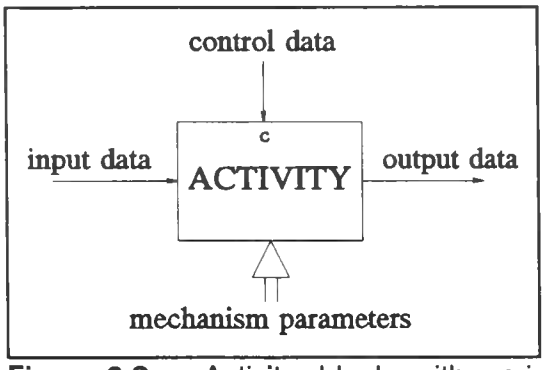

Figure 3.2 Activity block with variables

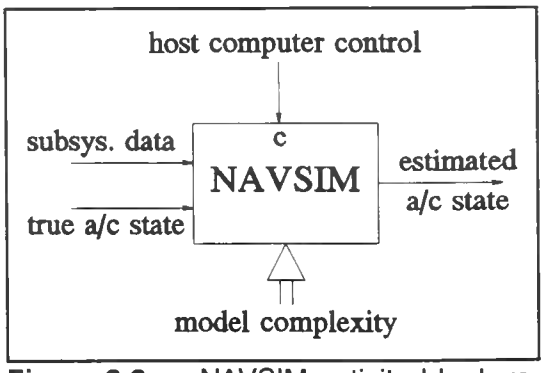

Figure 3.3 NAVSIM activity block representation

puter software, generates the true a/c state) and controls the NAVSIM module. A second input source consists of data coming from one of the BARESIM subsystems (see Chapter 1).

3.2 Activity block structure diagram of the navigation system

Now that we have found a proper way of representing the navigation modules (activity blocks), the activity block variables of each module are defined. Appendix A includes a complete listing of all variables.

Next we get to the structure diagram of the whole navigation system: the NAVSIM model. Note that this was done earlier, but in a rough manner (see Figure 3.3); but now we can go into details. The arrow connections between the modules represent the correlation of the input, output and control data of the various modules. Figure 3.4 shows the resulting NAVSIM block diagram. Some additional notes are given to clarify the diagram:

- The main input of the whole system is the **true aircraft state**, generated by the host computer. The main outputs are the **visual images** on **ND/PFD** including the **best estimated a/c state**.
- The **c** just inside a module indicates that the data entering the module at that point, consists of control data; **c/i** means both control and input data is provided by this link.
- Each module is controlled by a **reset** input. This is to protect the whole navigation system from going down if just one module is blocked. Resetting this module

Modular Navigation Simulation

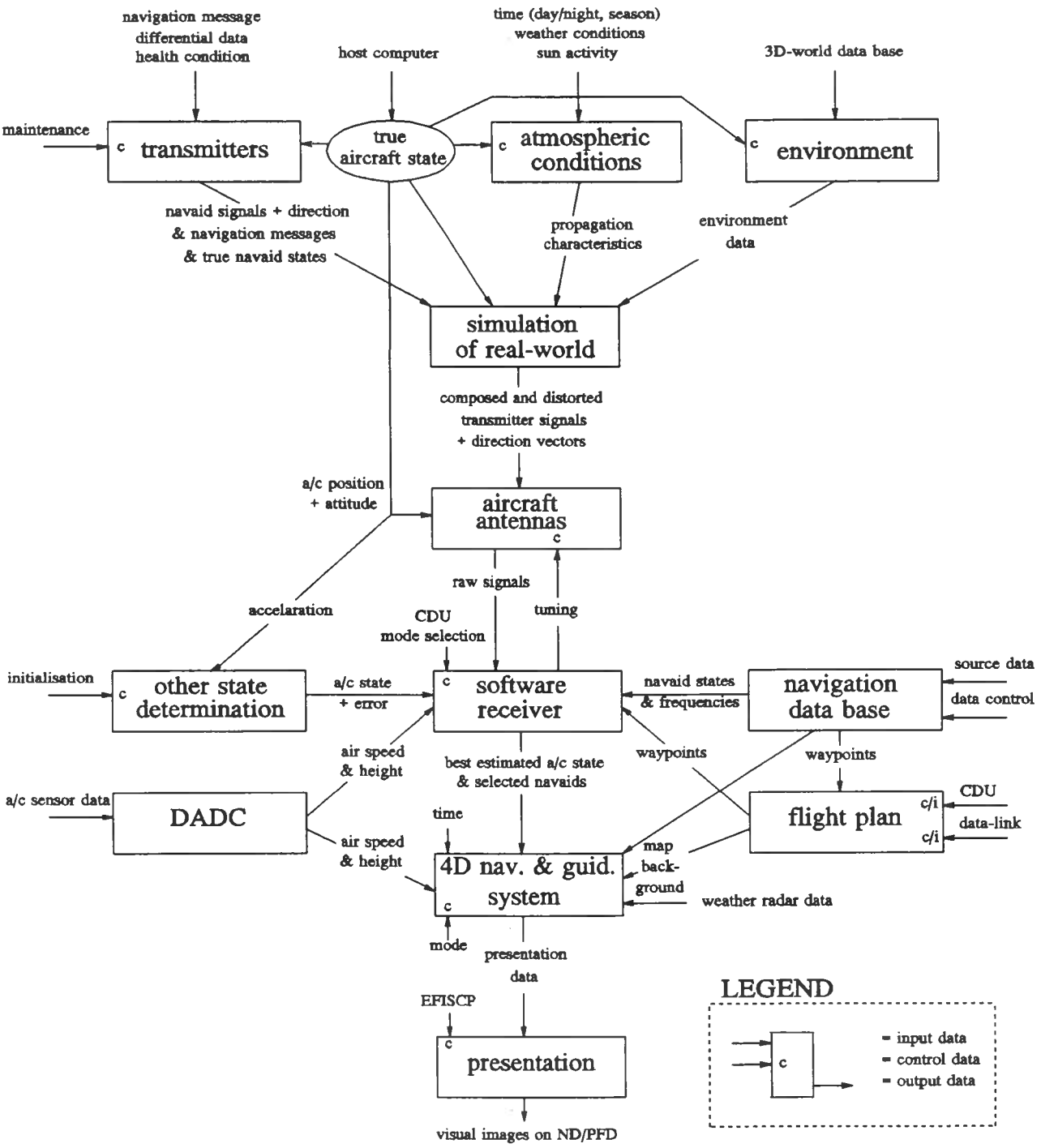

Figure 3.4 Detailed NAVSIM activity block representation

will solve the problem. This reset control data is not shown in the figure. This is done on purpose: omission of all **resets** increases the readability of the structure diagram.

- The **true aircraft state** is not a module. To indicate the importance of this data it is encircled:
  - The **true a/c position** controls both the *atmospheric conditions* and the *environment* module, hence reducing the amount of output data: only atmospheric and environmental conditions in the vicinity of the aircraft are considered.
  - The **true a/c state** is used as input for the *transmitters* module. The complexity of this module depends on the programmer. For a simple simulation of transmitter time delays and angles, the true a/c state has to be known beforehand (see also ?).
  - The **true a/c state** is inserted into the *simulation of real-world* module to allow for reflection calculations (multipath).
  - The **true a/c position and attitude** are used together with the relative antenna position and attitude by the *aircraft antennas* module to simulate attenuation or amplification of the received radio signals.
  - The *other state-determination systems* module includes the on-board INS. The INS needs the **true a/c acceleration** to output a best-estimated a/c state (a/c state + error).
- One of the inputs of the module *simulation of real-world* is coming from the module *transmitters*: it is named **true navaid states**. To calculate multipath propagation, the positions and velocities of the transmitters and receiver must be known. The **true aircraft state** contains the receiver position and velocity. The transmitters' positions and velocities are provided by the *transmitter* module. These parameters need to be known beforehand for reflection modelling in the *simulation of real-world* module. In reality however, they are not needed until the position solution in the module *software receiver.*
- The selection of the navaids used for the positioning process is done by the *software receiver*. Usually the navaids that are closest to the aircraft are selected from the *navigation data base*. An alternative selection method is based on the *flight plan*: only navaids (e.g. waypoints) that are directly involved with the flight

plan are selected. Note that only a restricted amount of navaids can be selected at a certain moment (for the Boeing 747-400, 5 frequency channels are available).

- The modules *software receiver* and *4D-navigation and guidance system* are both part of the aircraft's FMCS. Another important function of the FMCS (outside the scope of this report) concerns the performance of the aircraft [15, chpt.6]. Two contradicting performance criteria are a minimum fuel consumption and a minimum flight time. The performance function of the FMCS is directly connected with the Thrust Management Computer System (TMCS), which controls the engines.

3.3 Comparison of NAVSIM model diagram with reflection model diagram

Bloem developed a model to represent the environment in a radio-navigation simulation. In Chapter 8 of his report [12] he introduces a data block diagram for the reflection model as a part of the environment. In this section a comparison is made between the recently developed NAVSIM activity block diagram and the data block diagram developed by Bloem. If the NAVSIM model is complete, there should be no problem fitting the reflection diagram model in. One could say it is a fair test for the NAVSIM model completeness.

Figure 3.5 shows the data block diagram proposed by Bloem[1]. The different entries are briefly explained:

Polygon characteristics

These are the earlier discussed polygon parameters σ (conductivity), μ (permeability) and ϵ (permittivity). Roughness is not taken into account because only specular reflection and not diffuse reflection or diffraction is considered.

Signal data

The signal data represents the properties of the waves transmitted by a certain radio navigation system (e.g. GPS, MLS, VOR/DME, ILS). These properties include the signal frequency, polarization, modulation technique used, etc.

1 Used by permission of the author.

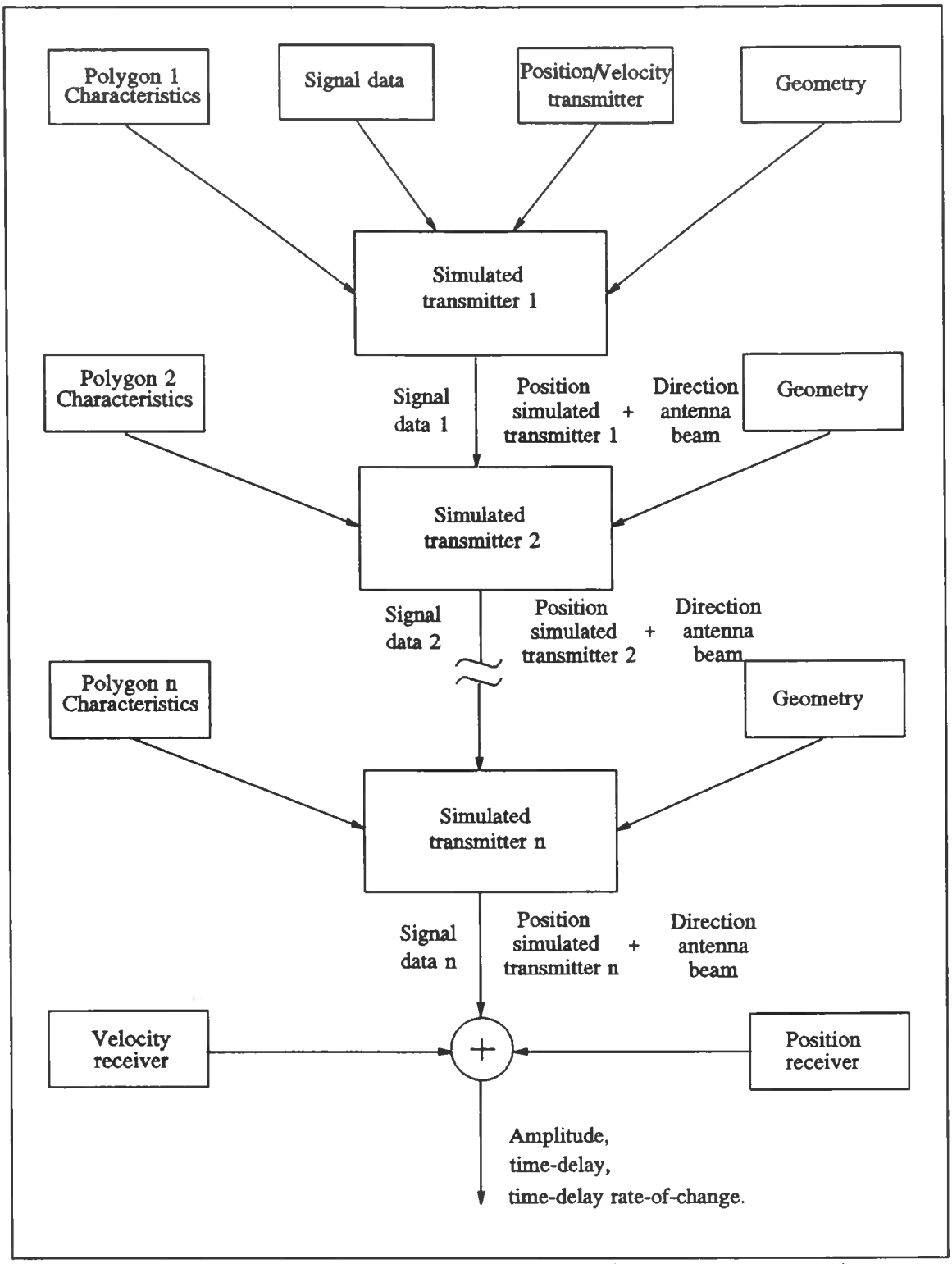

Figure 3.5 Data block diagram for reflection model (developed by Bloem)

Position/velocity of transmitter

The position of the transmitter is needed to calculate the reflection points that lead to multipath effects. Doppler shift in the multipath signal (in case of GPS) can only be calculated knowing the velocity of both transmitter and receiver.

Geometry

The geometry of the scene contains the positions of the polygons and their orientation with respect to each other.

Position/velocity receiver

The position and velocity of the receiver are needed to compute the reflected signals time delays and Doppler shift.

All of these data entries are needed to obtain an estimate of the reflected signal properties (amplitude, time-delay and phase and phase rate-of-change).

For the verification of the NAVSIM model the question rises: where does this data block diagram fit in? It is quite clear that calculation of reflections of the transmitter signal should be part of the module *simulation of real-world*. Consequently, the data blocks should correspond with the inputs and outputs of that same module. Indeed the **polygon characteristics** and **geometry** can be regarded as environment data from the module *environment* (4); the **signal data** is part of the input provided by the module *transmitters* (1); the **position/velocity of the transmitter** are extracted from the same *transmitters* module (1); the **position/velocity of the receiver** are part of the **true aircraft state** (2); the resulting **reflected wave properties** of the data block diagram are combined with the direct transmitter signal in order to produce the composed signal (o). Atmospheric distortions, based on the propagation characteristics from the module *atmospheric conditions* (3), may be added. The result is a composed and distorted signal (including its direction vectors) going to the *aircraft antennas*. Figure 3.6 shows the data block diagram properly inserted in the *simulation of real-world* module.

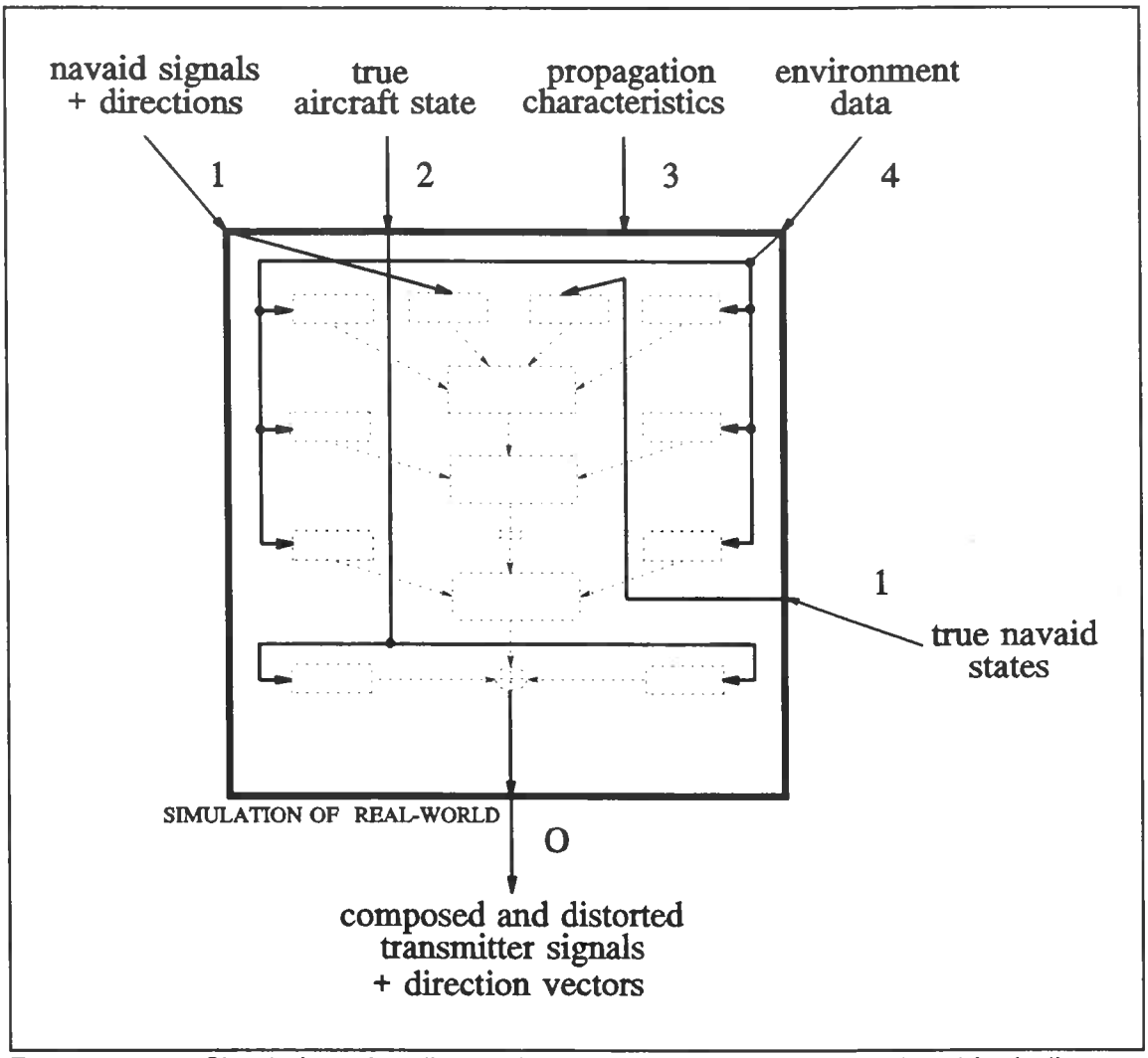

Figure 3.6 Simulation of reality module containing the reflection data block diagram

3.4 Comparison of NAVSIM model diagram with D/GPS-signal model diagram

Braasch ([17], [18] and [19]) developed a closed-loop DGPS signal model block diagram, for a simulation of GPS positioning and navigation. Figure 3.7 shows the model[1]. As with the reflection model of Bloem, Braasch' signal model should be embodied by the NAVSIM block diagram. A short explanation is given of the function of the various blocks:

1 Used by permission of the author.

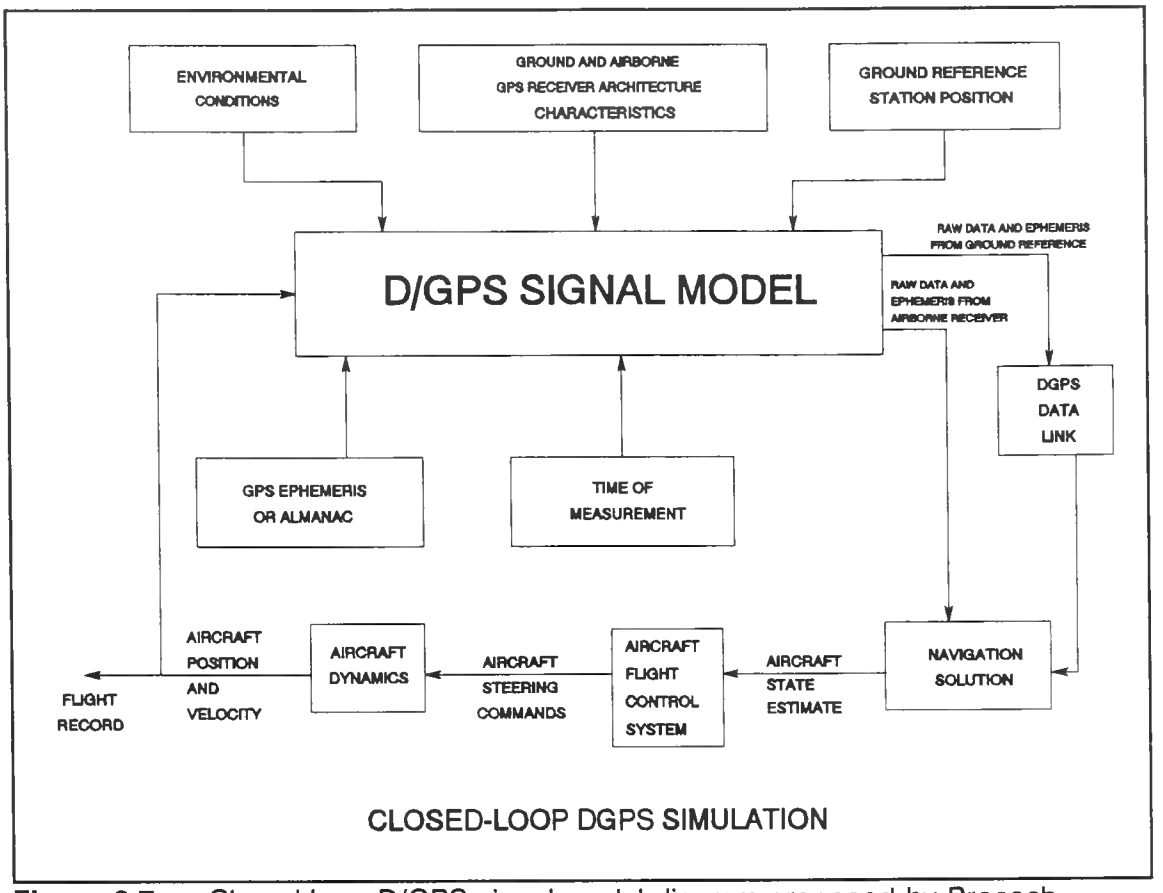

Figure 3.7 Closed-loop D/GPS-signal model diagram proposed by Braasch

D/GPS Signal Model

The D/GPS signal model activity block forms the heart of the block diagram. Its function is comparable to the NAVSIM module *simulation of real-world*: the undistorted signals of the satellites are polluted by it. But the D/GPS signal model incorporates more: the module *aircraft antennas* is included in this block. The output of the D/GPS signal model is raw data from an airborne receiver. This data is comparable to the output of the NAVSIM *aircraft antennas* module.

Environmental conditions

This data block consists of atmospheric condition parameters like weather conditions (to calculate tropospheric time delay), electron density (to calculate ionospheric time delays), etc. This block also includes multipath parameters to represent reflections.

Ground and airborne GPS receiver architecture characteristics

This data block input of the D/GPS signal model needs further explanation. As stated before the heart of the closed loop model is the D/GPS signal model: this activity block can be regarded as a GPS receiver emulator. With the variety of GPS receivers currently available, this implies that receiver related properties need to be inserted in the emulated receiver. In the NAVSIM model, these properties are not taken into account until the *software receiver*.

Ground reference station position

The true reference station position is needed to process DGPS error information.

GPS ephemeris or almanac

This data block contains the navigation message broadcasted by the GPS satellites.

Time of measurement

This data block represents the requested time of measurement by the simulation driver. This time is not exactly equal to the true time of reception (see [18] for an in-depth explanation).

DGPS data link

This activity block represents the DGPS data-link properties. The DGPS error information can be transmitted using any kind of data link: we earlier mentioned the possible use of Loran-C stations or the MLS ADW-channel as links.

Navigation solution

The best estimated aircraft state is calculated in the activity block navigation solution. In reality, this is done in the software of the navigation system of the FMCS.

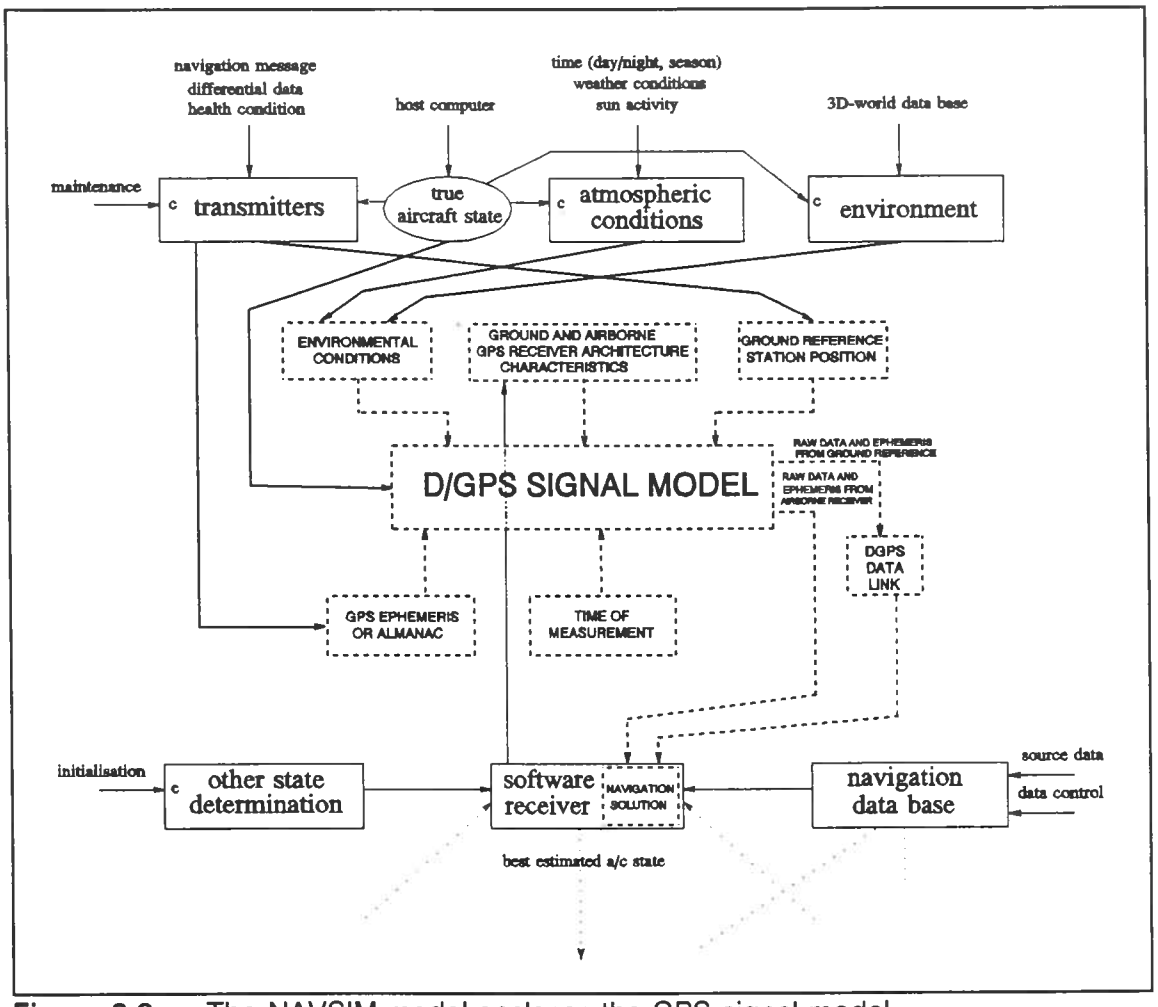

Figure 3.8 The NAVSIM model encloses the GPS signal model

Aircraft Flight Control System (AFCS[1])

In the activity block AFCS the estimated a/c state is compared to the desired state as defined in the flight plan. Aircraft steering commands are the result.

Aircraft dynamics

In the last activity block the steering command are converted into aircraft dynamic movements. The dynamic responses of an aircraft depend on aircraft type and configuration (i.e. number of passengers, amount of cargo).

1 The term AFCS is yet another expression referring to the on-board avionics: in general it is used to label the guidance function of the FMCS.

Just like the Bloem model, the Braasch diagram should be covered by the NAVSIM model (Figure 3.8). The **environmental conditions** stem from the modules *atmospheric conditions* and *environment*. The **receiver architecture** is delivered by the module *software receiver*. The **ground reference station position**, as well as **GPS ephemeris or almanac** are extracted from the module *transmitters*. The **time of measurement** equals the host computer simulation time. Finally the **navigation solution** can be considered a part of the *software receiver*. Though there exists no one-on-one correspondence of the GPS signal model and the NAVSIM model, all blocks of the GPS signal model are contained within the NAVSIM model.

3.5 Complexity of the NAVSIM model

In the preceding sections two existing models were compared to the NAVSIM model. Bloem's diagram proved the complexity of just one module of the whole NAVSIM model. Braasch' model pointed out the complications involved when taking into account just a part of one navigation aid: DGPS. Both examples stress the importance of thorough consideration on the simulation 'depth'[1] of the NAVSIM model: what amount of computer power is needed for different levels of simulation?; what modules are expected to be computer-capacity-consuming?; what data flow connections require a relatively large internal network bandwidth?

3.5.1 The specific contents of the NAVSIM modules

The abstraction level of the NAVSIM model is considered the highest level possible. This implies that none of the modules may be omitted without losing a complete representation of the whole navigation system. On the other hand, the modules themselves may consist of several internal layers. For example, the module *software receiver* does not only embody the algorithms (e.g. tracking loops) to calculate a best-estimated radio a/c state, but includes the integration algorithms to mix radio and non-radio determined states as well. Furthermore, the detail level of the modules highly depends on the purpose of the simulations. It would not be wise to make any speculations about the

1 The simulation depth indicates how close to reality the simulation is.

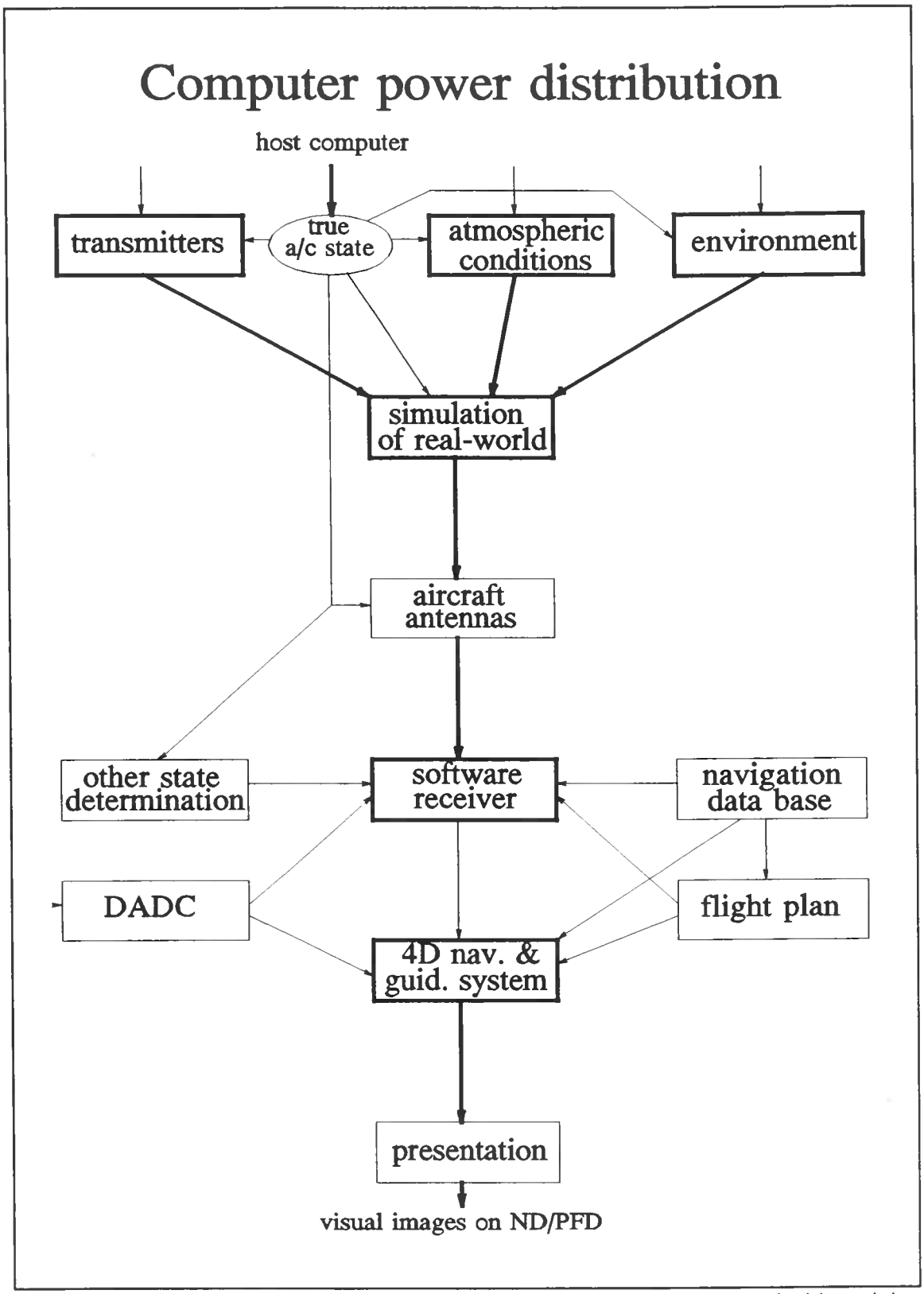

Figure 3.9 Computer power and connection bandwidth requirements: bold modules need more power; data flow is larger for bold arrows.

exact amount of computer power needed for a certain detail level; it is suggested that by the time a real navaid model will be implemented, a careful analysis should be made about the detail level required. The detail level is determined by the goal of the research. The maximum achievable detail level is directly related to the available computer power. If the available power is less than the required one, the simulated processes will be slower than in reality: the simulation is non real-time. On the other hand, if the amount of computer capacity equals or even exceeds the required power, processes can be run either in real-time[1] mode or in fast-time mode. Fast-time mode implies that test-data can be gathered from the simulated process faster than the same data would come available in reality. Computers must be purchased, corresponding to these requirements and cash funds available. It might well be that, for the benefit of a real-time simulation, certain modules are to be considered of minor importance for certain test purposes: these modules should be regarded as 'dummies'. Dummy modules simply copy the input variables to the output variables (1* operator). Nevertheless a rough indication can be made of where to expect a necessity of relatively large and small computer power. In Figure 3.9 modules that ask for huge computational power are represented in bold.

This section is concluded with two examples concerning GPS and MLS. The basic idea behind a varying simulation depth is illuminated focusing on the *transmitters* module.

GPS

A GPS signal consists of two circular polarized carrier signals both derived from a 10.23 MHz base frequency: the L1 band of 1.58 GHz and the L2 band of 1.23 GHz. These carriers are modulated by P (L1 and L2) and C/A (L1 only) PRN codes. Once again the frequencies (often referred to as chip-rates) of these codes are derived from the base frequency of 10.23 MHz: 10.23 MHz for the P code and 1.023 MHz for the C/A code. Additionally, a navigation message is modulated on the same L1 and L2 carrier frequencies, transmitted with a rate of 50 bps. Processes in the *transmitter* module, concerning GPS navaids, can be simulated by (in order of increasing depth):

1 The term real-time is related to the calculation speed of a simulated process: if a simulated process takes the same time (depending on sufficient computer power) as its real counter part, that is, if a simulated second corresponds to a real second, a simulation is labelled real-time.

Table 1.2 Computer power requirements for the module *transmitters* for different levels of detail (simulation depth); the last column gives the ratios of the data generated depending on the simulation depth.

DEPTH	*GENERATED OUTPUT (1 satellite)*	*FREQ.*	*BANDWIDTH (bits/s.)*	*RATIO*
1	time delays navigation message true satellite state	8 Hz 50 Hz n.k.*	58 Hz	1
2	C/A PRN code navigation message true satellite state	1.023 MHz. 50 Hz n.k.*	1.023 MHz	1.76e4
3	samples containing: carrier frequency (L1) C/A PRN code true satellite state	 1.58 GHz 1.023 MHz n.k.*	3.16 GHz	2.72e7

* Not Known: the frequency with which the true GPS satellite states are needed by the *simulation of real-world* module to estimate reflection parameters, fully depends of the simulation depth of this very module. There is no computer power needed inside the *transmitter* module; only a data-link bandwidth between *simulation of real-world* and *transmitters* will be required.

1. Generating time delays corresponding to the pseudo-range between satellite and a/c receiver. In most of the available receivers, this information comes available once a second per satellite[1]. Navigation messages are generated like simple ASCII strings and transmitted with a realistic 50 bps. rate. The true transmitter and a/c state are needed to calculate the time delays. The true navaid state is directly transferred to the *simulation of real-world* module.

2. Generating both modulating signals. That is, producing a PRN code and a navigation message. The signal time-of-transmission is broadcasted as a part of the navigation message. The navigation message is sent to the *simulation of real-world* module, and from there on transferred to the *a/c antennas* with a realistic rate of 50 bps. The true a/c state is not needed. The navaid state is broadcasted

1 Suppose a time delay can be coded in 1 byte = 8 bits.

both as a part of the navigation message (like in reality), and as plain ASCII numbers, needed for reflection simulation in the module *simulation of real-world*.

3. Generating a carrier frequency signal containing PRN code and navigation message. Samples of the signal (amplitude) are sent to the simulation of real-world module. Additional phase samples represent the circular polarization character of the satellite signal. Neither the navaid state, nor the a/c state are needed in the process. Once again the true navaid state is both embedded in the navigation message and directly sent to the *simulation of real-world* module.

Table 1.2 summarizes the computer power requirements for the three levels suggested above.

MLS

The MLS signals consist of a carrier signal with a frequency of 5 GHz. The omni-directional carrier signal is modulated (DPSK technique) by Basic Data Words (BDW) and Auxiliary Data Words (ADW), transmitted with a rate of 15.625 kHz [20]. Time multiplexing is used to distinguish between data and positioning info. Pre-ambles notify the receiver of the information sent. Directional elevation and azimuth antennas broadcast scan beams that cover a fan-shaped area. Simulation of MLS navaids implies (in order of increasing depth):

1. Generation of azimuth and elevation angles. BDW and ADW information (referred to as the navigation message) are produced like ASCII strings. No signal direction vectors are considered: all signals are supposed to be omni-directional.

2. Generation of the 'hit-times' of the elevation and azimuth scan beam. The BDWs and ADWs are generated during specific time slots with a 15,625 baud rate. Pre-ambles are added. Still no signal direction info is produced.

Table 1.3 Computer power requirements for the module *transmitters* for different levels of detail (simulation depths) for MLS simulations.

DEPTH	*GENERATED OUTPUT*	*FREQ.*	*BAND WIDTH (bytes/s)*	*RATIO*
1	azimuth angle elevation angle navigation message* true MLS state	39 Hz** 39 Hz 202 bps. n.k.***	103 Hz	1
2	azimuth hit-times elevation hit-times navigation message true satellite state	78 Hz 78 Hz 15,625 bps. n.k.	2109 Hz	20
3	samples containing: carrier frequency scan information navigation message true satellite state	5 GHz 15,625 bps. n.k.	10 GHz	9.7e7

* Only Basic Data Words are considered: 5 words of 18 bits (excluding preamble and parity check bits) are broadcasted every second, 1 word of 18 bits is generated every 0.16 s.

** High rate azimuth approach guidance rate is considered: 39 times per second an angle must be generated by means of 8 bits (1 byte).

*** Not Known: the frequency with which the true MLS antenna states are needed by the *simulation of real-world* module to estimate reflection parameters, fully depends of the simulation depth of this very module. There is no computer power needed inside the *transmitter* module; only a data-link bandwidth between *simulation of real-world* and *transmitters* will be required.

3. Generation of samples of the 5 GHz signal including the time multiplexed elevation and azimuth scans and the BDW's and ADW's. The samples include direction of the scan beam that are generated. Time slots are used for distinction between data and positioning info.

From these examples it is obvious how easily computer power is exceeded: increasing the simulation depth will slow down the model. Table 1.3 summarizes the computer power requirements for MLS transmitter simulation. An important earlier mentioned topic concerns the real-time requirements. For research of a pilot's overall response on new

response on new navigation systems, a real-time test environment might be essential, where as with reliability and so-called continuity of service tests a slower simulation would do.

3.5.2 The amount of data flow between the modules

The communication between the modules was represented by arrow connections. Just like the contents within the modules, the connections between the modules are not equally large. Some modules generate a lot of data, whereas a module like *flight_plan* only produces a small data flow. In Figure 3.9 a rough hint is given of where to expect large data flows. But one should keep in mind that all data flow depends both on the control data and the simulation depth of the modules involved. For example: the amount of data generated by the modules *atmospheric conditions* and *environment* is reduced by the control data **true a/c state**. The atmospheric conditions can be represented by the same characteristics for all navaid frequencies (low simulation depth = small data flow) or by varying influences for the different frequencies of ILS, MLS, GPS etc. (high simulation depth = large data flow between *atmospheric conditions* and *simulation of real-world*). The module presentation does not require a lot of computer power, where as its data-links ask for a large bandwidth.

4 Conclusions and recommendations: NAVSIM model

In this chapter, the last of the first part, some concluding remarks on the NAVSIM model are made. The NAVSIM module representation gives rise to several questions:

- Is it possible to implement the model in a real-time environment?

  This strongly depends on the complexity of each module and program skills of the software developer and therefore can be a problem with today's limited computer power. The next question evolves from this:

- Does it make sense, to first generate navaid signals (e.g. complex circular polarized GPS signals) in the transmitter module, and later on process them in the software receiver, to restore the initial information? Why not just generate simple radio-position and error data (corresponding to different atmospheric and environmental conditions) in the software receiver and save an enormous amount of signal calculation time?

  The answer to this question lies in the lessons learned in the past. Simplifying complex models can be profitable for the current computer capacity and speed (earlier discussed noise modelling), but one should bear in mind that future research on navigation systems will ask for extended model presentation. The NAVSIM modular block diagram can both be used for simple modelling with today's restricted amount of computer power and for future complex modelling when new and faster processors become available. The software developer determines the complexity and contents of each module.

It is recommended that if any changes are to be made in the NAVSIM model, this should be done with an awful lot of consideration. Adding new arrows results in a redefinition of many variables. What is more, often the addition of new arrows is not necessary: data can be transported using the already-existing connections!

PART II - NAVSIM: developing a test-environment for the model

5 Object-Oriented Programming

In the previous chapter a theoretical model for NAVSIM was developed. The implementation of this model asks for a programming language suited to represent the different modules. In this chapter the use of an Object Oriented Programming language is justified (Section 5.1). The basic principles of OOP are explained in Section 5.2. In the last section (5.3) the OOP-language Borland C++ is promoted to be used as the program language.

5.1 Modular NAVSIM and Object-Oriented Programming

The different modules of the NAVSIM model represent various parts of the navigation systems. To ensure easy implementation of the whole model, a programming style should be chosen that corresponds to the modular approach. OOP perfectly fits in here. Each module can be defined as an object. An object in OOP can be regarded as a self-supporting entity. That is, an object contains both data and methods describing the kind of action to perform on the data. This is exactly corresponding to the way that the NAVSIM modules should work: processes in the module should influence only data of the same module.

5.2 Basic principles of Object-Oriented Programming

In this section the basic principles of OOP are covered [21].

5.2.1 Introduction

OOP is as big a revolution in the programming world today as structured programming was 20 years ago. It is a whole new way of thinking. Many professional C programmers are just learning about OOP and debating its merits, and it has not yet become the new standard. Why make the change? What does OOP offer that warrants so much relearning and so much rewriting of existing code?
OOP's most obvious advantage is with group projects. Ten years ago, PC programs were often one-person projects, but that is rarely the case today. With the increasing power

available on the most simple desktop, PC programs that take advantage of that power are generally written by a team of programmers. The same holds for NAVSIM: numerous programmers will work on the implementation of the different modules of the model.

With these kinds of large programs, debugging and maintaining them has grown exponentially more difficult. OOP addresses the problems inherent in debugging and maintaining large, complex programs.

5.2.2 Encapsulation

One of the key features of OOP is *encapsulation*: data and the functions that operate on that same data are encapsulated into a self-contained object. A clear example of this feature are objects in so-called visual development environments: each object is painted on the screen and described by clicking the mouse on it; characteristics such as its color, its size and its behavior can be altered (independent of other objects). Encapsulation in OOP-languages is accomplished through the use of *classes*. Classes are comparable to C's data structures and Pascal's data records. But besides data, they also contain the program code for operating on that data. This program code is referred to as *member functions*.

Encapsulation provides numerous advantages. Data within a class can be declared *private*. This means it is accessible only within that same class. This ensures that only that class's member functions can access the data; a highly useful protection in a multi-programmer project such as NAVSIM. Besides, with encapsulation large programs are much more readable , because all the related code and data is in one place. Object-oriented programs are also more modular and therefore more easily maintained.

5.2.3 Inheritance

OOP is much more than encapsulation: *inheritance* is the second topic discussed. Inheritance lets us derive new classes that inherit the properties (both data and member functions) of a previously defined base class. The new properties are built on top of the inherited properties, which can save the programmers quite a bit of coding. *Multiple*

inheritance refers to a class that inherits from multiple base classes. Through the inheritance principle, a hierarchy of classes is created to form the backbone of a program. Designing a useful and elegant class hierarchy is no trivial matter. Therefore OOP programming takes much up-front planning. What is more, since the code from parent classes is pulled into derived classes, it is easy to create a huge overhead in a program without even realizing it.

5.2.4 Polymorphism

The third aspect of OOP is *polymorphism*; directly translated from the Greek it means: "having many shapes". With polymorphism, the same function or operator (such as a plus sign) can have multiple meanings, depending on the context in which the operator is used. For example, a plus sign can be *overloaded* to mean concatenation when the operands of either side of it are strings.

Example:

6 + 10 = 16	(normal + operator)
poly + morphism = polymorphism	(overloaded + operator for strings)

With functions, polymorphism is achieved either through overloaded functions, where all ambiguities are resolved statically at compile time, or through *virtual functions*, where ambiguities are resolved dynamically at run-time. Virtual functions provide the OOP-language with a special kind of extensibility that can be powerfully exploited in class libraries. Virtual functions can be used as provided, or overridden with customized versions of the functions. This flexibility makes OOP code highly re-usable. Programmers do not have to be forever re-inventing the wheel just because they want to make one aspect of the code slightly different.

5.3 The OOP-language Borland C++

After this general introduction about OOP, we have to decide what OOP-language kit to pick. Since C++ is one of the most powerful OOP-languages and smoothly connected to C, the choice is obvious. Canter [22] compared several versions of C++.

Borland C++ was rated editors' choice. The complete development kit consists of two entry-level compilers Turbo C++ for DOS and Turbo C++ for windows, and the professional-level compiler Borland C++. The latest release (june 93) is version 3.1.

5.3.1 C++ as a superset of C

The real power of C, gained by using a restricted instruction set with great flexibility [23], is maintained in C++. C++ is a superset of C, providing (optional) additional instructions for the normal C programmer. Pointer operations are similar, though in C++ a call-by-reference[1] is extended with the use of a so-called *reference parameter*. This means that no 'dummy' pointer has to be declared within a function to alter a variable. Instead the variable is referenced (& sign), meaning that it gets a second name. The variable changes automatically if the reference name is changed within the function.

5.3.2 The entry-level compiler Turbo C++ (version 3.0)

Turbo C++ is a part of the complete kit of Borland C++. It is an entry-level compiler, but fully equipped with Borland's famous Integrated Development Environment (IDE). The IDE is the same one as used with Borland Turbo Pascal. Pull down menus facilitate the integrated use of compiler, linker, debugger and C++'s project manipulations.

A great deal of confusion was met by a comparison of the version numbers and features of Borland C++ and Borland Turbo C++. Turbo C++ version 3.0 is the latest release and part of the complete Borland C++ version 3.1(!) kit. One of the new features of Turbo C++ version 3.0 is *color syntax high-lighting*, which helps the beginning programmers to spot the elusive C and C++ pointer errors. Color syntax highlighting gives comments, reserved words, operators, data types and the programmer's exclusive code, different colors to distinguish between one another. Turbo C++ version 3.0 is completely implemented with AT&T version 2.1 and ANSI C specifications.

1 A call-by-reference means a function is called with the address of a variable A: the variable A can be changed by the function; a call-by-value implies a variable A is copied to a local function variable L, therefore the function cannot alter the variable A directly.

6 A test-environment for NAVSIM using C++

Documenting software of a large program is often a very tedious and dangerous job. The moment the documentation is finished, the actual program software may already have been changed. This results in false information in the software document. Therefore it was decided not to give a fully detailed software description, but instead briefly discuss the main structure of the program and some sneaky tricks used during programming. Additionally the C++ source code is accompanied by a lot of comments[1]. It is recommended that these comments are frequently updated during the implementation of new program code. For a better understanding of the NAVSIM code, the reader is strongly advised to single step through the program[2].

In this chapter a test-environment for NAVSIM in C++ is discussed. It was by no means my intention to cover all details of the model. On the contrary: the main purpose of the basic test-environment is to facilitate the complete implementation of the NAVSIM by individual yet-to-be-assigned programmers. For these programmers, only the software of a specific module should be of their concern. The remaining part of the model will be *transparent* to them; implying that they can neither 'damage' the remaining part of the software, nor have to understand themselves the code of other modules. First the one-on-one correspondence between NAVSIM modules and C++ objects is exploited, resulting in the definition of classes (Section 6.1). Within these classes data and member functions are defined for the NAVSIM modules. Some special techniques used to program the test-environment are covered in Section 6.2.

1 In C++, comments are distinguished from program code by // followed by text; in C comments are recognized by /* at the begin and */ at the end of text.

2 The expression single step is used by programmers to indicate the execution of a program line by line (instruction by instruction). During single stepping, so-called watches can be used to show the values and meaning of variables.

6.1 Defining C++ Classes for the NAVSIM modules

In the previous chapter inheritance was promoted as a powerful tool to avoid the need of multiple definitions[1] of both data and member functions in multiple classes. In our NAVSIM test-environment we use the same technique to define some common variables for all modules.

6.1.1 The Base Class: MODULE

In Section 3.2 we stressed the importance of a control variable **reset** for each module, to prevent the system from crashing. Another important variable for each module is the local **simulation time**. Both variables are defined in the base class MODULE and consequently inherited by all derived classes. Special attention should be paid to the globally defined SIM_TIME. Although the definition of a global variable is strongly advised against in C and C++[2], the global simulation time must be accessible to all classes and their member functions. The purpose of the locally defined **simulation times** becomes clear now: once inside a class SIM_TIME is copied to the local simulation time, avoiding the danger to change SIM_TIME directly.

6.1.2 Derived Classes

All of the modules of our NAVSIM model are defined as derived classes of the base class MODULE. Class definitions are always in upper-case characters, to distinguish between definition and declared instances (=objects) of a class. Objects are always declared in lower-case characters. The derived classes defined are: TRANSMITTERS, ATMOSPHERIC, ENVIRONMENT, REAL_WORLD, ANTENNAS, OTHER_STATE, RECEIVER, DATA_BASE, DADC, NAV_GUID, FLIGHT_PLAN, and PRESENTATION. The tree-hierarchy of NAVSIM is shown in Figure 6.1. So, each module of the NAVSIM module is repre-

1 NOTE: a lot of confusion can be avoided by distinguishing between the **definition** of classes, structures, variables, and member functions, and on the other hand the **declaration** of instances or objects of these classes or structures. Computer memory is reserved the moment variables are **declared**; a **definition** of a class does not take any memory!.

2 Global variables are 'dangerous' to use: they go directly against the principle of encapsulation; a global variable can be changed by any 'unaware' programmer, not knowing that he may not alter this variable.

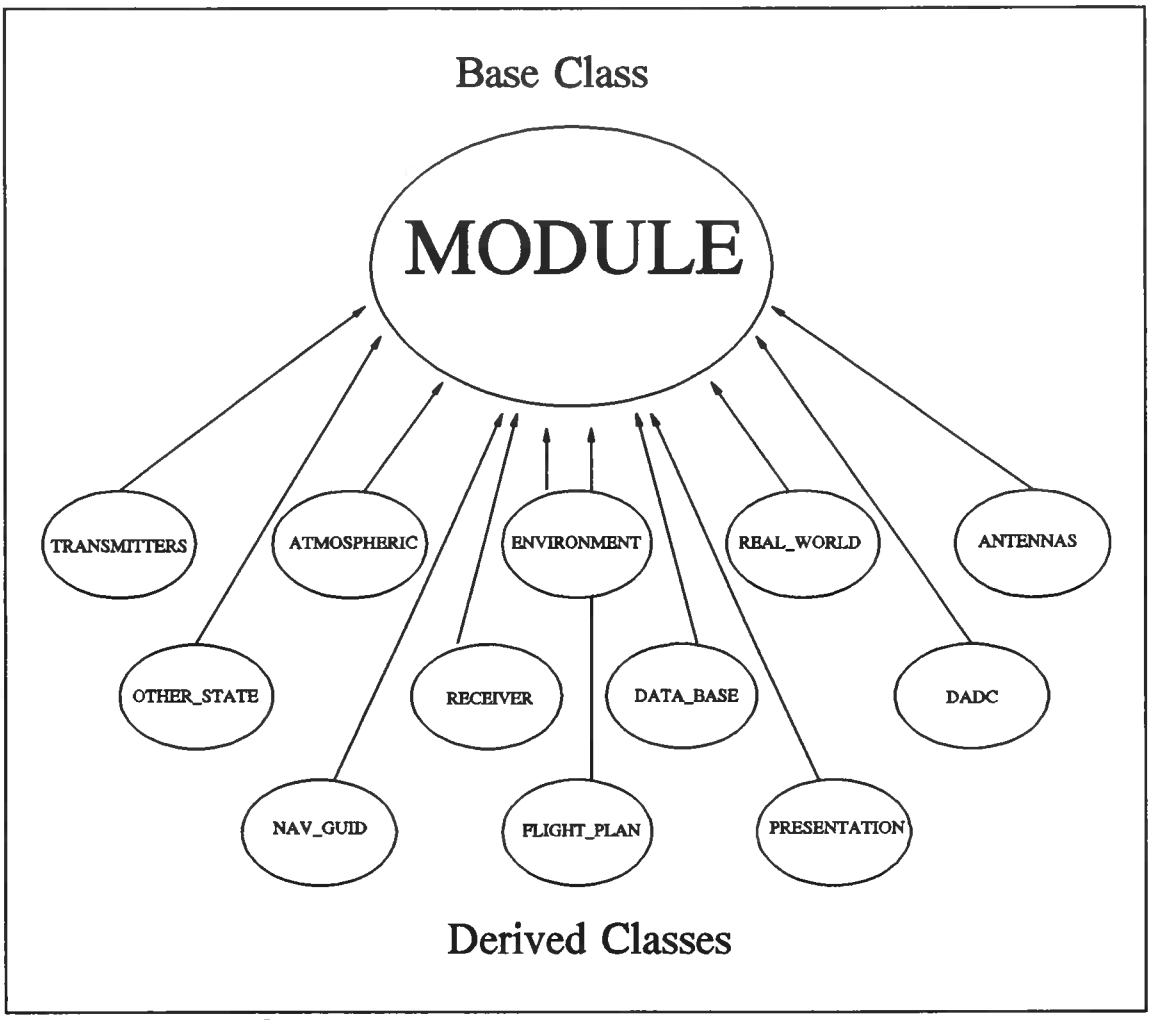

Figure 6.1 NAVSIM class hierarchy

sented by a class.

Before defining input, output and control data (data=variables) for each class, a short consideration on our modules is in place. Instead of defining inputs and outputs separately, inputs alone will do. The reason is that the one-on-one correspondence of the output data of the 'current' module and the input data of the 'next' module. Figure 6.1 clarifies this thought. All this boils down to more simple class definitions: only defining input and control variables. In addition to this member data, member functions are defined within the classes.

We focused earlier on the importance of encapsulation: the protection of certain data and member functions (together referred to as class members). The next section covers

three kinds of access that we have to class members.

6.1.3 Base and derived class access

For a complete and detailed description on class access the reader is referred to [24], Chpt. 13. In this section class member access control is described.

There are three key-words used to specify the protection of class members. These are:

- **public** The member can be used by any function outside the class.

- **private** The member can be used only by member functions and **friends** of the class in which it is defined.

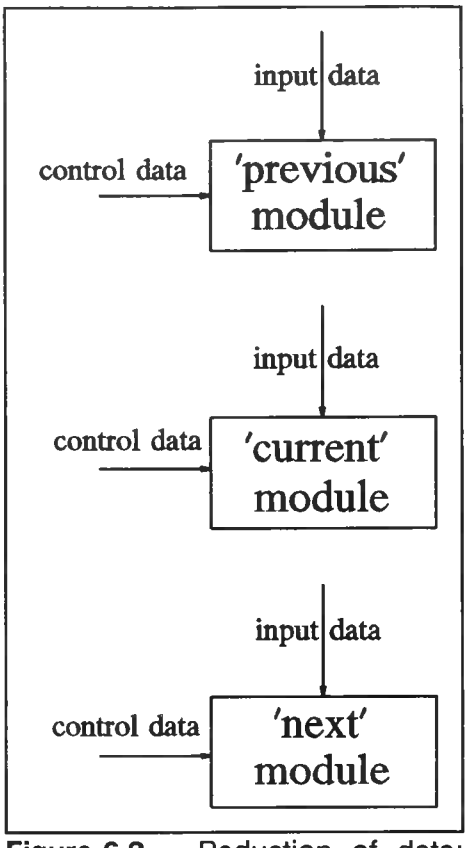

Figure 6.2 Reduction of data: input equals output

- **protected** Same as for private, but additionally, the member can be used by member functions and **friends** of derived classes from the defined base class.

The specifier **friend** needs some more explanation. A **friend FRIEND** of a (base) class **BASE** is a function or class that, although **not** a member function of **BASE**, has full access rights to the private and protected members of **BASE**. Apparently we can make use of **friend** classes to establish the arrow connections in our NAVSIM model. For example:

The derived classes TRANSMITTERS, ATMOSPHERIC and ENVIRONMENT can be made **friends** of the derived class REAL_WORLD, so that they can access the input data of this module. Other modules of the model will not be able to access this data and

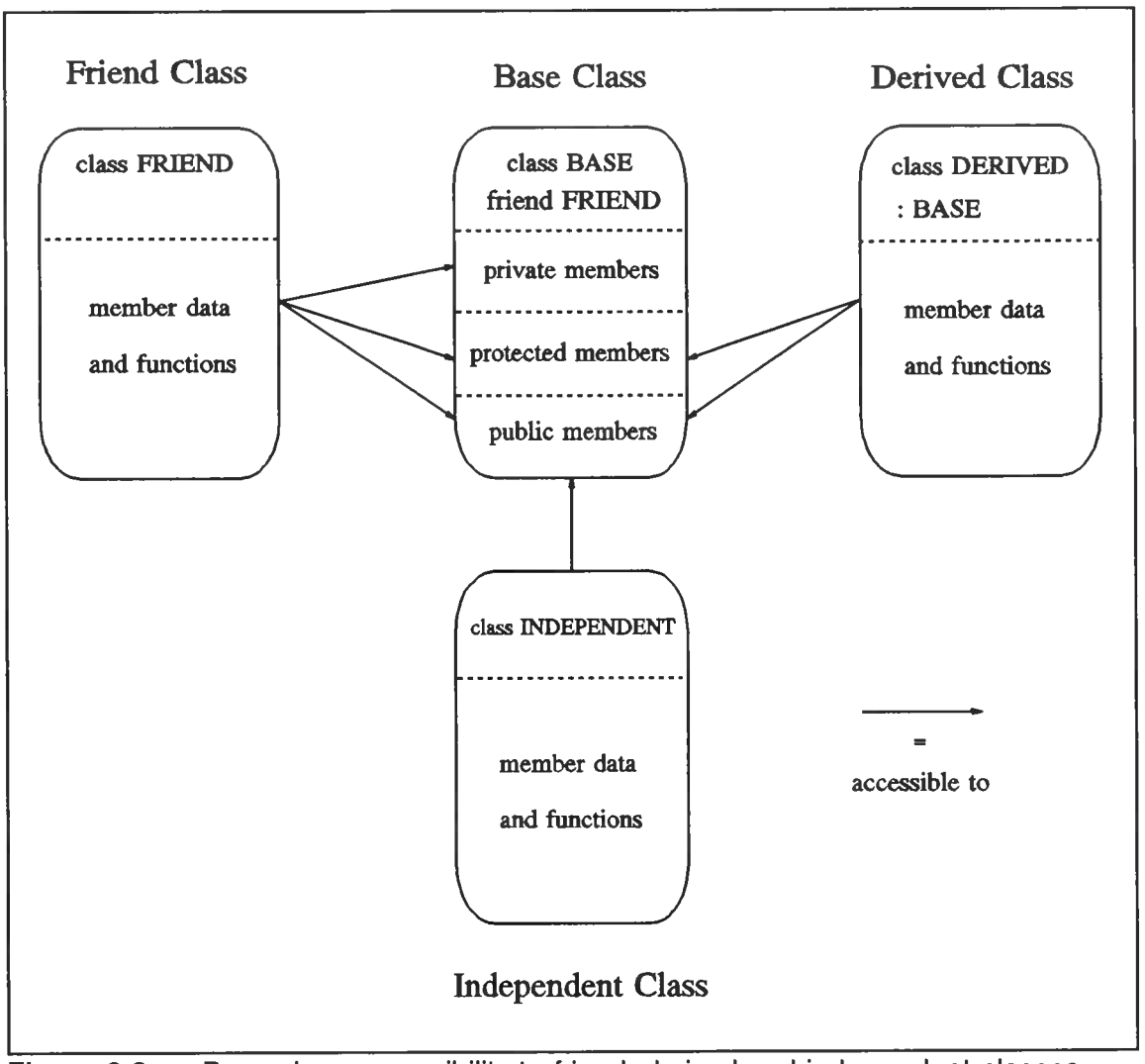

Figure 6.3 Base class accessibility to friend, derived and independent classes

therefore the encapsulation of REAL_WORLD members is still guaranteed to a safe extent. Figure **6**.4 shows the accessibility of the REAL_WORLD inputs. The member functions discussed in the next section are also shown.

6.1.4 Member functions: Process(), Evaluate() and Generate()

We return to the basic properties of the NAVSIM modules. Input data is processed and output data (= input of 'next' module) is the result. This means each class should have

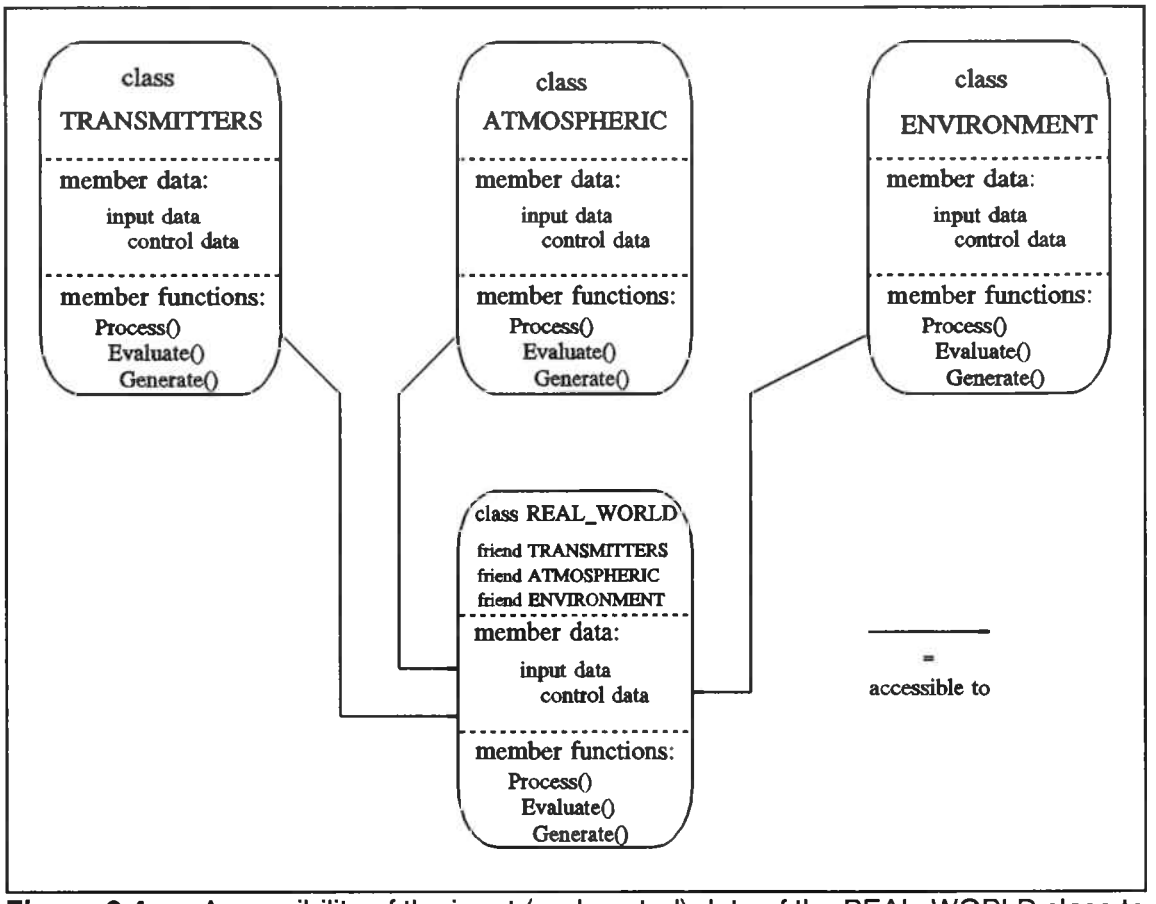

Figure 6.4 Accessibility of the input (and control) data of the REAL_WORLD class to the member functions of the TRANSMITTER, ATMOSPHERIC and ENVIRONMENT classes

a function Process()[1], in which specific code is programmed, depending on the navaids simulated. In our test-environment the Process() functions of the classes are filled with **dummies**: input of 'current' module is simply copied to input of 'next'.

Of course we want to keep track of the input and control data of the modules. The function Evaluate() shows us the current state of all input and control variables of a specific class.

The last function implemented is Generate(). Generate() facilitates testing of (unexpected) inputs of each module. For the class TRANSMITTERS, Generate() lets us assign a value

1 In general in C++ parentheses are used to indicate we are dealing with a (member) function and not with (member) data.

to the input data: in_navigation_message, in_differential_data and in_health_condition. Special attention should be paid to the input data true_aircraft_state. This input should be equal for all modules and will eventually be generated by the host computer (see Section 1.1). However, with the member functions Generate() of each module, true_aircraft_state can be altered for testing purposes of that specific module.

6.2 C++ specifics used for NAVSIM

There are a lot of special tricks in C++ that make programming a lot easier (see [23], [25] and [26]). One should really be familiar with these C++ features, for a reasonable understanding of a program. That is why the next sections seem to be tedious and annoying, but really help the reader to get more insight in the complex C++ code.

6.2.1 Constructors and destructors

Constructors are a special kind of member functions of a class. Each time an object of a class is declared (either statically or dynamically[1]) the constructor of that class is automatically invoked. Actually the real creation of that object is done by the constructor. Apart from creating an object, the real power of a constructor lies in the initialization of the member data. For example: all variables can be set to 0, and all pointers are set to a meaningful address, instead of pointing to 'garbage' or worse to NULL[2]. A constructor gets the same name as its class: TRANSMITTERS has a constructor function TRANSMITTER(). If no user-defined constructor exists for a certain class, C++ automatically generates a default constructor. A special case occurs if an object of a derived class is declared: first the base class constructor is invoked, followed by a call to the derived class constructor.

1 Dynamic memory allocation allows for creation and deletion of objects during run-time. This implies that memory can be free-ed if an object is no longer needed. Static objects cannot be de-allocated during run-time.

2 C++ and C programs often crash because of pointers referring to NULL. To avoid this, pointers should be initialized as soon as possible once they are declared.

In the NAVSIM program, the base class MODULE has its constructor MODULE(). We mentioned earlier (Section **6**.1.1) the two base class member variables **reset** and **simulation time**. In MODULE() reset is set to an initial non-true (0) and the simulation time to 0 for all derived class objects. As far as the derived class constructors concerns, 'dummy' constructors are defined and can be filled with the appropriate code later (if desirable).

The destructor for a class is called to free members of an object before the object itself is destroyed. The destructor is a member function whose name equals the class' name preceded by a tilde (~). If a destructor is not explicitly defined for a class by the programmer, the C++ compiler will generate a default one. Up to now there is no user-defined destructor code implemented in the NAVSIM program.

6.2.2 Overloading functions and operators

One way that Turbo C++ achieves polymorphism is through the use of *function overloading*. In C++, two or more functions can share the same name as long as their parameter declarations are different. In this situation, the functions that share the same name are said to be *overloaded* and the process is referred to as *function overloading*. Though there are no overloaded functions yet in the NAVSIM program, further implementation of the modules could undoubtedly ask for this feature. The next example clarifies overloaded functions:

Imagine that you want to have a function **Cube** that operates on and returns three different types of variables (int, float and double). In C the declarations would look like:

```
int Cube (int number);
float Fcube (float float_number);
double Dcube (double double_number);
```

In other words, three functions with different names should be declared, though performing the same kind of action. In C++, however, one name can be used according to the "overloading" principle:

```
int Cube (int number);
float Cube (float float_number);
double Cube (double double_float);
```

As long as the argument lists are all different, C++ takes care of calling the correct function for the argument given. As stated before, there is no need for overloaded functions in the NAVSIM environment yet. The functions Process(), Evaluate() and Generate() are all declared member functions of different derived classes. This means they are accessed differently. For example:

Let **transptr** be a pointer referring to an object of type TRANSMITTERS and let **atmosptr** be a pointer linked to an object of type ATMOSPHERIC. The member functions are invoked differently:

```
transptr->Generate();                    // invokes TRANSMITTERS::Generate()
atmosptr->Generate();                    // invokes ATMOSPHERIC::Generate()
```

A second way of achieving polymorphism in Turbo C++ is through the use of *overloaded operators*. The example of polymorphism given in Section 5.2.4 is actually an overloaded (unary[1]) operator. Overloading an operator let us redefine the action of any standard operators when applied to the objects of a given class.

In the NAVSIM program the operator **new** has been overloaded. This operator is used for dynamic memory allocation and comparable to the C function **malloc()**. Instead of digging to deep into the real meaning of **new**, the overloading trick is illustrated by a copy of the code of the NAVSIM program:

```
// dynamic memory allocation
TRANSMITTERS *gpsptr = new TRANSMITTERS;

// new overloaded for all classes
void * operator new(size_t size) {
        void *p;
        p=malloc(size);
        if (p==0) {
                cerr<<"\nmemory allocation error - aborting\n";
                exit(1);
                };
        return p;
        };
```

What actually happens in this code is that **new** is overloaded with a function that checks

1 The following operators are called unary operators in C++: & * + - ~ ! ++ and --

if the newly declared pointer refers to NULL[1]. If so, an error message is generated. The original new does not include this safety check.

Apart from the **new** operator, the extraction operator >>, and insertion operator << are overloaded as well. Originally these operators are used for bit manipulation (shift right, shift left). But once again, if the context in which >> and << are used has to do with input/output streams (in detail discussed in next section), C++ overloads the original bit manipulators by (default) inserters and extractors. The last step to be made is overloading of these default stream operators by user-defined stream operators. The next example clarifies this:

```
// overloaded inserter for class TRANSMITTERS objects

ostream &operator<<(ostream &stream, TRANSMITTERS *transptr) {
        // here user-defined program code for console output
        }

// main program code

int number;
TRANSMITTERS *gpsptr = new TRANSMITTERS;          // dynamic allocation of a new TRANSMITTERS object


cout << "a default inserter is used";             // a string is printed to the screen
cout << number;                                   // the integer number is printed by an overloaded
                                                  // default inserter
cout << gpsptr;                                   // normally, this instruction will result in a
                                                  // compiler error; but << is overloaded for
                                                  // objects of type TRANSMITTERS
```

In the same way that the inserter operator << is overloaded, the extractor operator >> can be overloaded. This comes in very handy when writing and reading to (formatted) files. The next section focuses on file in- and output.

6.2.3 File I/O using C++ streams

A stream is an abstraction referring to any flow of data from a source (or *producer*) to a sink (or *consumer*). The synonyms extracting, getting and fetching are frequently used when speaking of inputting characters from a source; terms like inserting, putting and storing are used for outputting characters to a sink. Classes are provided that support

1 Confusingly, a pointer is checked to be referring to 0 and not to NULL. In C NULL was defined as macro to be 0 (#define statement). However, C++ programmers regretfully tend to use 0 instead of NULL.

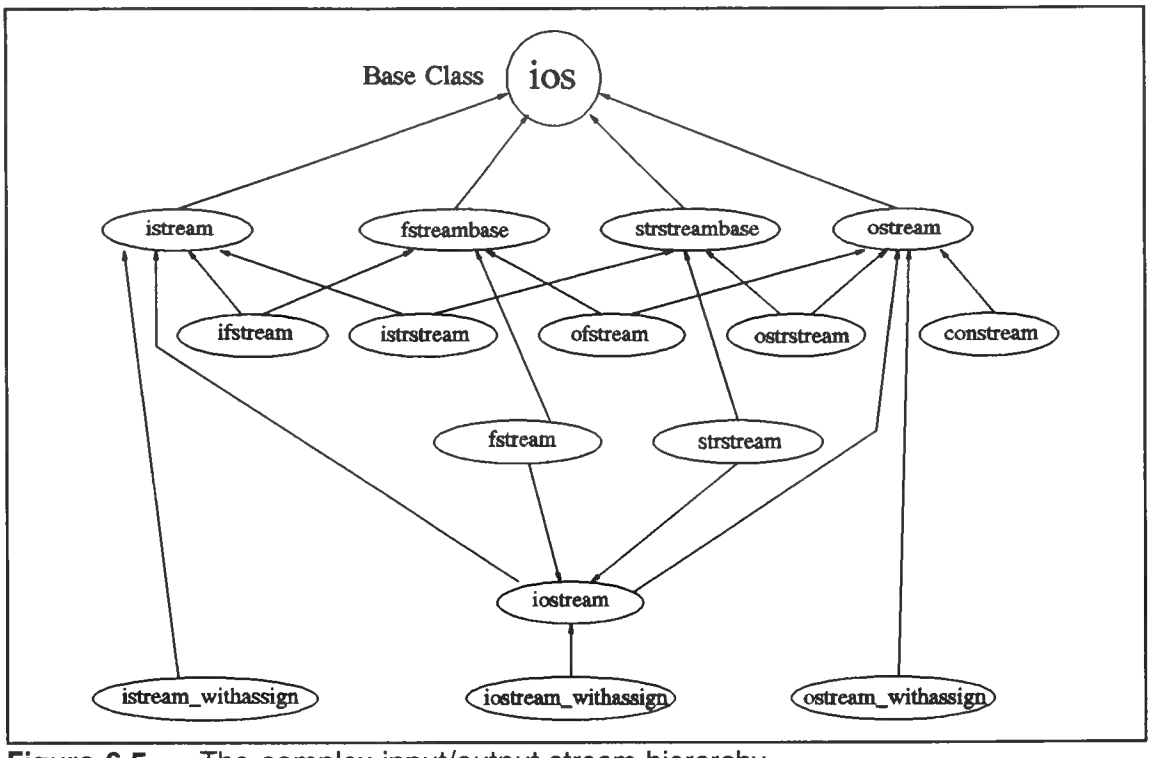

Figure 6.5 The complex input/output stream hierarchy

console output (constrea.h[1]), memory buffers (iostream.h), files (fstream.h), and strings (strstrea.h) as sources or sinks (or both). Like C, C++ contains several predefined streams (i.e. pre-declared objects of stream classes) that are automatically opened when a C++ program begins execution. These are **cin, cout, cerr** and **clog**. The stream **cin** is associated with standard console input and **cout** is the stream associated with standard console output. The streams **cerr** and **clog** are both linked to standard console output and used for error handling. All stream class functions are defined in header files (*.h) and have their function bodies recorded in class libraries. The classes are linked to each other by a complex mechanism of multiple inheritance. Figure 6.5 shows the tree hierarchy.

In the NAVSIM program, the class **fstream** is used to declare a stream for file in- and output to NAVSIM.DAT. An additional advantage of this class is the stream buffer between the program and the output devices, in this case the NAVSIM program and the

1 The extension **h** refers to so-called **header files**. Header files contain class definitions and function prototypes. For more details see ref. [22].

file NAVSIM.DAT. This means that data is not written to the file until the file is actually closed. This is done at the end of the program. While running, all data is written to the stream, saving a lot of hard disk access time[1]. The stream tree hierarchy provides **fstream** with a lot of member functions. Together with its own member functions an object of fstream can access files with special file-pointer functions and format options inherited from its base classes. A complete overview can be found in ref.[23].

6.3 The NAVSIM project file

At the end of this chapter we return to one of the basics of C and C++. Let us briefly describe the way an executable file is made in C/C++. First an ASCII text is written with the use of an editor, for example with the help of the Borland's IDE editor. The code is saved in a file with an extension like **.C** or **.CPP**. Next the *compiler* is called and the ASCII code is translated in a sort of assembly code. The result is an *object*[2] file with extension **.OBJ**. The last step involves the *linker*, which connects the different object files, eventually creating an executable file (extension **.EXE**). The question arises: where do the different object files come from with just one program? The answer lies in project files.

A stubborn programmer may actually compile his program in one go, but this leads to a larger code, and a longer compilation time. Therefore it is strongly advised to break a large program down in modules (not to be confused with the NAVSIM modules!). This is where the project file (extension **.PRJ**) comes in. In a project file the different modules of a program, all with extension **.C** or **.CPP** are brought together. In fact there are more files in a project: pre-compiled library files (extension **.LIB**, either user-defined or provided by Borland) can be added and linked to the object files. The advantage of library files over object files is that only truly used functions of an **.LIB** file are linked to the body of the program; where as object files are linked completely, thereby increasing the size of your program enormously.

1 Computer freaks know that the same hard-disk-movement-saving can be obtained by 'caching' the disk with a special cache program, like *SMARTDrive(R)*: a part of computer memory is an emulated hard disk.

2 The term object file has nothing to do with objects of a class!

The great benefit of project files is the reduction of compilation time: an unaltered program part does not have to be re-compiled by C/C++; the already existing object file is used.

After this short explanation the question arises: what about the earlier mentioned header files? Header files (extension **.H**) are just ordinary parts of a **.C** or **.CPP** file. To avoid 'ugly' definition of structures and classes at the beginning of a program file, header files

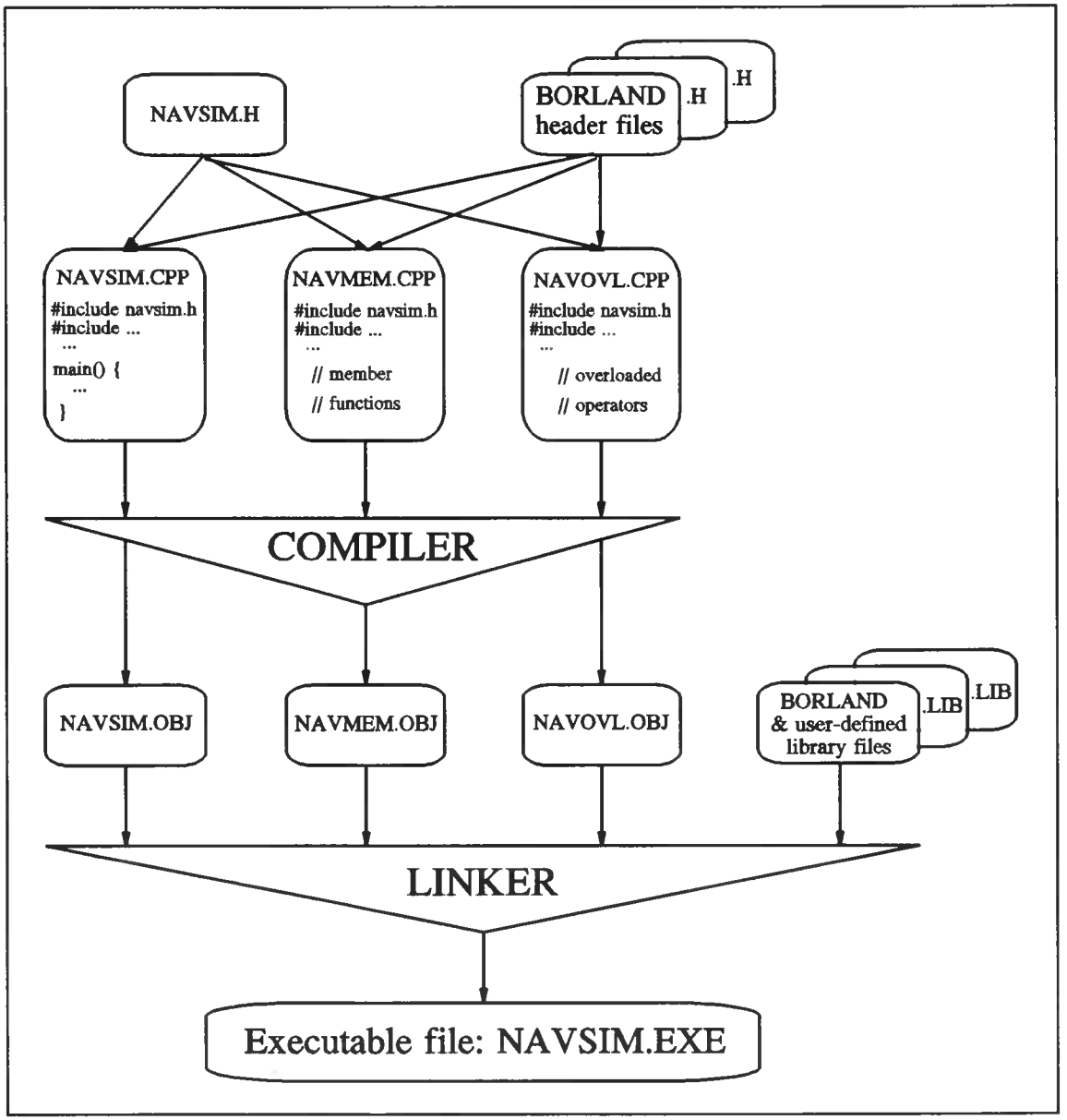

Figure 6.6 The process involved to get an executable file; the project NAVSIM.PRJ is used as an example.

contain all these definitions. The **.C** or **.CPP** file can make use of all the defined types by including (`#include` statement) the header files. During linking the bodies of the functions defined in the header files, are linked from either **.OBJ** files or **.LIB**[1] files. The exe-file process is elucidated in Figure 6.6 with the NAVSIM project as example.

The NAVSIM project file contains the following modules:

NAVSIM.CPP: Main program, contains dynamic object declarations of all classes and calls to the various member functions of these classes.

NAVMEM.CPP The member functions of the classes are defined in this module. This program part will grow extensively large when more of the NAVSIM model is implemented; a further program break-down would be advisable.

NAVOVL.CPP This program module consists of all overloaded operators.

The NAVSIM classes are defined in the file NAVSIM.H. All data input is read from, and written to the file NAVSIM.DAT.

1 Each C++ program is automatically linked with the standard Borland libraries C*.LIB and MATH*.LIB (* depends on the memory model used).

7 Conclusions and recommendations: implementation

This chapter concludes the second part of this paper. Concluding remarks on the implementation of the theoretical NAVSIM model are made.

The implementation of NAVSIM proved to be a very complicated task. With the use of an OOP language, like Turbo C++, a lot of pre-planning had to be done, before one single line could be programmed. The earlier-stated article of Canter [21] points out the slow process of getting really familiar with C++: a time span of two - three years is suggested for optimal use of C++. Nevertheless the basic structure and initial outlay of the NAVSIM test-environment were implemented.

The robustness of the C++ code needs to be verified. To overcome the tedious file format constraints, I spent a lot of time testing little independent program parts. But all of this effort really paid off: the code could easily be copied to the main program.

The main objective of the test-environment was to make a body transparent for the module-dedicated, yet-to-be-assigned programmers. For most of the program this is actually the case. However it is recommended that this report is studied before implementing specific module software. Also, it is strongly advised to 'single step' the program at least once for a better understanding of both C++ and the main program structure. Last but not least the lack of flow diagrams in this report is justified: up to now there are no real loops and condition jumps in the program, simply because no concrete navigation system is covered yet; if any part of the program asks for explanation, comments are added. Project notes are added to the NAVSIM.PRJ file. In this file special comments are made on specific program items. These notes should be read thoroughly.

Up to now there is no User Friendly Interface (UFI) implemented in the test-environment. In general, UFIs facilitate the use of a program, e.g. by means of pull down menus. The menu-controlled environment of Windows(R) is a typical example of a UFI. Borland C/C++ comes with a graphics library, suited for presentation and UFI purposes. It is recommended to extend the NAVSIM program with a UFI.

References

[1] IFALPA HUPER Committee,
AA committee submission to the huper committee concerning an identified problem with FMS navigation systems, agenda item 2.4: Computer Assisted flying - including Automation
IFALPA HUPER Committee Meeting, Somerset West, 24th-27th October, 1990.

[2] S. Advani,
The basic research simulator programme and the industrial and aerospace community: opportunities for cooperative research, report LR-662,
Dept. of Aerospace Engineering,
Delft University of Technology, October 1991.

[3] E. Theunissen,
D^3S: The DELPHINS DISPLAY DESIGN SYSTEM, A-402
Ir. Thesis, Dept. of Electrical Engineering,
Delft University of Technology, 1991.

[4] T. Lamerigts,
The Basic Research Simulator System Interconnection, A-457
Ir. Thesis, Dept. of Electrical Engineering,
Delft University of Technology, April 1992.

[5] H.J. Berghuis van Woortman, W. Aardoom,
Cursus Luchtverkeersleiding, Ir94 (in Dutch),
NLR/RLD, Amsterdam, November 1991.

[6] D. van Willigen,
Radio plaatsbepaling, L104 (in Dutch),
Dept. of Electrical Engineering,
Delft University of Technology, 1992.

[7] F.J. Abbink,
Vliegtuiginstrumentatie I, Ir93I (in Dutch),
Dept. of Aerospace Engineering,
Delft University of Technology, July 1984.

[8] B. Forssell
Radionavigations systems,
Prentice Hall International, Hertfordshire, 1991.

[9] F.J.J. Brouwer,
Navigatie en Geodetische Puntbepaling met GPS, (in Dutch),
Delft University Press, 1989.

[10] An.,
Relative navigation offers alternative to differential GPS,
Aviation Week & Space Technology, pp. 50-51, November 30, 1992.

[11] P.M. van den Berg,
Propagatie van radiogolven, L9 (in Dutch),
Dept. of Electrical Engineering,
Delft University of Technology, August 1988.

[12] C.R. Bloem,
Efficient Simulation of Possible Radiotransmission Path for Simulation of Reflections in Flight Simulators, A-507
Ir. Thesis, Dept. of Electrical Engineering,
Delft University of Technology, April 1992.

[13] L.P. Ligthart,
College antennesystemen, L82 (in Dutch),
Dept. of Electrical Engineering,
Delft University of Technology, March 1990.

[14] E. Theunissen,
The Navigation and Guidance System of a modern commercial aircraft,
Draft Version 0.1, Amsterdam, November 1990.

[15] F.J. Abbink
Vliegtuiginstrumentatie II, Ir93II (in Dutch),
Dept. of Aerospace Engineering,
Delft University of Technology, November 1983.

[16] D.A. Marca, C.L. MacGowan,
SADT; Structured Analysis and Design Technique,
McGraw-Hill, New York 1988.

[17] M. Braasch,
A Signal Model for GPS,
NAVIGATION: Journal of the Institute of Navigation, Vol. 37, No.4,
Winter 1990-91.

[18] M. Braasch,
Manual on the Use of the D/GPS Signal Model, draft version
Dept. of Electrical and Computer Engineering, Ohio University,
Dept. of Electrical Engineering, Delft University of Technology,
May 1993.

[19] M. Braasch,
Navigation Signal Modeling in the Terminal Control Area, presentation sheets
Dept. of Electrical Engineering,
Delft University of Technology, June 2 1993.

[20] ICAO AERONAUTICAL TELECOMMUNICATIONS,
ANNEX 10: to the convention on international civil aviation,
Part 1 - equipment and systems, attachment G, 4th edition,
ICAO, april 1985.

[21] S. Canter,
Object Oriented Programming,
PC MAGAZINE, Vol. 11, p. 374, July 1992.

[22] S. Canter,
C/C++ Compilers: One-Box Development Systems / The Low-Cost Alternative: Entry-Level C/C++ Compilers,
PC MAGAZINE, Vol. 11, pp. 371-373/375-412, July 1992.

[23] R. Lafore,
The Waite Group's Turbo C Programming for the PC, revised edition
Howard W. Sams & Company, Indianapolis (U.S.A.), 1989.

[24] An.,
User's Guide, Turbo C++ Version 3.0
Borland International, 1992.

[25] H. Schildt,
Using Turbo C++,
Osborne McGraw-Hill, Berkeley, 1990.

[26] H. Schildt,
Turbo C/C++: The Complete Reference, second edition
Osborne McGraw-Hill, Berkeley, 1992.

Appendix A: Activity block variables of the NAVSIM modules

Transmitters

input:	-	data from monitor station (e.g. control segment GPS: navigation message)
	-	differential error information from reference station
	-	integrity data (health condition of own or other navaid elements)
	-	true aircraft state
output:	-	signals containing range (rho) or angle (theta) information + direction vectors
	-	navigation message data (including position parameters, eventually needed for accurate radio positioning)
	-	true navaid states
control:	-	maintenance (number of elements per navaid in use)
	-	reset
mechanism:	-	number of navaids simulated
	-	signal complexity

Atmospheric conditions

input:	-	time (day/night, season)
	-	weather conditions
	-	sun activity
output:	-	propagation characteristics (group delay time/phase shift/phase-rate-of-change)
control:	-	true aircraft position (only atmospheric conditions in the vicinity of the aircraft are considered)
	-	reset
mechanism:	-	frequency dependency of distortions (variation of propagation characteristics for each frequency band)
	-	noise models

Environment

input:	-	visual information from a 3D-world data base (e.g. filtered information from BARESIM *visual subsystem*)
output:	-	environment representation data (e.g. polygon mesh parameters)
control:	-	true aircraft position (only the environment within a certain range from the aircraft is considered)
	-	reset
mechanism:	-	representation method
	-	level of detail

Simulation of real-world

input:	-	undistorted transmitter signal + direction vector
	-	navigation messages
	-	true navaid states
	-	true aircraft state
	-	environment data (world as seen by aircraft)
	-	propagation characteristics
output:	-	composed and distorted transmitter signals + direction vectors
control:	-	reset
mechanism:	-	reflection depth (number of reflections considered)
	-	diffraction consideration
	-	noise models

Aircraft antennas

input:	-	composed and distorted transmitter signals + direction vectors
	-	true aircraft position and attitude
output:	-	amplified or attenuated raw signals
control:	-	active tuning (frequency, direction)
mechanism:	-	antenna type (directivity/directive gain)
	-	number of antennas per navaid system
	-	position and orientation of antennas on aircraft

Software receiver

input:	-	raw signal from antennas
	-	aircraft state + error from other state-determination systems (including magnetic compass heading)
output:	-	best estimated aircraft state (3D-Position + attitude + velocity)
	-	selected navaids
control:	-	mode selection (either manually through CDU or automatically by FMCS)
	-	reset
mechanism:	-	navigation mode specification (priority navigation modes)
	-	radio navaid selection used for a certain mode (reliability/availability based on health data)
	-	hybridization or integration of radio navaid signals (filtering techniques used)
	-	filter technique used for integration radio and non-radio determined positions

Other state-determination systems

input:	-	true aircraft acceleration
output:	-	aircraft state + error + reliability + accuracy
control:	-	initialisation at start of flight
	-	reset
mechanism:	-	number of IRU's
	-	system error specifications

DADC

input:	-	aircraft sensor information (e.g. static pressure, dynamic pressure)
output:	-	air speed
	-	barometric height
control:	-	reset
mechanism:	-	kind of sensors used

- number of sensors
- calculation method

Flight plan

input:
- waypoint info from navigation data base
- CDU input
- data-link data

output:
- map background data

control:
- reset
- pilot control through CDU: flight plan creation
- flight plan orders via data-link

mechanism:
- detail level flight plan

Navigation data base

input:
- source data:
  - standard data (public property obtained from e.g. International Civil Aviation Organisation (ICAO))
  - tailored data (route structures of specific airlines)

output:
- map background info: position of waypoints, airports, navaids
- navaid frequencies: used by the software receiver to select the navaids
- navaid states: needed for radio positioning

control:
- reset

mechanism:
- data format

4D-Navigation and Guidance System

input:
- best estimated aircraft state
- selected navaids (by software receiver)
- map background (including weather radar data)
- DADC data (air speed & height)

	-	time
output:	-	presentation data for EFIS
	-	data for auto-pilot function of guidance system (resulting in steering commands)[1]
	-	data for performance function of the FMCS[2]
control:	-	reset
	-	mode selection
mechanism:	-	steering command algorithms

Presentation

input:	-	presentation data from FMCS
output:	-	visual images on ND/PFD
control:	-	mode selection via EFISCP
mechanism:	-	number of display modes (e.g. ND modes include: MAP, APPROACH, VOR and PLAN DISPLAY mode for 747-400)
	-	presentation method (display lay out)

1 This data is mentioned for a better understanding of the navigation & guidance function, but it may not be regarded as real output data: it is internal output from navigation to guidance system.

2 In fact this data is not directly involved with the presentation of the best-estimated a/c state by EFIS, and therefore falls outside the scope of this paper

Appendix B: Detailed NAVSIM model (A3 format)

Appendix C: Source code NAVSIM.PRJ (supplied on request)