

Micro-satellite Data Handling: a Unified Information Model

Benjamin van der Merwe



March 2001

THESIS PRESENTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR
THE MASTER OF ENGINEERING SCIENCES DEGREE AT THE UNIVERSITY OF
STELLENBOSCH

Supervisor: Mr. S. Mostert

Declaration

I, the undersigned, hereby declare that the work contained in this thesis is my original work, and has never been submitted, in part or in its entirety, at any University, for any requirements towards the achievement of any degree.

B. van der Merwe

Abstract

This thesis describes various software technologies implemented, or specifically developed, for the SUNSAT micro-satellite mission.

With the discussion centered on the Mission Operations System functions of Data Handling and Mission Control, particular emphasis is placed on data processing aspects such as the deployed database schema, and the data communications mechanisms implemented as part of the communications protocol stack. Both the groundsystem architecture and the Flight Software are discussed, their constituent components are analysed, and recommendations are made for improvement. Finally, a Unified Information Model for the design and operation of future, integrated satellite groundsystems is proposed, with suitable implementation technologies being identified and introduced.

Opsomming

Hierdie tesis beskryf die sagteware tegnologieë wat geïmplementeer, of spesifiek ontwerp is vir die SUNSAT mikro-satelliet missie.

Met die bespreking gefokus op die Missie Operasionele Stelsel funksies van Data Hantering en Missie Beheer, word daar veral klem gelê op data prosesserings aspekte, soos byvoorbeeld die databasis skema wat ontplooi is, asook die data kommunikasie meganismes wat geïmplementeer is as deel van die kommunikasie protokol stapel. Beide die grondstelsel argitektuur en die Vlugsagteware word bespreek, hul onderskeie komponente word geanaliseer, en aanbevelings ter verbetering word gemaak. Laastens word daar 'n Verenigde Informasie Model voorgestel vir die ontwerp en operasionele werking van 'n toekomstige, geïntegreerde satelliet grondstelsel. Geskikte tegnologieë vir die implementasie hiervan word ook geïdentifiseer en voorgelê.

Contents

Acknowledgements	iv
List of Abbreviations and Acronyms	v
List of Figures	vii
List of Tables	viii
1 Introduction	1
1.1 Statement of the problem	1
1.2 Prior Work	3
1.3 Thesis Scope and Overview	4
2 SUNSAT Mission Overview	6
2.1 Introduction	6
2.2 Terms and Concepts	7
2.2.1 Micro-satellite Industry Trends	10
2.3 SUNSAT Data Communications and Data Processing	11
2.4 Spacecraft Hardware Overview	12
2.4.1 Subsystems	13
2.4.2 Processors and Communication Channels	14
2.4.3 Scientific Payload	16
2.5 SUNSAT Groundstation Overview	17
3 Flight Software Architecture	19
3.1 Introduction	19
3.1.1 Development Environment	21

*CONTENTS**CONTENTS*

3.2	The RTX Scheduler	22
3.3	The SFS File System	24
3.4	Communication Software Payload	25
3.4.1	AX.25 Link-Layer Protocol	26
3.4.2	Secure Satellite Transport Layer (SSTL)	30
3.4.3	Satellite File Transfer Protocol (SatFTP)	32
3.4.4	Packet Bulletin Board System (PBBS)	34
3.4.5	PACSAT	35
3.5	The Command Processing Module	39
3.5.1	The Diary Entry Format	40
3.5.2	Command Reception, Validation and Execution	41
3.6	Conclusion	41
4	Groundsystem Software Architecture	42
4.1	Introduction	42
4.1.1	Whole Orbit Data (WOD) Files	44
4.2	Groundstation Command & Control Client	45
4.3	SatFTP Client	46
4.4	Telemetry Decoding Program	47
4.5	SUNSAT Database	47
4.5.1	The SUNSAT Database Schema	49
4.6	Telemetry Extraction and Visualisation Client	51
4.7	Operational Experience and Recommendations	53
5	Communications Software Critique with Recommendations	55
5.1	Introduction	55
5.2	AX.25 Evaluation	56
5.2.1	First Alternative – AX.25 Version 2.2	58
5.2.2	Second Alternative – TCP/IP and Derivatives	59
5.2.3	Third Alternative – DUAL	61
5.2.4	Fourth Alternative – Advanced Solutions	62
5.3	PACSAT Evaluation	62
5.4	Memory Management	64

6	Towards a Unified Information Model	66
6.1	Introduction	66
6.2	Background Study	67
6.2.1	Future Development Language	67
6.2.2	Data Description Language (DDL)	69
6.2.3	Groundsystem Architecture	69
6.3	Characterisation of a UIM	71
6.4	The Extensible Markup Language (XML)	72
6.5	Mission Control Architecture	73
7	Results and Conclusion	77
7.1	Thesis Results	77
7.2	Future Work	78
7.3	Conclusion	79
A	SSTL Protocol Specification	80
A.1	Constants and Definitions	80
A.2	SSTL Process Registration	82
A.3	AX.25 Connection and Disconnection Procedures	82
A.4	SSTL Challenge and Response Mechanism	82
A.5	Information Frame Exchange	83
A.6	Resynchronisation Procedure	84
B	SUNSAT Database Schema	85
B.1	List of Defined Entities	85
B.2	Entity Definitions	86
C	Global Frame Pool	91
C.1	Global Frame Pool (GFP) Module	91
C.2	Frame Pool Representation	91
C.3	Frame Request Mechanism	92
C.4	Frame Reference Passing Mechanism	92
C.5	Frame Deallocation Mechanism	93
	Bibliography	94

Acknowledgements

I would like to acknowledge and thank the following people who played a special role in the completion of this thesis:

- My fellow code-smiths, Niki Steenkamp and Hans Grobler, for their comradeship and endurance;
- Mr. S. Mostert, my supervisor, for his guidance, advice and ultimately, for succeeding in getting me focused;
- my comrade-in-arms, Gerhard Esterhuyse, for his invariably good and often even relevant advice;
- Ludwig Schwardt, for many a late-night discussion;
- my family for their interest and support;
- Jean Jordaan, for some urgently needed stylistic advice; and
- my good friends, Robbie Engela and Carel Nolte, for their encouragement.

List of Abbreviations and Acronyms

ACK	...	Acknowledgement
ADCS	...	Attitude Determination and Control System
AMSAT	...	Amateur Satellite Organisation
ARQ	...	Automatic Request Repeat
ARRL	...	American Radio Relay League
AX.25	...	Amateur X.25
BER	...	Bit Error Rate
CCITT	...	Consultative Committee in International Telegraph and Telephone
COMMS	...	Communications (Subsystem)
CORBA	...	Common Object Request Broker Architecture
CRC	...	Cyclic Redundancy Check
DARA	...	Deutsche Agentur Raumfahrtangelegenheiten
DBMS	...	Database Management System
DBP	...	Delay Bandwith Product
DDL	...	Data Description Languages
DOM	...	Document Object Model
DSP	...	Digital Signal Processor
FEC	...	Forward Error Correction
FTL0	...	File Transfer Layer 0
GEO	...	Geostationary Earth Orbit (Satellite)
GPS	...	Global Position System
GSM	...	Groupe Spécial Mobile
HDLC	...	High-level Data Link Control
IDL	...	Interface Definition Language
IETF	...	Internet Engineering Task Force
IP	...	Internet Protocol
ISO	...	International Standards Organization
LEO	...	Low Earth Orbit (Satellite)
MCA	...	Mission Control Architecture
MOS	...	Mission Operations System
NASA	...	National Aeronautics and Space Administration

NFS	...	Network File System
OBC	...	On-Board Computer
OSCAR	...	Orbiting Satellite Carrying Amateur Radio
OSI	...	Open Systems Interconnection
PACSAT	...	Packet Satellite
PBBS	...	Packet Bulletin Board System
PBP	...	PACSAT Broadcast Protocol
PFH	...	PACSAT File Header
PID	...	Protocol Identifier
RFC	...	Request for Comments
RTT	...	Round-trip Time
SatFTP	...	SUNSAT File Transfer Protocol
SCC	...	Serial Communications Controller
SOAP	...	Simple Object Access Protocol
SQL	...	Structured Query Language
SSID	...	Secondary Station Identifier
SSTL	...	SUNSAT Secure Transport Layer
SSTP	...	SUNSAT Simple Time Protocol
SUNSAT	...	Stellenbosch University Satellite
TCMD	...	Telecommand (Subsystem)
TCP	...	Transmission Control Protocol
TLM	...	Telemetry (Subsystem)
TNC	...	Terminal Node Controller
UHF	...	Ultra High Frequency
UI	...	Unnumbered Information (Datagram)
XML	...	Extensible Markup Language
VHF	...	Very High Frequency
UIM	...	Unified Information Model
UOSAT	...	University of Surrey Satellite
W3C	...	World Wide Web Consortium
WOD	...	Whole Orbit Data

List of Figures

2.1	<i>SUNSAT in its Operational Configuration</i>	12
2.2	<i>SUNSAT Structural Organisation</i>	13
2.3	<i>Logical Structure of the Satellite Bus and Scientific Payload</i>	15
2.4	<i>SUNSAT Groundstation Topology</i>	17
3.1	<i>OBC Software Structure and Dataflow Interdependencies</i>	20
3.2	<i>The FTL0 State-machine</i>	37
4.1	<i>Groundsystem Dataflow</i>	42
4.2	<i>The SUNSAT Database Schema as an E-R Diagram</i>	50
4.3	<i>A screenshot of the Telemetry Client</i>	52
5.1	<i>Efficiency improvement with the addition of the SREJ mechanism</i>	59
6.1	<i>Service System Architecture – A layered View</i>	74
6.2	<i>A Dataflow diagram centered on Telemetry and associated Services</i>	76

List of Tables

1.1	<i>Summary of Prior Work</i>	3
3.1	<i>UI and S frame construction</i>	28
3.2	<i>I frame construction</i>	28
3.3	<i>Diary Entry Format</i>	40
4.1	<i>Header Format for a WOD File Entry</i>	44
A.1	<i>RC4State - RX4 State Record Definition</i>	81
A.2	<i>ChallRespPkt - Challenge/Response Packet Type</i>	81
A.3	<i>SyncPkt - Synchronisation Packet Type</i>	81

Chapter 1

Introduction

1.1 Statement of the problem

The management of a complex, physically remote scientific instrument can be a daunting challenge. Establishing a reliable and efficient means of communications is the first problem. Given a large set of scientific and engineering payloads, a second problem is that of controlling these in a orderly and manageable fashion. Thirdly, a large body of scientific and engineering data is produced, either as mission results, or as telemetry data. This must be evaluated in order to effectively, and safely, develop and execute the mission plan. The data must be efficiently stored, transferred, organised, archived, distributed and analysed.

Over the more than 40 years of scientific access to the space environment, a wide range of instruments have been developed to carry out missions of varying durations and objectives. Though a large body of knowledge has been produced, both in the form of printed literature and research publications, each mission presents its own unique requirements. Also, given rapid technological development, new design methodologies and engineering practices must continuously be developed.

SUNSAT is an example of a low earth orbit (LEO) micro-satellite. Firstly, this implies that the satellite is only visible for short periods (typically 12-15 minutes), three or four times per day. The second implication is that the satellite hardware has to adhere to very strict design specifications, particularly in terms of weight and power consumption. These restrictions severely limited such design options as the type and performance of the on-board computers, and the bandwidth of the communications hardware. Subsequently,

a heavy burden was placed on the Flight Software to take maximum advantage of these limited resources.

The primary concerns of this thesis are those mechanisms, protocols and procedures that were developed in order to successfully manage the operation of the SUNSAT satellite mission. While the initial focus will be on communications systems, two important, all encompassing functions will receive our foremost attention: Data Handling and Mission Control.

These two functions are defined in much greater detail in the next chapter. They are implemented by a number of Flight Software and groundsystem modules. These will be discussed, analysed and criticised for the purpose of making recommendations for future systems.

Most of the actual development work involved in the completion of this thesis was software development. In this respect the SUNSAT project presented many unique challenges. In addition to all the hurdles that developing non-interactive, embedded software presents, the Flight Software in actuality comprises an operating system for the satellite. With the exception of a few compiler libraries, every instruction executed by the on-board computers were coded as part of the project work. Such systems-programming is non-trivial, and speaks volumes of the abilities of the chronically understaffed group of people (as compared to the hardware team), that were involved with the software project over the years.

Often, throughout the chapters that follow, short introductions will be made to software modules that were not developed as part of the development work that lead to this thesis. These modules are invariably essential to the operation of the software systems involved and will be only briefly discussed, with due references being made, in order to provide a complete and thorough overall presentation.

In many respects, this thesis represents a body of work and study that was aimed at unifying a set of goals, mechanisms and interfaces that was, at first, disparate. As such it has achieved its primary objective, but has also exposed many possibilities for future research and development. In the final chapter, an initial analysis and technological survey will be undertaken in order to propose next-generation systems, specifically for the groundsystem, that will go even further towards achieving the goal of a fully automated micro-satellite Mission Operations System.

1.2 Prior Work

Several theses and publications have been produced on the subject of the SUNSAT Flight Software. Through the efforts exerted in producing this thesis some improvement was made on many of these results, though such improvements were more often than not based on recommendations made by the original contributor.

Table 1.1 summarises the most relevant prior work on which this thesis is founded.

Title	Reference	Summary
<i>Kernel Support for Embedded Reactive Systems</i>	[Ack93]	Design and Validation of the RTX Scheduler mechanism.
<i>The Integration of a Link Layer Protocol on a Satellite Subsystem</i>	[dB95]	Original Implementation of the AX.25 Link-Layer protocol.
<i>Implementation and Integration of Ground and Space Segment Communication Software for the SUNSAT Micro-Satellite</i>	[Boo96]	Implementation and Performance Evaluation of the PACSAT Protocol Suite.
<i>A flexible environment for Software Development targeted for Embedded Systems</i>	[Tri97]	Design and Optimisation of the SUNSAT Embedded Software Environment.
<i>The AX.25 and SatFTP protocols: Design and Implementation</i>	[vdM98]	Re-implementation of AX.25 protocol stack. Design and implementation of SatFTP.
<i>Development of the On Board Computer Flight Software for SUNSAT 1</i>	[Ste00]	Design and Implementation of the Subsystem Managers. Error Detection and Handling in the Flight Software. Housekeeping and Health Monitoring.

Table 1.1: *Summary of Prior Work*

1.3 Thesis Scope and Overview

The development work on which this thesis is based can be categorised into three distinct sets of software, each set roughly corresponding to the phase of the SUNSAT project during which its implementation took place. Very briefly, they are:

Launch Preparation Finalisation of the Flight Software, particularly the Protocol Stack and SUNSAT File System (SFS). Initial Data Management requirements evaluation. Design of the database schema.

Commissioning of Satellite Implementation of the data depository. Design and implementation of data extraction and visualisation tools.

Routine Operation Future technology survey. Second round requirements analysis. Prototyping and experimentation.

Consequently, the subject matter discussed will focus on the results of these activities. In summary, the document structure is:

- Chapter 2 introduces those terms which circumscribe and define a micro-satellite Mission Operation System (MOS) environment. The discussion is presented from the viewpoint of data handling and mission control, and provides a thorough elucidation of the concepts involved. The rest of the chapter provides a brief overview of the SUNSAT satellite hardware architecture and its capabilities and concludes with a similar overview of the groundstation.
- Chapter 3 provides a systematic analysis and discussion of the SUNSAT Flight Software developed within the scope of this thesis. Some of the software modules they depend on are also briefly introduced.
- Chapter 4 continues in the same vein as Chapter 3, but, in turn, covers the groundsystem software architecture.
- Chapter 5 presents a critical and detailed evaluation of select design elements and technologies already introduced in Chapters 3 and 4. The emphasis is on harboring the SUNSAT developmental and operational experience. Several design and technological solutions are presented to address unfulfilled operational requirements.

- Chapter 6 presents a conceptual framework for a future, integrated and network-centric Mission Control Architecture (MCA), by means of analysis. A groundstation topology based on an organisation of functionally distinct network services is proposed. Combined with the introduction of an advanced data-description language, a vision for a Unified Information Model is presented. Some of the key technologies that might realise this vision are evaluated.
- Chapter 7 concludes with a few brief remarks.

Chapter 2

SUNSAT Mission Overview

2.1 Introduction

The SUNSAT micro-satellite was successfully launched as a secondary payload on a Delta 2 launch vehicle on 23 February 1999, by the National Aeronautics and Space Administration (NASA). As the first South African scientific instrument in space, developed almost in its entirety at the Electronic Systems Laboratory (ESL), at the University of Stellenbosch, it represents a momentous technical achievement.

As of this writing, the SUNSAT satellite has been functional and successfully operated for a period of more than 20 months. During this period, a vast body of operational experience has been added to the already expansive list of theses, technical reports and publications that the project members have produced over the course of SUNSAT's design and development.

From the outset, the SUNSAT project presented three mission goals: providing a challenging research program for post-graduate work; stimulating international co-operation and initiating academic and scientific contact; and, stimulating the interest of the South African youth in science and technology through media exposure and schools programs. Given the wide interest and support that SUNSAT has attracted, the project was a success in all these respects.

One of the stated mission goals for SUNSAT was supporting the international Amateur Radio community. This large, knowledgeable and active group have played an essential role in stimulating the initial development of many micro-satellite missions. Due attention will be given throughout to those technologies that will lead to improving support for this goal.

2.2 Terms and Concepts

In 1995 the German Agency for Space Affairs (Deutsche Agentur Raumfahrtangelegenheiten; DARA) funded a study [Mic95] on concepts and structures for space vehicles. Its aim was:

- to compile and analyse operations concepts and methods, management structures, support, hardware and software infrastructure for the operation of satellites,
- to extract and evaluate trends, and
- to give recommendations as a steering tool for DARA.

The results of the DARA study [RF99] is particularly applicable to the analysis of the SUNSAT program, due to its special consideration of the concepts and trends involved in the development of small and low-cost satellites.

The term *Mission Operations System* (MOS) is defined by the study as describing the overall execution framework of a space vehicle project, from initial planning to space operations. In the operations phase a MOS would consist of:

- a groundsystem – i.e. the infrastructure, hardware and software, with which to conduct and support the mission operations;
- an equivalent system on the spacecraft for Command and Control as well as for the exchange of data between the spacecraft, its payload and the groundsystem;
- an operations organisation supplying the necessary personnel and procedures to operate and control the spacecraft.

The MOS thus consists of all the functions required for engineering and operating spacecraft. These functions, listed below, were identified and clearly defined in the cited [Mic95] study. Some of the definitions have been altered and expanded in order to better apply the results of the analysis to the SUNSAT project's own environment and requirements.

Management Combines all necessary activities to define the mission goals, create suitable interfaces, define required procedures and rules and to create the overall mission plan.

Mission Control Consists of all the activities, personnel and the groundsystem to control and operate a satellite. With an autonomous spacecraft environment, it would also consist of the spacecraft command and control systems.

Data Processing Comprises the processing, display, distribution and archiving of telemetry and other data, in the groundsystem. On the satellite it entails the source gathering, initial processing and storage of such data.

Data Transfer and Communications Provides for the tracking, telemetry and command data between the satellite and groundsystem, and the necessary communications support systems.

Orbit and Attitude Comprises all activities to determine and correct the orbit and attitude of the satellite and to generate antenna predictions for accurate satellite acquisition.

Designing the Satellite Bus Comprises the planning activities for the satellite bus with respect to the payload and mission operations requirements.

Planning the Payload Comprises the planning activities for the payload with respect to the satellite and mission operations requirements.

Activity Planning Comprises all activities and procedures involved in generating the mission control sequence and defining the command activities.

Engineering The planning, development, integration, testing and maintenance of the ground and satellite systems.

In addition, the following terms not defined by the study are added here.

Data Handling Combines the activities of, and provisions made by, the functions of *Data Processing* and *Data Transfer and Communications*.

Mission Planning Comprises those activities and systems of *Activity Planning* which interact with the *Mission Control* function to create a command sequence (commonly called the Diary) for the automated operation of the satellite.

Housekeeping Comprises those automated activities on the spacecraft and groundsystem which maintain the healthy state of the systems involved, so as to allow effective *Data Handling* and execution of the *Mission Plan*. Specifically *Housekeeping* on the satellite involves monitoring and managing the satellite bus and capturing and storing status information.

The following important distinction will be made between the concepts of *Command* and *Control*:

Control Refers to an on-going activity of decision-making and resulting action, involving the continuous consideration of input and the issuing of action instructions.

Command A signal for a specific, pre-defined operation to take place. Such an operation consists of one or more action instructions, which may lead to the initiation of control activities. Initial status conditions are often first checked, but a command generally does not transmit back any intermediate results to its issuer, only a final one (typically an indication of the success or failure of the operation).

Command and Control This phrase, which combines the two terms, is specifically used to refer to the automated management capabilities of a satellite and groundsystem. Such capabilities mainly involve the automated execution of the *Mission Plan*.

The above terms are considered to be well suited for describing the Mission Operations System of SUNSAT, and will be used consistently to describe the various functions of the satellite, the groundstation and their associated activities, as these are introduced.

*

2.2.1 Micro-satellite Industry Trends

A number of rather pungent recommendations were also made by the referred study [RF99] and are reproduced here verbatim, as they sum up the current trends in the micro-satellite industry quite succinctly. These remarks will receive due consideration as recommendations are made in the chapters that follow. They are:

- Adopt the “smaller-faster-cheaper-better” concept.
- Involve mission operations as early as possible in the project phases.
- Move away from the “nice-to-have” mentality.
- Introduce the principles of standardisation and commonality.
- Utilise commercial and off-the-shelf technology.
- Move to automatic ground operation.
- Employ fewer ground operations staff.
- Implement autonomous satellites.

2.3 On the relation between SUNSAT Data Communications and Data Processing

The definition provided in the previous section for the term *Data Handling* suggests a strong relation between data communications and data processing. Such an unusual definition warrants some explanation.

The Flight Software developed for SUNSAT can, first and foremost, be characterised as an embedded system. By nature, embedded software must adhere to particularly high levels of reliability and autonomy. Furthermore, as will be discussed in section 3.2, the Flight Software has the secondary characteristic of performing the functions of an operating system for the satellite. The limitations imposed by the the execution platform (limited memory and CPU execution speed, see section 2.4.2), added even more restrictions.

The implications of these remarks are that the Flight Software had to be, firstly, highly efficient and, secondly, highly integrated and functionally complete in order to perform all the tasks demanded of it.

While all the commonly held ideals for application software design imply a high degree of abstraction and generality, the developers of the SUNSAT Flight Software had to follow a low-level design approach – as is characteristic of systems programming. Accordingly, mechanism often dictated design, and abstraction was only considered in terms of achieving strict modularity. For example, as will be noted throughout Chapter 3, nearly all the communications software modules are highly dependent on the exact mechanisms and interfaces provided by those above and below them in the so-called communications protocol stack. Furthermore, the characteristics of the scheduling mechanism employed (see section 3.2) dictated a very specific model for process implementation.

It should now be clear that the specifics of the Flight Software data processing functions – in terms of scheduling, inter-process communications and data transfer and storage capabilities – cannot be separated from those of the the data communications mechanisms involved. Consequently, a significant portion of this thesis (Chapters 3 and 5 in particular) will be spent on analysis and discussion of the communications protocols either selected or specifically designed for the SUNSAT Flight Software.

2.4 Spacecraft Hardware Overview

One of SUNSAT's major goals was to serve as a technology demonstrator. The project originators followed a high-risk approach, developing a fairly complex micro-satellite as their first attempt. The aim was to test as many subsystems and experiments as weight and size restrictions would allow. The designed life of the satellite is 4 to 5 years, with battery endurance being the expected limiting factor [SMM⁺97].

Whilst the satellite's hardware interfaces and subsystem specifications were of considerable importance to the design and implementation of the Flight Software, this subject has been thoroughly examined [Ste00, p.45-81] and only a brief overview of the satellite's hardware specifications is therefore provided in the sections that follow.

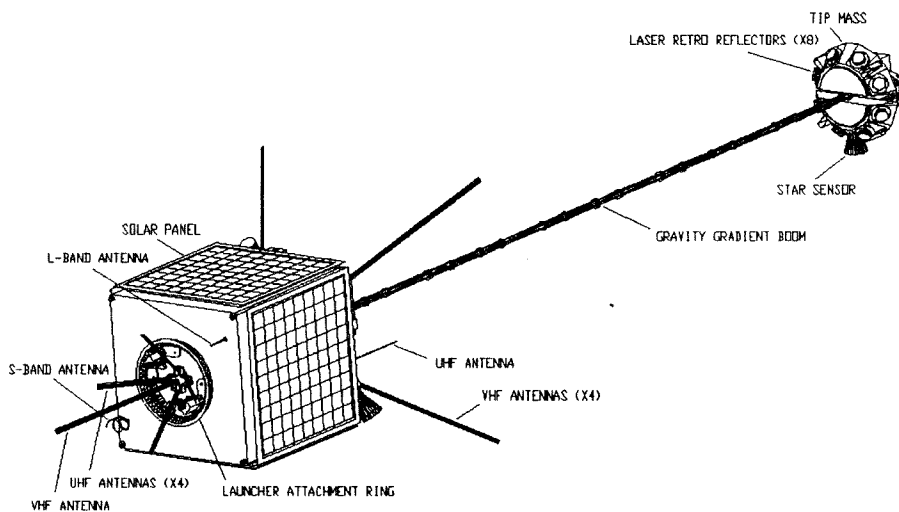


Figure 2.1: *SUNSAT in its Operational Configuration*

The main physical components of the satellite are:

- the main satellite body of dimensions 45 x 45 x 60cm;
- a 2.2m deployable boom;
- a 4.2kg aluminum tip-mass which aids in stabilisation.

Figure 2.1 depicts SUNSAT in its operational configuration with the gravity boom deployed.

2.4.1 Subsystems

The satellite is physically organised into a number of trays. Stacked on top of each other, they form the *Satellite Bus*. With the various electronic subsystems being implemented as physically separate units, two wire harnesses (see Figure 2.3) on opposite faces of the satellite are used for interconnection amongst subsystems and payloads. Figure 2.2 is an exploded view of the satellite with the trays exposed.

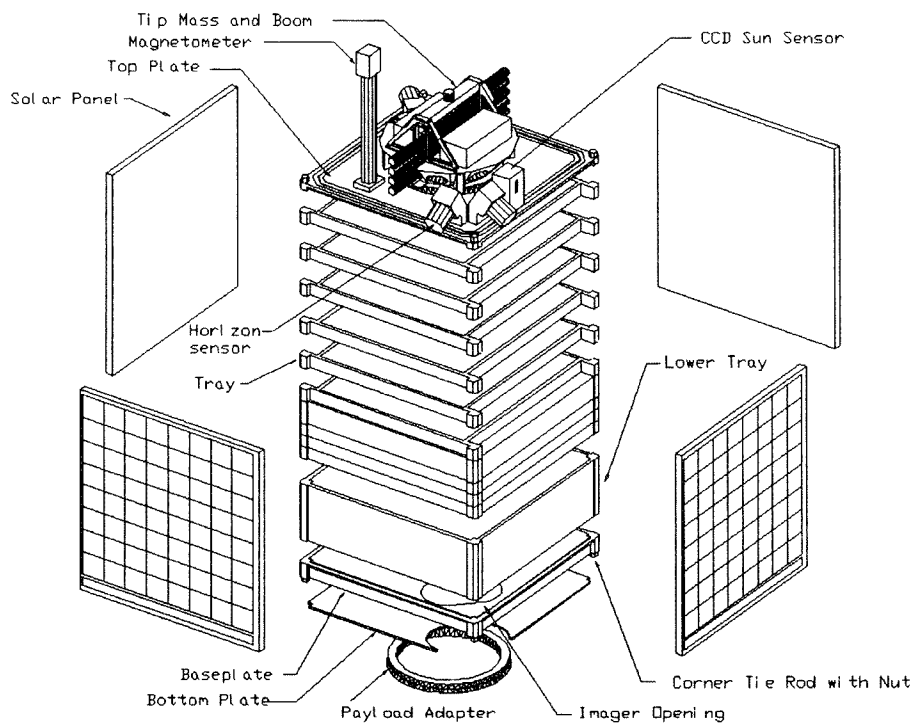


Figure 2.2: *SUNSAT Structural Organisation*

The SUNSAT bus consists of the following subsystems:

ADCS The Attitude Determination and Control System interfaces a number of sensors (including magneto-meters, horizon sensors and a star sensor) with a control processor. The control processor analyzes its sensory input in real-time, in order to dynamically determine the satellite's attitude via a number of actuators (magnetic torque coils edged along the solar panels and a set of reaction wheels).

On Board Computers (OBC1 & OBC2) The next section will discuss SUNSAT's computational capabilities in detail.

Telecommand Controllable from either OBC or the groundsystem, the Telecommand system provides *Mission Control* signal functionality for the other subsystems.

Telemetry A configurable, centralised data collection and packetising processing board which cycles continuously through a number of telemetry channels gathering status and health information from the other subsystems.

Mass Memory (RAMTRAY) Employing static RAM, the RAMTRAY provides a 64MB secondary storage facility to the OBC software. Images taken by the imager payload are also stored here.

Power The electrical power sources on the satellite are solar panels located on four of its sides. These panels are connected by means of a power bus to two battery packs, which in turn consist of five NiCd cells each. Voltage regulation is performed by several distributed regulators.

COMMS SUNSAT's communications subsystem provides for amateur radio communications (VHF, UHF, L and S bands), data downlinking (VHF, UHF and S-Band), data collection (VHF, UHF) and command and control (via interconnection with the Telecommand subsystem). It consists of a number of modems, radios and antennae.

2.4.2 Processors and Communication Channels

OBC1, the primary general purpose computer on SUNSAT, is a 13MHz Intel 80188EC microprocessor. The memory arrangement consists of dual-redundant 512KB SRAM memory banks with Flight Software loaded from 256KB of flash memory. Backup functionality to OBC1 is provided by OBC2, a Intel 80386EX microprocessor. In addition, the ADCS subsystem is equipped with a dedicated T800 transputer. Numerous support processors (mostly 80C31 micro-controllers) perform various other dedicated functions.

The on-board computers, their memory, as well as the bus and payload hardware are interconnected by a number of communication channels:

- The SUNSAT serial bus for Command and Control interconnection between the OBCs and the Telemetry, Telecommand and Power subsystems.
- An instrumentation bus which provides a medium data transfer rate for interconnecting the scientific instruments and the OBCs.

- Dedicated serial links between the OBCs and the ADCS subsystem.
- Dedicated parallel links between each of the OBCs and some of the other subsystems, including the RAMTRAY.

From a *Mission Control* viewpoint the OBCs play a central role as the execution platforms for the Command and Control software, as well as providing *Data Transfer and Communications* and other *Data Handling* functionalities.

The following figure depicts the logical arrangement of the satellite subsystems, communication channels and scientific payload:

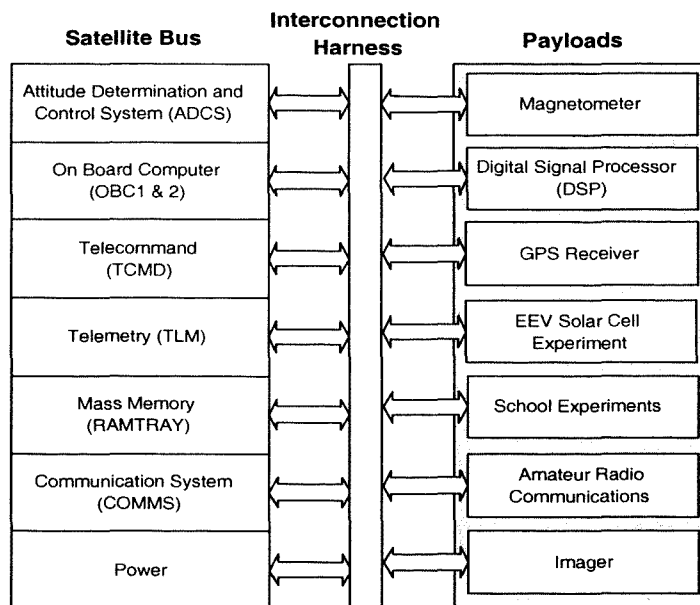


Figure 2.3: *Logical Structure of the Satellite Bus and Scientific Payload*

2.4.3 Scientific Payload

The *Scientific Payload* on SUNSAT consists of seven electronic systems and experiments:

- The primary payload is a 15m/pixel, three-channel push-broom imager. Images are stored on the RAMTRAY and downloaded via the highspeed S-Band transmitter. The camera typically has a 50km wide ground sweep.
- An important secondary mission objective was to facilitate public relations and scientific contact and interchange via Amateur Radio Communications.
- A NASA-supplied TurboRogue GPS receiver and the laser reflectors on the tip-mass allows for experiments in atmospheric, ionospheric and gravity mapping.
- A top-plate mounted magnetometer for measurements of Earth's magnetic field.
- A number of school experiments.
- EEV Solar Cell Experiment.
- A Digital Signal Processor (DSP) experiment.

Each of these experiments are controlled by specialised Flight Software modules. Though most of these modules are dependent, in one way or another, on the software modules which were developed as part of this thesis (for instance the GPS receiver's software updates are uploaded via SatFTP) no further details of the scientific payload will be discussed. Both the hardware and software specifications are considered outside the scope of this thesis. For more information on the scientific payload interfaces, refer to [Ste00, p.15-17, p.131-138].

2.5 SUNSAT Groundstation Overview

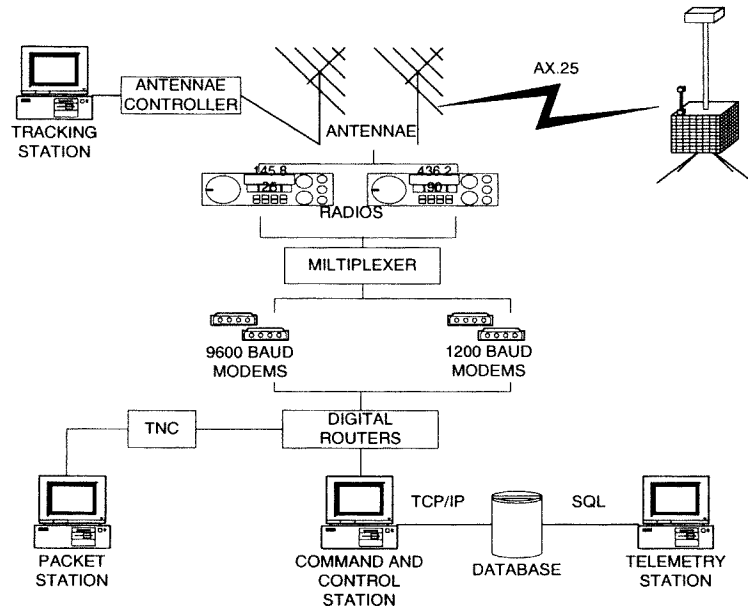


Figure 2.4: *SUNSAT Groundstation Topology*

Figure 2.4 provides an overview of the existing SUNSAT groundstation's topology.

The primary components of the SUNSAT groundstation, which implements the groundsystem, are:

Tracking Station Interfaces with the antennae controller to facilitate acquisition and subsequent tracking of the satellite as it enters a groundstation's footprint. This functionality is supervised by orbit prediction software.

Multiplexer A complex, programmable serial bus device which allows for the configurable routing of analog data between multiple radios and modems.

Command and Control Station A console which provides for the centralisation of all the configuration and communication activities on the groundstation, in addition to providing for the transmission of commands and control sequences to the satellite. Together, this station implements the *Mission Control* and *Data Transfer and Communications* functions of the groundsystem. The functionality provided by this station is

(as of this writing) not automated and thus necessitates the involvement of a human operator.

Database Implements the data depository. The database schema implemented allows for the storage of a variety of datatypes from a range of satellite and groundsystem data sources. Examples of such data include telemetry, status and error information and captured scientific data.

Telemetry Station Implements the *Data Processing* function. User-friendly software allows for the selective extraction of all data stored in the data depository, and its visualisation and exportation to third-party software.

Packet Station The station interfaces with a Terminal Node Controller (TNC) and provides the operator with an interface to the ground-segment software of the PACSAT protocol suite. Broadcasted packets from PACSAT satellites can be captured here and such spacecraft facilities as a Packet Bulletin Board System (PBBS) and file uploading (via FTL0) can be accessed.

Chapter 3

Flight Software Architecture

3.1 Introduction

The structure of the SUNSAT Flight Software architecture has been repeatedly analysed [Boo96, p.17-19] [MvdM99] [Ste00, p.38-42] and slightly varying sets of descriptive categories, as determined by functional classification, have been proposed. For ease of reference, the terminology of [Ste00] will be used. These terms are:

- Navigation;
- Housekeeping and Health monitoring;
- Subsystem Management;
- Payload Management;
- Command Processing; and
- Communications.

Besides attitude adjustment (controlled by the ADCS subsystem), SUNSAT has no other Navigation functionality and this topic is therefore of little interest to us. Housekeeping and Health Monitoring [Ste00, p.89-98] as well as Subsystem and Payload Management [Ste00, p.45-80] have been thoroughly examined.

Though the Communications function has received considerable attention in the past (see [dB95] [Boo96] [Ste00, p.40-41]), advances in the implemented code-base (including revision and even re-implementation) still provides fertile ground for discussion. Important

Communications components (such the Secure Satellite Transport Layer) have also received scant attention. Command processing has been previously examined [Ste00, p.82-88], but its importance to the Mission Control function of the satellite warrants an introduction.

In addition to these functions, which are by and large particular to a satellite environment, the following more general, though still essential, software components will be discussed:

- The Run-time Executive (RTX) process scheduler and dispatcher.
- The SUNSAT File System (SFS) which provides an interface to the secondary storage facility on the satellite.

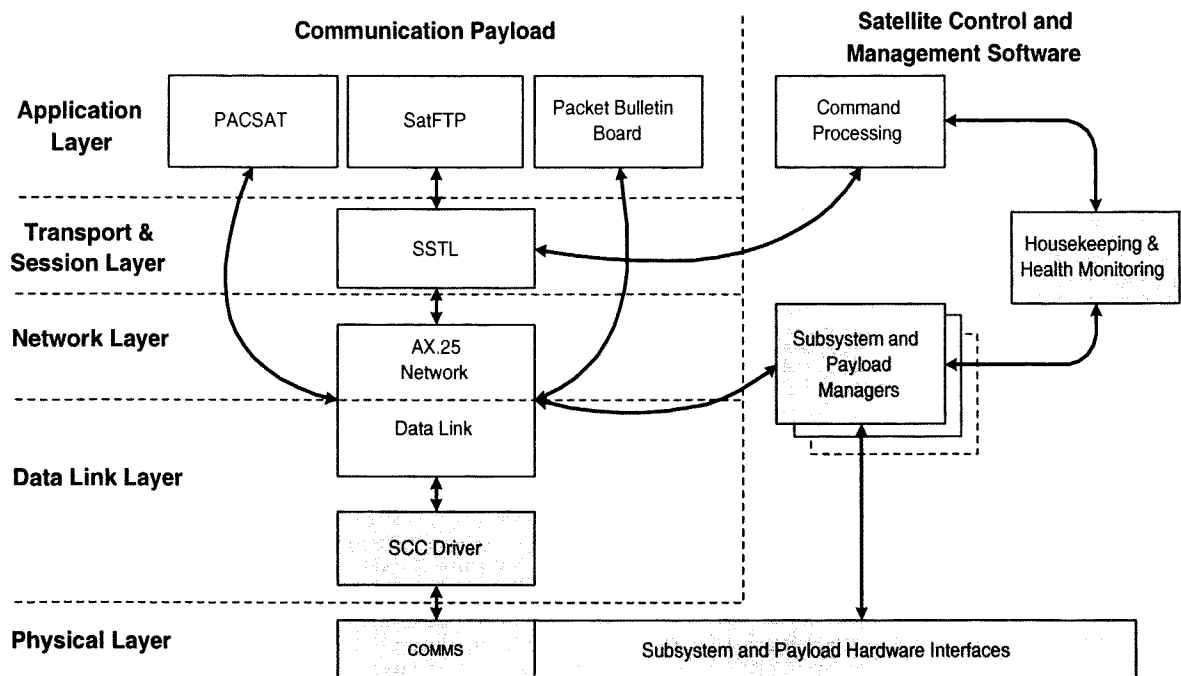


Figure 3.1: OBC Software Structure and Dataflow Interdependencies

Figure 3.1 depicts the logical organisation of the SUNSAT Flight Software as designed and developed for execution on the OBCs.

Those functions that will receive attention in this section have been clearly indicated. The dataflow interdependencies between the modules have also been indicated by connecting arrows. The captions on the left of the figure relates the SUNSAT communications protocol

stack to the widely referenced International Standards Organisation's (ISO) Open Systems Interconnection (OSI) model [Sta94, p.437-450].

Figure 3.1 serves as an apt illustration of the interdependent nature of the relation between the data processing and data communications functions, as discussed in section 2.3.

3.1.1 Development Environment

With the exception of the Serial Communication Controller (SCC) and a few other optimisations that were implemented in assembly language, the SUNSAT Flight Software in its entirety was developed in Modula-2. A member of designer Niklaus Wirth's Pascal family of computer languages, Modula-2 is high-level, strongly typed and procedural. As its name implies, it is a highly modular language with clear separation between interface and implementation. As such, it was one of the first computer languages to implement this new methodology first published in a classical paper on system composition [Par72].

The choice for this particular language was based on the following considerations:

- Members of the software team were already familiar with the language, and retraining would have entailed a considerable investment of time.
- The language's case-sensitive and spartan syntax renders it highly readable, as well as accessible to those who are not advanced in Computer Science.
- Modula-2 is strongly typed. This aspect reduces the possibility of mistakes entering such operations as variable conversion and assignment, since datatypes are explicitly declared and not readily interchangeable.
- The language is highly modular with explicit interfaces which facilitates the adherence to such programming principles as implementation hiding and componentisation.
- A strong consideration was the support that existed in a readily available development system for embedded software (namely, the Topspeed compiler and editing environment). Such support allowed, for instance, for customisation of the library modules which enabled better control over the nature of module initialisation as well as efficiency improvements.

- The available development system also included built-in support for the development of, and linkage to, assembly language routines.
- In addition, Modula-2 is considered well-suited for development of real-time software [BW93].

Substantial criticism has been leveled against the continued use of Modula-2 for the next generation of Flight Software. This includes:

- Both Modula-2 and its development systems have been showing their age for quite some time and are generally considered antiquated by software developers. Further use of the language may leave programmers frustrated and unmotivated.
- Modula-2 lacks most modern language features such as language constructs for exception handling, derivative type declaration and thread support.
- Unlike its younger siblings (Oberon in particular), Modula-2 has poor memory-management support, and is therefore prone to memory leakage.
- Standard library function support is weak compared to modern standards and there is no support for common data-structures.

3.2 The RTX Scheduler

The SUNSAT Flight Software does not contain an operating system as it is generally defined [Dei90, p.3]. A more reasonable perspective would be to regard the *entire* set of modules, with the exception of the communications software applications (PACSAT, PBBS and perhaps SatFTP), as the satellite's operating system. These modules together perform all the functions of communications, secondary storage management and device handling usually assumed by the OS. Two crucial functions, however are still not covered: that of scheduling and task dispatching.

Scheduling provides for the selection for execution of a single process from amongst all those that are currently idle, but eligible for execution. This is done according to some scheme judged fair and efficient [Dei90, p.287]. Task dispatching performs the function of passing the point of execution from the currently running process to the newly selected process.

The scheduler and task dispatcher selected for the SUNSAT Flight Software is called Run-Time Executive (RTX) [Ack91]. The choice of this particular solution was based on RTX's adherence to the several requirements set beforehand, particularly that of real-time execution. For a complete discussion of the criteria and concepts relevant to embedded real-time systems, refer to [Ste00, p.21-26].

RTX implements a form of co-operative or non-preemptive scheduling where the task dispatcher does not actively stop execution of the current process. Execution is voluntarily relinquished. More specifically, RTX implements earliest-deadline-first scheduling, an optimal scheduling policy. This means that if a feasible schedule exists for a certain system, earliest-deadline-first scheduling will ensure that all tasks always meet their deadlines. A feasible schedule exists for a set of processes if their execution cycles can be ordered in such a way that all deadlines are met (from [Ack91, chap. 1]).

RTX provides application constructs and interfaces for the creation of the following:

- *Processes*, which are the main programming construct on which RTX scheduling is based. All the SUNSAT software modules are implemented as individual modules, which are in fact Modula-2 procedures registered with RTX as processes. Processes communicate with each other by placing messages in channels or mailboxes and by setting timers or alarms. Thus, a basic producer/consumer model is followed.
- Single-slot *channels*, when the consumption of messages is guaranteed to be faster than their production.
- *Mailboxes*, which can receive a configurable number of messages.
- *Ports*, for the triggering of events when *signals* are set on them.
- *Timers*, which notify their expiry on *channels*.
- *Alarms*, which notify their expiry on *mailboxes*.

The notion of time is very important to RTX and to the operation of the satellite in general [Ste00, p.83-84]. When a port, channel or mailbox is created, a period is specified which determines the maximum period that any message may remain "alive" before the execution of its intended process must take place. If these deadlines are continuously reachable, a RTX schedule is considered feasible.

Two fundamental constraints are placed on RTX processes:

- Firstly, they must always terminate. RTX can only schedule the next process once the last one has terminated; and
- secondly, blocking on receive is not supported. A process can only receive one message per execution cycle.

According to [Ste00, p.123] the full Flight Software at the time of writing consisted of 112 RTX processes, 125 message channels (channels and mailboxes) and 62 timers.

Please note that the implementation and the maintenance of RTX falls outside the scope of this thesis. For more information on the theory and validity of the RTX scheduler, refer to [Ack93]. For programming details on RTX, refer to [Ack91] (currently outdated). The RTX code itself is exceptionally well documented and contains numerous comments in both its interface and implementation modules.

3.3 The SFS File System

The SUNSAT File System (SFS) forms the foundation of all Data Handling activities on the satellite. Based on the UNIX inode-style file system [Dei90, p.576-578], SFS allows files to be created and accessed according to a hierarchical scheme employing textual filenames and directories.

The SFS module provides for the creation of a file system in either unallocated high-speed OBC RAM or on the slower 64MB RAMTRAY subsystem (see section 2.4.1). Existing file systems can also be mounted and unmounted. This allows for some operational flexibility, particularly for periods when a OBC is temporarily switched off.

During the groundwork for this thesis, some optimisation was performed on the SFS module, particularly on the *ReadBytes* and *WriteBytes* routines. A function for the calculation of file content CRCs was added in order to support PACSAT. Memory usage was reduced by only allocating file handles when they are initialised, and not upon declaration. Initial work for the mount and unmount procedures was also performed.

An interesting addition to the SUNSAT Data Handling interfaces, was the development of the File System Manager (FILE_MAN) module as documented in [Ste00, p.78-80].

FILE_MAN performs the dual purpose of, firstly, serving as a simplified interface to the SFS module and, secondly, adding such efficiency measures as a limited write-cache and file segmentation. FILE_MAN is for instance used by the Telemetry subsystem manager for storing telemetry data in WOD files (see section 4.1.1).

The write-cache involves buffering all write operations to file systems on the RAMTRAY subsystem into 1KB blocks. Since the data bus between the OBCs and the RAMTRAY subsystem is particularly slow, by buffering write operations into 1K blocks, as employed by the SFS file system for read and write access to its file systems, fewer file accesses are performed and performance is greatly improved. The FILE_MAN module can also segment data into multiple files for easier downloading during multiple groundstation overpasses.

Given the critical nature of SFS, both in terms of the overall flight system performance and for data integrity, some work is on-going for adding caching and better data integrity tests to SFS, as part of a general revision of the code. This work is being done outside the scope of this thesis.

3.4 Communication Software Payload

Space communications is a very exciting and challenging field to be involved in. It combines all the intricacies of protocol design with the special challenges posed by the space environment. The field has therefor in the past drawn such an respected authority from the fields of Networking and Communications as Vint Cerf [Cer82].

The communications software for SUNSAT was developed over the course of a number of years and, as such, it is the result of numerous revisions in design and implementation. Together comprising a so-called protocol stack, the communications software implements the Data Transfer and Communications functions of the Mission Operations System (refer to section 2.2) on the satellite. An equivalent protocol stack is implemented as part of the groundsystem.

The architecture consists of the following distinct components (also see figure 3.1):

SatFTP, PBBS, PACSAT These are the top-layer service applications. SatFTP (a file transfer protocol) was specifically developed for SUNSAT. PBBS is a simple and standardised packet-radio bulletin board service. PACSAT, though not operational

in the current Flight Software configuration, would provide a packet-radio store-and-forward service.

SSTL The Secure Satellite Transport Layer provides restricted access to the satellite application layer and provides routing of packets based on pre-defined Layer 3 Protocol Identifiers (PID) [BNT97, p.7].

AX.25 AX.25 is an implementation of Version 2.0 of the Amateur Packet-Radio Link-Layer protocol, as documented in [Fox84].

SCC Driver The SCC layer provides the error-detection and frame encoding/decoding functions, as specified by the AX.25 protocol [BNT97, p.8], in addition to interfacing with the COMMS subsystem.

The specification, design and implementation of these components will receive detailed attention in the following sections.

3.4.1 AX.25 Link-Layer Protocol

The AX.25 Amateur Packet-Radio Link-Layer Protocol specification [Fox84] and its revisions [BNT97] defines a protocol for use between two amateur radio stations in a point-to-point or networked communications environment. The protocol specifies only link-layer and physical layer functions. It was not intended to specify any upper-layer (layer 3 or higher, see 3.1) protocols.

The protocol conforms to the ISO Information Standards (IS) 3309, 4335 and 7809 High-level Data Link Control (HDLC). It conforms with ANSI X3.66, which describes ADCCP, balanced mode. It follows the principles of the CCITT Recommendation Q.920 (X.25), with the exception of an extended address field and the addition of an Unnumbered Information frame. It also follows the principles of CCITT Recommendation Q.921 (LAP-D) in the use of multiple links, distinguished by the address field, on a single shared channel.

Development of the AX.25 standard

The AX.25 protocol version 2.0 [Fox84] was originally published in 1984 by the ARRL (no reference could be found to earlier versions). Due to the rapid growth in amateur packet-radio communications that it stimulated, the limitations of the already revised protocol was

soon apparent. Poor performance under conditions of high frame loss, as well as weaknesses in the protocol specification itself (particularly with regard to standards for interfacing with other protocol layers) counted among the most pressing problems.

A serious effort towards updating version 2.0 was published in 1988 at the 7th Annual Computer Networking Conference [Sca88]. Acceptance of its proposals did not follow immediately and version 2.2 of the protocol [BNT97], mostly the work of the Tucson Amateur Packet Radio Corporation (TAPR), was only published by the ARRL in 1996. As of this writing, the fourth revision (version 2.2), published in November 1997, is considered current.

A Brief Technical Introduction

Only version 2.0 will be discussed. For more information on version 2.2, refer to section 5.2 and [BNT97].

Link-layer packet radio transmissions take the form of small blocks of data called frames. Three general types of frames are defined for AX.25:

Information frame The I frame is the regular application information transport frame.

Supervisory frame S frames provide supervisory link controls such as acknowledging or requesting retransmission of I frames, and flow-control.

Unnumbered frame U frames are used for the establishment and termination of links.

Unnumbered information frames (UI frames) are also defined and used for the unreliable broadcast of information outside the operation of the flow-control mechanism.

Each frame consists of several smaller data groups, called fields. Tables 3.1 and 3.2 illustrate the three basic types of frames, with the leftmost bit being the first that is transmitted.

Flag	Address	Control	FCS	Flag
01111110	112/560 bits	8 bits	16 Bits	01111110

Table 3.1: *UI and S frame construction*

Flag	Address	Control	PID	Info	FCS	Flag
01111110	112/560 bits	8 bits	8 Bits	N*8 bits	16 Bits	01111110

Table 3.2: *I frame construction*

An explanation of each field's purpose follows:

Flag The flags are used to delimit each frame. Bit stuffing is employed to maintain the unique bit pattern (7E hex) used.

Address AX.25 uses regular Amateur Radio callsigns for addressing. Additional secondary station identifiers may be added to allow up to 16 stations per callsign. The SUNSAT Flight Software employs the SSIDs to address individual applications services (such as PBBS and SSTL). In addition, the address fields (both source and destination addresses appear in each frame) serve to identify command or response frames and for level 2 repeater operation (phased out in version 2.2).

Control The control field is used to identify the type of frame and determines several attributes, such as the frame sequence number (for flow-control) in I and S frames, and indicating the poll (command) or final (response) status for all frames. The poll mode is used to request an immediate reply to a frame. The reply to a poll must have the final bit set.

PID The protocol identifier indicates the layer 3 protocol in use. It is used by SSTL to route received frames to the correct destination application process.

Information The I frame, UI frame (unnumbered information) and FRMR frame (error response) may contain up to 256 bytes of information.

FCS The frame check sequence allows for error-checking and thus data link integrity. It is a 16-bit value calculated in accordance with the ISO 3309 (HDLC) recommendation.

Flow-control is performed in all S and I frames. Up to seven unacknowledged information frames are allowed in either link direction.

The specification defines three timers. An acknowledgement timer (T1), a response delay timer (T2) used for improving channel efficiency, and an inactive link timer (T3) used for maintaining link integrity when T1 is not set.

The SUNSAT implementation deviates from the specification in only one respect: namely, that an interface is provided for the broadcasting of unnumbered information (UI) frames in the absence of an associated connection. This is used by PACSAT and other application processes for the broadcasting of status and other telemetry information.

Notes on the SUNSAT Implementation History

The original SUNSAT AX.25 implementation completed development in early 1995 with the publication of [dB95]. While this first attempt at implementing a fairly complex protocol served as an adequate basis for the development of the high-layer protocols required for the mission, it was deemed unsuitable as a component in the final Flight Software. Firstly, it deviated from the published specification to the extent that it would not interoperate with implementations in other packet-radio systems. Secondly, the higher and lower layer interfaces were not well designed and significant efficiency improvements could be made, some of which were suggested by [Boo96, p.65].

A redesign and implementation followed [vdM98]. Care was taken to develop interfaces that were both efficient and served the functional requirements of the high-layer protocols adequately. Additional interfaces were also added to allow for better error-handling and the provision of link status information.

The current implementation has undergone more changes from the one documented in [vdM98], particularly in refining the interfaces, simplifying the implementation, and in optimisation work. An example of such an optimisation is the ability to add so-called raw frames to an AX.25 transmission buffer. This is used by the SSSL layer, since most of the headers in information packets sent by SSSL can be pre-assembled.

It should be noted that none of the version 2.2 enhancements have as yet been implemented (for a discussion, refer to section 5.2). Version 2.0 has been implemented in its entirety.

The continued suitability and efficiency of AX.25 has been questioned [MTB95, esp. p.5-6] [Kar94, p1-2]. These issues, and possible alternatives to AX.25, are covered in section 5.2.

3.4.2 Secure Satellite Transport Layer (SSTL)

SSTL performs three primary functions, corresponding to its role as a transport layer protocol [Sta94, p.544-554]:

Addressing Though actual station addressing is provided by AX.25, SSTL does implement an authentication mechanism as a form of user identification in addition to transport entity identification (via AX.25 Protocol Identifiers (PIDs)).

Multiplexing All processes registered with SSTL share a single AX.25 link, with SSTL performing the necessary routing of packets to their correct destination processes.

Connection Establishment and Termination SSTL registers a link with AX.25 and manages the connection, notifying its registered processes when a connection has been established and again when it is terminated.

A final function specified by the OSI for a transport layer protocol is that of flow-control. AX.25 deviates from the OSI in this regard, by implementing flow-control as part of a link-layer protocol (along with specifying the frame bit encoding method – a physical layer function). SSTL therefore performs no flow control.

In addition to serving as a unifying interface to AX.25 for the communications software application layer, SSTL's primary role is the enforcement of security policy. [WL99, p.582] cites the ease with which a link may be intercepted by an unauthorised user as a characteristic of space-ground communications. It is warned that control of a satellite might even be taken over by re-transmission of previously intercepted commands. Data encryption is offered as a technique to avoid such problems.

Consequently, SSTL provides a secure authentication system for restricting connection access to the SSTL-supported communications software application layer, as an initial measure. Since PBBS and PACSAT depend directly on AX.25, only SatFTP, the OBC log server and the Command Processing module (see section 3.5) currently make use of this functionality. As an additional measure, SSTL also allows for the selective encryption of

individually registered packet types. The addition of encryption is particularly important to guard against session hijacking and packet insertion, common network-attack techniques.

In summary, SSTL provides the following services to the applications layer software:

- Connection management and notification.
- Packet and process registering for link sharing and packet routing.
- A secure authentication mechanism for access restriction.
- A selective encryption service for additional security.

The next two sections will discuss the authentication and encryption mechanism employed by SSTL. A complete specification appears in Appendix A.

The Secure Authentication Mechanism

Immediately upon the establishment of a AX.25 connection addressed to the SUNSAT SSTL process (determined by callsign and SSID), the SSTL server transmits a challenge to the connecting party. This challenge consists of a randomly chosen token, a random sequence number and the random initialisation seed for a stream cipher. The challenge is encrypted by a secret key.

The client must now know the correct secret key to decrypt the challenge in order to reply with the exact same token and its own stream cipher initialisation seed, encrypted by a different secret key.

Upon decrypting the response, the SSTL will check that the correct token was received back, thus authenticating the client. Each party now initialises two different ciphers streams using the exchanged seeds – one stream for encrypting the packets it intends to transmit and one for decrypting the packets it receives.

Given the random nature of all the initialisation values (produced on the satellite by noise collection in the SCC driver) the authentication mechanism is guaranteed to remain secure, as long as the secret keys shared by both parties remain secure. New sets of secret keys are typically generated before each upload of updated SUNSAT Flight Software to the satellite.

The Encryption Mechanism

After completion of the challenge-response sequence, both client and server can begin transmission of packets. Due to the high computational cost of encryption procedures, all packets exchanged are not encrypted. SSTL forces its registered processes to also register and identify (by PID) all the packet types they intend to transmit. This procedure also allows for specifying whether a particular packet type should be encrypted. SatFTP, for instance, as performance measure only registers its command packets for encryption, while its data packets are transmitted in the clear.

In addition to being run through the stream cipher, all encrypted packets are first assigned an incrementing sequence number. The sequence number allows for defending against denial-of-service attacks when random packets are inserted into an existing encryption stream, possibly leading to software failure. The decrypted sequence number is first checked by a receiving party before the packet is routed to the correct parent process. In addition, resynchronisations are performed every time a fixed number of packets have been transmitted. Newly generated seeds and sequence numbers are sent, once more encrypted by one of a set of available secret keys.

The stream-cipher employed by a SSTL is a 128 bit implementation of the widely trusted and deployed RC4 algorithm [Sch96a, p.397-398]. RC4, for example, is implemented as part of the Transport Layer Security (TLS) protocol [DA99] that secures HTTP connections in modern web browsers. The American Cellular Digital Packet Data specification [Com93], also specifies RC4 as its stream cipher. It should be noted that SSTL was largely conceived by following the mechanisms employed by the TLS specification.

3.4.3 Satellite File Transfer Protocol (SatFTP)

With the advent of mass-storage on micro-satellites, the volume of flight data being handled has increased substantially. Experience has taught that the management of such data can be vastly simplified if they are treated as files [KG98]. Additional operational benefits are gained by the availability of a generalised upload facility, avoiding the redundant implementation of specialised interfaces. The SatFTP protocol was developed to work in conjunction with SFS as the primary Data Handling mechanisms of the SUNSAT Flight Software.

Development of SatFTP was deemed particularly crucial given the difficulties foreseen if

relying solely upon the PACSAT protocol for SUNSAT's Data Handling requirements. The SFS file system provides sophisticated interfaces for the manipulation and management of files and directories for the organisation of mission data. It was therefore considered imperative that an appropriate communications service be developed to exploit the full functionality the SFS facility provides.

Just as SSTL is conceptually based on the TLS protocol, SatFTP is based on the familiar file transfer protocol (FTP) [PR85]. Two other protocols, specifically designed for the packet-radio environment, were reviewed (see [Jac86] and [PWWL96]). These were found to be inappropriate for several reasons, including their lack of general acceptance and the desire to develop interfaces tailored for the SUNSAT environment, in particular SFS.

The same reasons hold for not relying on the PACSAT FTL0 protocol. On many PACSAT satellites, only the upload function of the FTL0 protocol have been implemented. Refer to section 5.3 for a complete discussion on why PACSAT is considered inappropriate.

The SatFTP protocol provides for the following Data Handling operations:

File Transfer The simultaneous up- and downloading of specific file data is supported. Uploaded data may be appended to existing files, and downloads may start from a specified file offset. Transfers in progress may also be canceled.

File Management The size of files may be queried and files may be permanently removed.

Directory Management Directories may be created and removed. The current active directory may also be specified. The contents of a particular directory can also be stored within a file for download and display.

File System Status SatFTP operations may be performed on any file system image (located either on the RAMDISK or OBC RAM). The status operation returns information on the current selected file system and its free space.

Protocol Settings All SatFTP commands are encrypted by the SSTL layer. In addition, the option exists for file data transfers to be encrypted as well. It is also possible to set an option to delete partially uploaded files.

Fully automated operation was a primary objective in the design of SatFTP, along with efficiency and reliability. Though the ground-based client is currently implemented as part

of an operator console, the fully scripted operation of the protocol is assisted by detailed error responses and other design elements, such as status command and response.

The design of File Transfer Protocols in the space environment is an increasingly active field. Specialised research, such as adaptation to deep space environments, is being pursued [Gre98]. SatFTP is conceptually independent of any lower-layer protocols, which renders it easily adaptable to other environments. Such an adaptation of SatFTP to the GSM environment has already been performed [Car99]. In summary, SatFTP has performed well operationally [GM99, p.13], and compares well with other current-generation systems of its kind [KG98].

Notes on the SUNSAT Implementation History

The SatFTP protocol was first proposed and specified in [vdM98]. It has since undergone several revisions, extensions and an entire redesign and implementation. During the most recent redesign, the implementation was divided into the following distinct code sections:

Link Control and Packet Decoding SSSL notifications, the decoding of command packets and the reception of uploaded data packets are performed in this section.

Download Scheduler The transmission of data packets during downloads is performed here.

Command Handler The execution of the correct command execution procedure is determined here.

The most recent revision has mostly involved simplifying the actual packet encodings used. The original SatFTP implementation also allowed for multiple simultaneous connections and buffered file transfer requests. Both these functions were considered non-essential and were subsequently removed in order to limit unnecessary complexity, and thereby potentially improve the reliability of the software. Functionality such as command buffering should in any case form part of the client, and not the server, specification.

3.4.4 Packet Bulletin Board System (PBBS)

Though the Packet Bulletin Board System (PBBS) module was developed outside the scope of this thesis, it is discussed here as it completes the set of application modules in

the SUNSAT communications protocol stack.

PBBS is a very simple client-server protocol which allows for the storage and retrieval of short messages based on the recipient's Amateur Radio callsign. Messages are kept for a limited period and can also be manually deleted. PBBS allows for a number of simultaneous connections and is quite popular with Amateur Radio operators. It is directly dependent on AX.25 as link-layer and is therefor assigned its own SSID.

3.4.5 PACSAT

Though PACSAT has become a generic term used to describe a digital store-and-forward satellite mission in the Amateur Radio Service, it is more specifically used to refer to a particular protocol implementation which provides such a service.

The first ever satellite communications was via a store-and-forward relay using a tape recorder on the 1958 SCORE satellite. During the 1960s and '70s, such techniques were nearly completely abandoned in favour of the newly developed Geostationary Earth Orbiting (GEO) satellites. In 1973, the MITRE Corporation published a paper which proposed a "message courier" satellite embodying many of the concepts that were eventually implemented a decade later. The sudden availability of personal computers in the late 1970s, combined with the continued popularity of Amateur Radio, lead to a rebirth of interest in store-and-forward-capable satellites.

A second group with a keen interest in this technology stimulated the development of civilian store-and-forward satellites, namely relief and development agencies, particularly those working in developing countries [War90].

Soon universities and amateur satellite groups were building and operating increasingly sophisticated satellites, with much of the effort being led by the ARRL and the Spacecraft Engineering Research Unit at the University of Surrey (home of the UoSATS). This trend continued throughout the 1980s and '90s, and has recently taken another new leap forward with the successful launch of the Phase-3D satellite by the American AMSAT group.

The most widely used store-and-forward protocol is the so-called PACSAT suite [PW90b]. First launched as the communications software payload on the UoSAT-3 satellite in January 1990 [War90], it has become the de facto standard for amateur satellite packet-radio com-

munications.

The PACSAT suite contains three major elements:

- an encapsulation format (PACSAT File Header, PFH) [WP90a] prepended to all PACSAT files;
- a point to multi-point protocol for downloading (PACSAT Broadcast Protocol, PBP) [PW90a]; and
- a point-to-point protocol for up- and downloading, as well as content broadcasting (File Transfer Layer 0, FTL0) [WP90b].

A Brief Technical Introduction

A more thorough discussion can be found in [Boo96]. The following is only intended as background to a critique of the protocol in section 5.3.

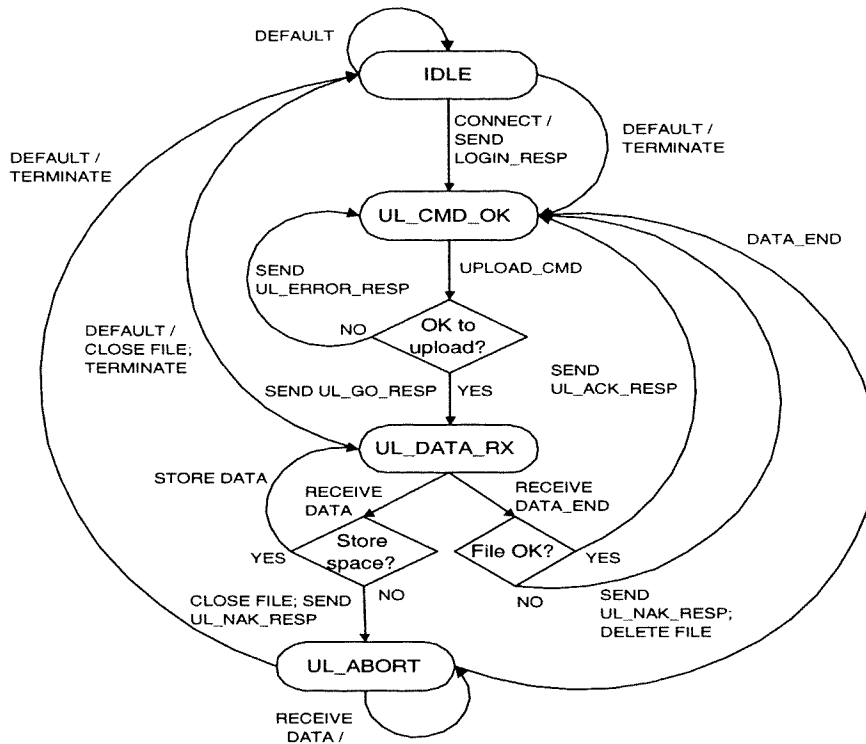
PACSAT was developed in response to the problem of maximizing the available bandwidth of the communication system to a LEO satellite when providing a multiple-access store-and-forward service. The solution was to follow a dual approach. FTL0 is a single-access connected-mode protocol for the up- and downloading of files, while PBP is a broadcast protocol which broadcasts the contents of files according to requests received on a competitive basis.

A PACSAT implementation keeps a directory listing of all files uploaded to the satellite or added by software processes. Each file is given a unique file handle (a 32 bit identifier). The PBP protocol continuously performs directory broadcasting, enabling listening groundstations to determine the appearance of new entries since the last satellite pass. Groundstations can then transmit broadcast requests to the satellite.

Additional, separate, broadcasts indicate the status of the PBP software (such as when the request queue is full) and also allow listening groundstations to determine whether their requests have been accepted into the request queue or whether they should be resent.

Broadcast requests may take three forms:

- A request for the broadcasting of particular file's contents to be started or stopped.

Figure 3.2: *The FTLO State-machine*

- A request for a list of specific frames from a file (called a hole list) to be re-transmitted.
- A request for specific ranges from the directory to be broadcasted.

Request responses are broadcasted in response to these, indicating the success of the operation which depends on the validity of the request and the availability of an open slot in the request queue. In addition to these file data and directory broadcasts, updates to the specification also specifies how status broadcasts are to be performed.

FTLO is a slightly more complicated connection-based protocol which can be implemented as state-machine, following the more or less exact pseudo-code provided in the specification. In addition to file transfers, it also provides a search capability; however neither this nor any download functionality has been implemented for SUNSAT. Figure 3.2 depicts the FTLO file upload state-machine as a flowchart diagram.

For their operation, both PBP and FTLO depend on the existence of a PFH for all files in the PACSAT directory. The PFH contains a number of mandatory and optional fields

describing such parameters as the name of the file, its type, its date of creation and directory addition, a more accurate length description, and so forth.

Notes on the SUNSAT Implementation History

The first design study and implementation of PACSAT for the SUNSAT project was the subject of a thesis published in 1996 [Boo96]. This work also analysed the protocol specification for implementation as RTX processes, exposed the inner mechanisms of the algorithms involved as flowchart diagrams [Boo96, Appendix B], and made several recommendations regarding the optimisation of the SCC and AX.25 modules.

During the time between the completion of this thesis and the finalisation of the Flight Software architecture, the various software modules (see figure 3.1) underwent one or more revisions. The PACSAT implementation was not maintained during this period, and eventually became incompatible with the external interfaces that it depended on.

It was subsequently decided in January 1999 to perform a redesign and implementation of the PACSAT modules. This work was performed as part of the efforts that led to this thesis. The following describes the major changes in the design approach:

- The design of the packet encoding and decoding mechanisms shifted from a imperative to a more declarative approach. Data structures (mainly Modula-2 record types) were employed throughout to map onto all incoming and outgoing packets, instead of executing elaborate decoding and encoding procedures. Existing data-structures were redesigned to allow for more efficient operation and easier code maintenance.
- Suitable application programming interfaces, exposing the functionality of PBP and FTL0 to external processes, were added.
- Functionality implementing more recent enhancements to the specification was added.
- Memory usage was greatly reduced by keeping in memory only those parts of a PFH that are used in directory broadcasts.
- A newer command decoding and execution model, similar to that employed by the revised SatFTP module, was introduced (see section 3.4.3).

PACSAT was originally intended as the primary means of accessing the data (such as WOD files, logs and other output) produced by the Flight Software. With the introduction of SatFTP this role was filled, and it was foreseen that PACSAT would eventually be configured as part of the software payload supporting the mission goal of Amateur Radio Communications. As of this writing, this has not been done for a number of reasons, primarily because PACSAT has yet again fallen into disrepair after not being fully flight-tested and debugged after implementation, due to other priorities at the time.

There are also currently several other PACSAT satellites in operation and this operational role is therefor not considered crucial. In addition, doubts exist as to the viability of effectively operating PACSAT given the fact that the OBC resources are already being fully utilised.

3.5 The Command Processing Module

The Command Processing Module (CPM) has the responsibility of receiving Command & Control frames, either individually from the groundsystem C&C console via SSTL (see section 4.2), or as a single SatFTP uploaded file (commonly referred to as a Diary), and to dispatch the contents of such frames, at the indicated times, to the correct Flight Software application or subsystem manager for execution. As such, the CPM functions as the central Mission Control mechanism on the satellite.

The CPM maintains a linked list of all undispatched C&C frames in separate event records. A special C&C frame, currently sent manually from the groundsystem C&C console, addressed directly to the CPM by its SSTL PID, indicates when a new Diary file has finished uploading and should be processed. The CPM will then parse such a file while checking its integrity, and proceed to add its contents to the existing linked list. The linked list is always kept sorted by the individual event records' timestamp values which determines the correct time for their dispatchment. The header of the C&C frame transmitted from the groundsystem console also contains a field which indicates whether the timestamp values of the Diary entries are meant to be interpreted as relative or absolute. In the case of relative time, the current system clock value will be added to all the timestamp values of the Diary's contents.

The event dispatching mechanism of the CPM always keeps a RTX alarm set to the time-

stamp value of the first entry in its linked list. When the alarm is triggered, RTX will schedule the CPM's event dispatcher for execution. The dispatcher removes the first event record from the linked list and places its contents (the actual C&C frame) into the indicated recipient's RTX mailbox. There are, as of this writing, 24 such mailboxes defined, each addressed to one of the Flight Software's major application or subsystem manager modules.

Should the event be intended for rescheduling, the CPM dispatcher will place the event record back into the list, keeping it sorted by timestamp value, after adding the rescheduling period to the timestamp value. After each rescheduling, the rescheduling count value is decremented, and the event record is only dropped when this value reaches zero.

The Flight Software Diary mechanism was developed outside the scope of this thesis, being first proposed in [IR96]. It has been revised several times. For another discussion on the CPM and the Diary mechanism, refer to [Ste00, p.82-89].

3.5.1 The Diary Entry Format

Name	Size	Purpose
Code	32 bits	Uniquely assigned code
Time	32 bits	Timestamp value (absolute or relative) in seconds
Flags	8 bits	Bit 0 - Auto reschedule if set Bit 1 - Disable logging if set Bit 2 - Error recovery C&C if set
ReschedPeriod	16 bits	Period for rescheduling in seconds
ReschedCount	16 bits	Number of times event should be rescheduled
Recipient	8 bits	Unique recipient identifier
CMDLength	8 bits	Length of actual C&C frame in bytes
CMDBody	8 bits	Used for addressing the first byte of the actual C&C frame

Table 3.3: *Diary Entry Format*

Table 3.3 illustrates the format of a single Diary file entry. Only the header format is shown, since the actual C&C frame format is unique to each recipient module.

3.5.2 Command Reception, Validation and Execution

As noted earlier, each major Flight Software application or subsystem contains a C&C manager process which registers a RTX mailbox for receiving C&C frames from the CPM. Upon receiving such a frame, the manager process will decode the frame and validate its contents for correctness (as determined by such parameters as correct length, version and suitability for execution based on the current status of the module). The manager process will then either perform the necessary actions immediately, or schedule further control processes in order to perform the desired action sequence.

Some of the more complex modules (such as the ADCS and IBUS subsystem managers) make use of a more sophisticated mechanism that employs status vectors, masks and validation vectors to determine whether a module is able to execute the received C&C frame [Ste00, p.73-74]. The status vector, representing the module's complete state, is first mapped by the supplied mask, effectively leaving only those status values that are of interest to the frame's contents. The resulting vector is then compared to the validation vector to determine whether the module is in a state that would allow execution. Most of the subsystem managers are implemented as discrete state machines that relate well to such a method of validation.

For a more complete discussion on the workings of the subsystem manager components, refer to [Ste00, p.70-78].

3.6 Conclusion

This chapter described and analysed the various components that comprises the Flight Software Architecture. In the next chapter, the an equivalent study will be performed for the groundsystem.

Though operational experience with the software has been highly favourable, some criticism must be leveled against key modules, particularly PACSAT. In addition, a re-evaluation of the choice for the link-layer protocol is warranted, given the advances made in the field over the past two decades of AX.25's existence. A critique of the Communications Software thus follows in Chapter 5.

Chapter 4

Groundsystem Software Architecture

4.1 Introduction

Section 2.5 introduced the physical topology of the SUNSAT groundstation. This chapter will analyse and describe the architecture of the groundsystem software. In the final section some operational limitations of the groundsystem will be discussed, as experienced by the groundstation operators, and suitable extensions will be offered to address these.

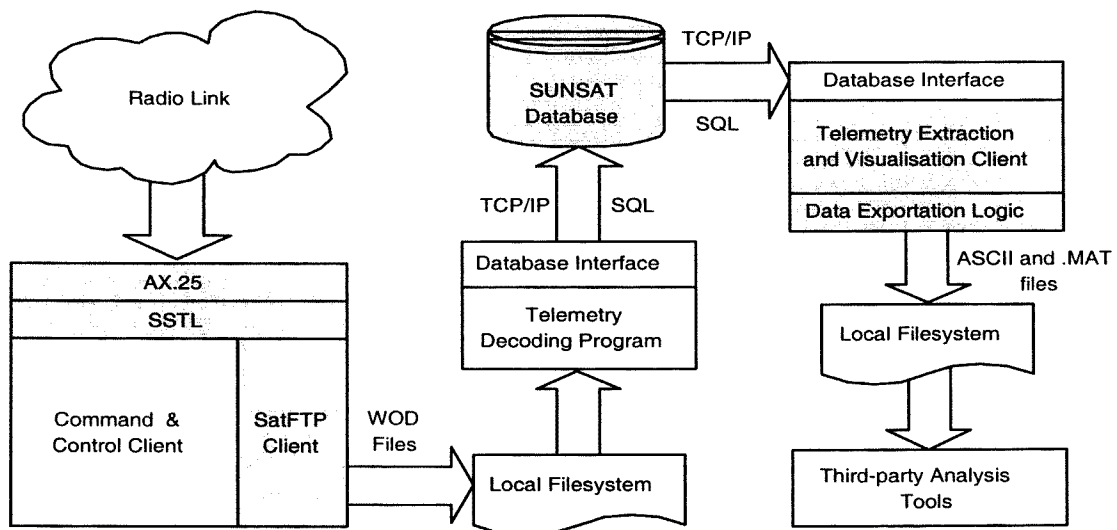


Figure 4.1: *Groundsystem Dataflow*

The groundsystem software modules are in many respects the functional mirror-images of their Flight Software counterparts – with the general distinction of implementing only the

client-side of a client-server model. The main exception to the rule is the AX.25 link-layer, which operates in so-called balanced mode with no distinction being made between the parties involved. Similarly, the SSTL protocol is highly symmetrical with the only exception being the authentication mechanism which differentiates between the client and server role.

Figure 4.1, which should be considered along with figure 2.4, analyses the logical dataflow in the groundsystem when whole orbit data (WOD) files flow through the data processing procedure:

- Downloading via SatFTP.
- Storage on a local file system.
- Decoding and possibly repair.
- Storage in the SUNSAT Database.
- Extraction, calibration and visualisation by the Telemetry Client.
- Exportation for analysis by third-party tools.

The figure also identifies the major components involved, with those developed as part of this thesis being clearly indicated.

The groundsystem modules that will be covered in this chapter are:

- Groundstation Command & Control Client
- SatFTP Client
- Telemetry Decoding Program
- SUNSAT Database
- Telemetry Extraction and Visualisation Client

The database interfaces, linked into both the decoding program and the telemetry client, are third-party libraries. They were obtained as part of the deployed database management system (DBMS) – also a third-party product.

4.1.1 Whole Orbit Data (WOD) Files

Though many of the specialised subsystem and payload modules produce their own files on the SFS file system, the WOD file storage facility, provided by the Telemetry Manager, is used as a central mechanism for the storage of both hardware and software orbital data that are intended for download and analysis.

A typical WOD file consists of a large number of sequential entries of the following types:

Telemetry Data Frame The TLM subsystem collects hardware telemetry data from a total of 255 channels on the satellite at a frequency of 1 frame approximately every 2.5 seconds. The Telemetry Manager allows for configuring a 256 bit mask that determines which channels' sampled data are actually stored. This mask can be updated from the groundstation Command & Control client.

Full Telemetry Data Frame In order to preserve some data from those channels that are not being sampled, a full telemetry frame, consisting of data sampled from all the channels, is stored at a configurable interval.

Engineering Frames Engineering frames, that allow the decoding program to determine which channels were stored, are placed in the WOD files every time the configuration mask is changed. An engineering frame is also saved in the WOD file whenever a configurable number of telemetry frames have been stored.

Software Telemetry Data Though some of the subsystem and payload modules produce data files of their own, many produce too little data to justify their individual management, especially file transfer. The Telemetry Manager provides an interface whereby software modules can store their telemetry in the WOD file.

Name	Size	Purpose
SyncBytes	16 bits	A number unique to a specific Flight Software update version.
TimeStamp	32 bits	The epoch (in seconds) when the frame was created.
FrameCounter	32 bits	A uniquely assigned frame number.
PacketSize	16 bits	The size of the stored data frame.
ID	8 bits	A value identifying the frame type.

Table 4.1: Header Format for a WOD File Entry

Table 4.1 describes the format of the header that precedes each WOD file entry.

Refer to [Ste00, p.89-93] for a lengthier discussion on WOD file creation, as part of a discourse on the Housekeeping function of the SUNSAT Flight Software.

4.2 Groundstation Command & Control Client

The Command & Control (C&C) Client, which was almost in its entirety developed outside the scope of this thesis, is implemented as a single executable program. It provides a switchable, multi-screen user-interface to a number of command and configuration facilities. These are:

AX.25 Status Display and Command Console The status of the AX.25 connection to the satellite is displayed, and the connection activity is decoded and displayed for easier analysis.

Radio Command Console and Status The status of the various radios employed in the groundstation is displayed here. Any of a set of configuration commands call also be send to individual radios.

SatFTP Client Console This facility will be discussed in the next section.

Multiplexer Configuration The multiplexer (see figure 2.4), that was specifically designed for the SUNSAT groundstation, supports the simultaneous operation of up to eight radios and eight modems. This facility allows for configuring the routing of both analog and digital signals between individual modems and radios, thus allowing any modem to be connected to any radio.

Log Server Client Broadcasted log messages from the satellite are decoded and displayed here.

SSTP Console Commands which affect the operation of the SUNSAT Simple Time protocol (SSTP) [Ste00, p.163-170] can be entered here.

Client Configuration The C&C Console supports up to four SCC cards whose individual configuration settings (such as baud rate and time delay settings) can be adjusted here.

Telecommand and Command & Control Console Commands to the Telecommand subsystem on the satellite can be sent directly from here. In addition, any of a predefined collection of 524 Command & Control instruction sets can be sent via AX.25 to the Command Processing Module (see section 3.5).

Though the automation of the SUNSAT groundsystem has long since been contemplated [Car97], this goal has not yet not been attained. Specifically, the C&C Client is a monolithic implementation of a set of functions which should rather be separated and extended into individual services. Such a configuration could be more easily automated. This subject will receive more attention in section 6.5.

It was judged that it would be too time consuming to port all the implementation (SSTL, AX.25 and the SCC driver) and definition modules (nearly all the Flight Software components) that the C&C Client depends on to a different programming language. The C&C Client was therefor implemented in Modula-2 as well. Unfortunately, the available Modula-2 compiler only supports the DOS executable format. A modern GUI environment was therefor not available. A limited, text-based interface had to be developed. In addition, the DOS memory model imposed severe memory restrictions. A more flexible and modern programming language will be considered in section 6.2.1.

4.3 SatFTP Client

The SatFTP client, as part of the C&C Client package, provides an operator's console for the dispatch of commands to the satellite-based SatFTP server. Commands for the execution of all the SatFTP operations described in section 3.4.3 can be issued.

The SatFTP client also displays status information, detailed error responses, and progress indicators for file transfers. Files downloaded via the client are stored in a directory on the local file system. Any files on the local file system can also be uploaded to directories on the SFS file system. Because of the limited network connectivity of the DOS-based C&C Client, access to network-based resources is restricted.

4.4 Telemetry Decoding Program

The Telemetry Decoding Program is used to decode the WOD files downloaded via the SatFTP client. The decoding program reads sequentially through a WOD file (see section 4.1.1), extracting each entry and then, in turn, extracting individual elements from the stored telemetry frame. Masks stored in the engineering frames are used to identify the sampled channels for hardware telemetry frames. The decoding program can optionally connect to the SUNSAT Database via a network-capable interface library. Entire frames or individual frame elements are then stored in the appropriate database tables.

By using the SyncBytes and FrameCounter fields from the WOD file entry record (see table 4.1), the decoding program is able to recover frames from damaged WOD files. It should be noted that the decoding program was developed outside the scope of this thesis.

4.5 SUNSAT Database

The SUNSAT database forms the center of the Data Processing activities of the Mission Operations System (MOS) (see section 2.2).

Organised according to a relational database schema and managed by a dedicated Database Management System (DBMS), the SUNSAT Database provides for the archiving and retrieval of all mission data. The DBMS employed fully supports the standardised Structured Query Language (SQL) [RC93, p.79-123]. All connections to the DBMS are managed by the SQL-capable database interface libraries, which translates the textual SQL queries into the DBMS's native application protocol.

SQL is a non-procedural, declarative language. As such it specifies *what* actions should be performed, but not *how* they are to be carried out. By describing data management operations in SQL, such activities can be better optimised by the DBMS. In addition, data processing systems are kept DBMS independent.

The relational database model [RC93, p.29-33], as first proposed by E.F. Codd in 1970 [Cod70], is an abstract approach to data modeling. The central feature is the concept of tables (also known as an entities): matrices of columns and rows, representing data attributes and sets of database records consisting of such attributes. Tables are related to

each other by sharing a common entity characteristic (a table attribute of the same type, whose records share matching values for that attribute).

The most important advantage of the relational database model is that a single, integrated data depository is made available via a standardised interface (SQL), with all the physical storage particulars being hidden behind a more convenient, logical abstraction. The relational database management system (RDBMS) is a complex piece of software which performs such additional tasks as query optimisation, table locking for simultaneous access, data indexing and meta-data management.

Additional concepts of the relational database model include:

Attributes Each table consists of a number of attributes, often referred to as fields, that describe the component parts of each record from that table. Each attribute has a specific type (such string, character or number) and other meta-data information such as its exact size, and whether it is a key, or forms part of a composite key.

Null Value A null value is a special value that indicates that, for a particular record attribute, no value was supplied.

Keys Keys (or composite keys in the cases when attributes are combined), are table attributes whose values are used to specify selective sets of table record entries for retrieval.

Superkey A Superkey is a table attribute, or set of attributes, for which the table records all contain unique values.

Primary Key A primary key is a superkey that has been selected to uniquely represent each record of a table. It cannot contain null values.

Foreign Keys An attribute (or combination of attributes) of one table, whose values must either match those of the primary keys of a related table, or be null.

Index Indexes are defined on keys. It is the task of the RDBMS to maintain data structures for the fast retrieval of records based on queries that depend on such keys. Primary keys are indexed by default.

Entity Integrity The database condition which exists when the primary keys of all tables are unique for all their records.

Referential Integrity The database condition when, for all the foreign keys for all records from all the tables of the database, there exists at least one corresponding record in the related table whose corresponding attribute shares the same value.

Database storage is not always the best solution for the archiving of mission data. When particularly large sets of binary data are to be kept, it is often more efficient to store such data in a dedicated mass-storage facility. The central database is then only used as a catalogue of the data sets which are stored externally.

The European Space Agency (ESA) SOHO (Solar and Heliospheric Observatory) [Sch96b], for instance, receives approximately 0.7GB of raw data per day. That translates into about 2GB of processed data. Over 2 years, this has accumulated to over 1.5 terabytes of stored data. In response, an automated cartridge library was installed, with a 100GB Network File System (NFS) acting as a temporary storage facility.

The storage requirements for the SUNSAT mission data are far more modest. This is primarily due to the limited time periods that the satellite is in contact with the groundstation, but also because of the limited bandwidth of the digital link (9600 baud). As of this writing, the SUNSAT database contains approximately 62.5 million data elements, translating into almost 100 megabyte of uncalibrated, but decoded data. This data was drawn from the approximately 265 megabytes of WOD file data (4923 files) that has been downloaded in the period starting with the launch date of 23 February 1999, and ending on the 30th of November 2000. This is in addition to the 250 megabytes of image data (1506 files) that has also been downloaded. In total, all the groundstation activities (including downloads) have produced roughly 1.25 gigabytes of data over the same period.

4.5.1 The SUNSAT Database Schema

The SUNSAT Database schema is a model, expressed in the terms of the relational database model, that was designed to satisfy the requirements of the MOS data depository function for mission data storage and retrieval.

Figure 4.2 is a diagram that illustrates the entities, attributes and relations of the SUNSAT Database schema, usually referred to as an entity relationship (E-R) model [RC93, p.148-168].

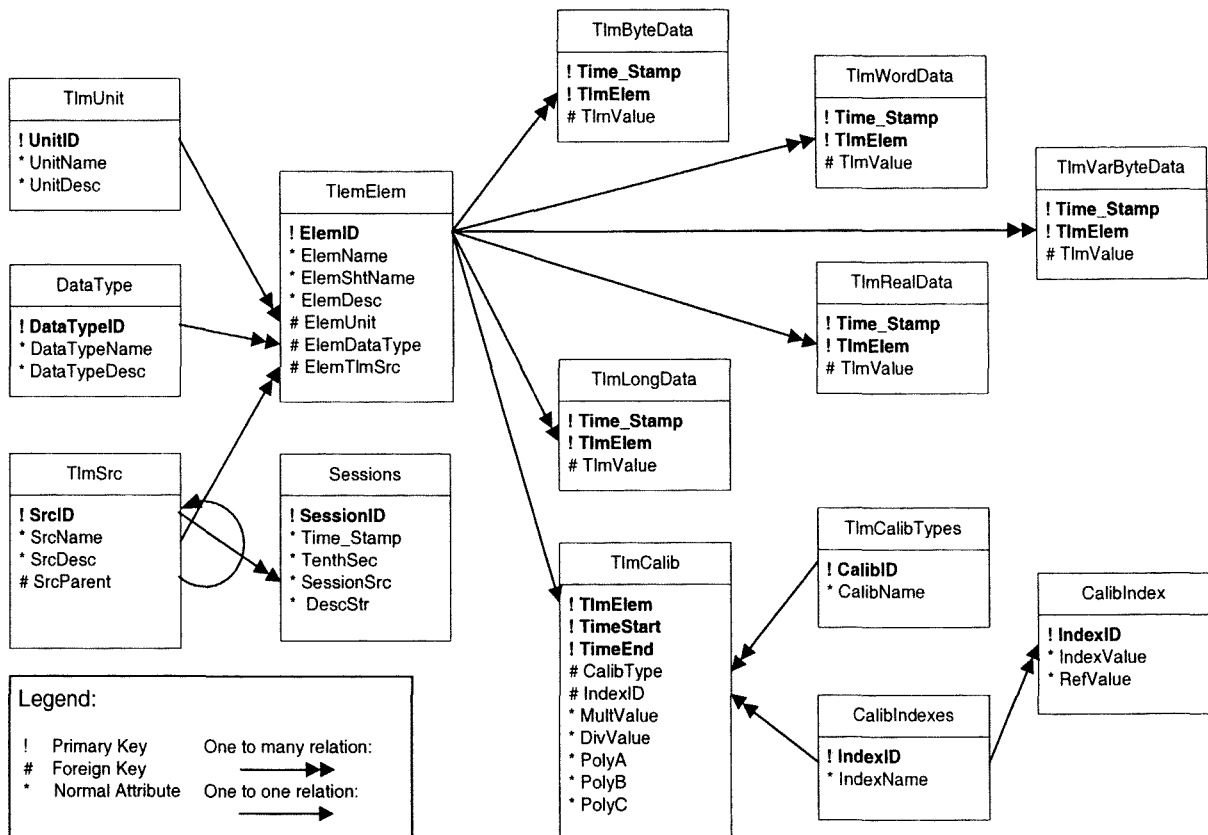


Figure 4.2: *The SUNSAT Database Schema as an E-R Diagram*

The model consists of the following entities:

Telemetry Source (TImSrc) This entity contains meta-data used for the hierarchical organisation of the telemetry data. The main category names are those of the various SUNSAT hardware subsystems and the software modules.

Telemetry Element Entries (TImElem) This is a index describing the primary attributes of the individually stored data elements. There are, as of this writing, over 360 defined elements.

Unit Measure (TImUnit) The units which the telemetry measures (such as centigrade, ampere or volts) is specified here.

DataType The datatype of the stored element. This also determines in which of the type-specific tables the actual stored data element can be found. Currently only

byte, variable byte-array, word (16 bit), long (32 bits) and real (32 bit) datatypes are supported.

Sessions Details of every action performed from the C&C console during contact with the satellite are stored here.

Calibration Meta (TImCalib) Calibration meta-data for every telemetry element, such as polynomial coefficients.

Index of calibration procedures (TImCalibTypes) The available calibration procedures are defined here.

Calibration Lookup Definitions (CalibIndex) Some calibration procedures require sets of indexed values. These are defined here.

Indexed Lookup Tables (CalibIndexes) The actual indexed values, required by the calibration procedures defined in CalibIndex, are stored in this table.

This database schema has served its initial purpose of facilitating the archiving of the SUNSAT mission. Several extensions, addressing specific operational limitations, will be proposed in section 4.7. An updated and more detailed specification of the SUNSAT database is provided in Appendix B. In Chapter 6 the need for a more integrated approach to data descriptions, based on the principles of a Unified Information Model (UIM), will also be considered.

4.6 Telemetry Extraction and Visualisation Client

The Telemetry Client performs the following Data Handling functions:

- The selective extraction of mission data from the database.
- Calibration of mission data by means of configurable calibration procedures and meta-data.
- Visualisation of calibrated data in graph form.
- The exportation of calibrated mission data sets to files for analysis by third-party tools. Only ASCII and Matlab (.MAT) files are currently supported.

Groundsystem Software Architecture 4.6 Telemetry Extraction and Visualisation Client

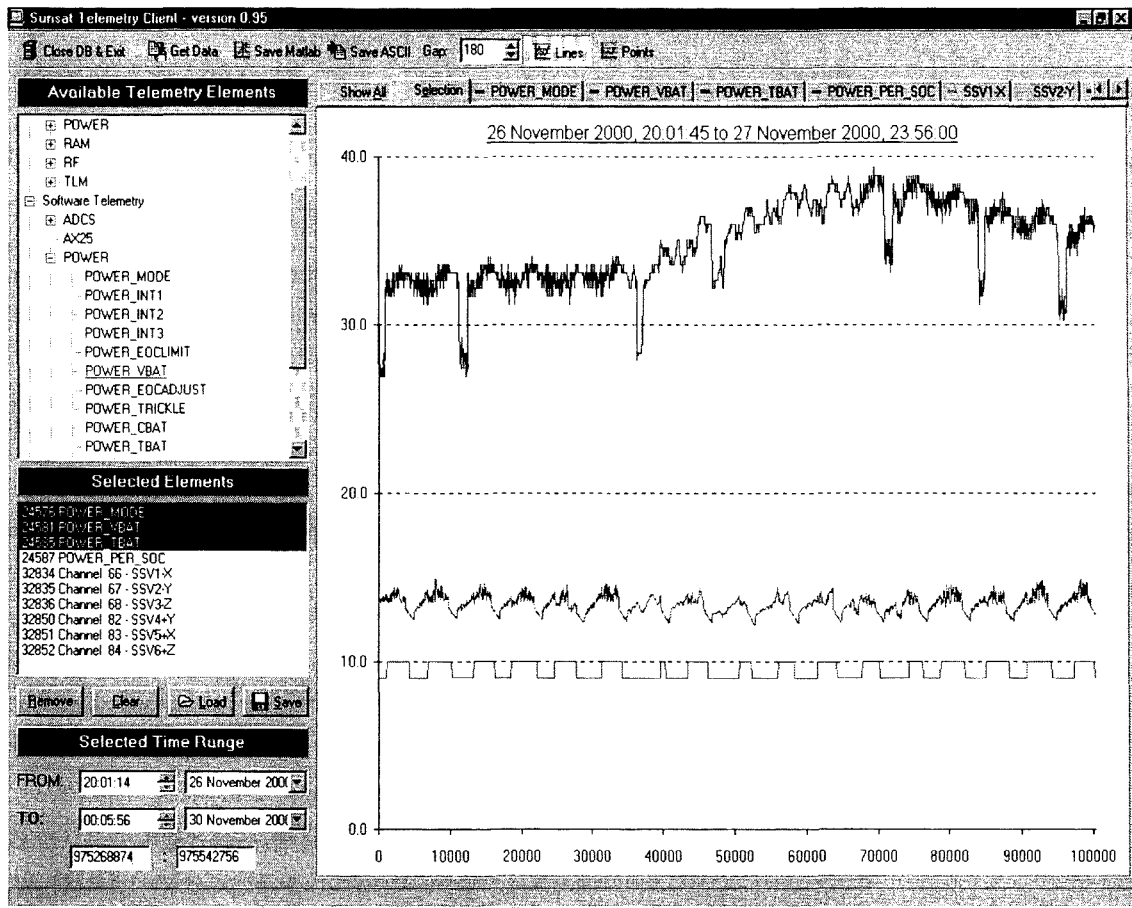


Figure 4.3: A screenshot of the Telemetry Client

The Telemetry Client is implemented as a user-friendly GUI interface (figure 4.3), following accepted visual interface design concepts. Sets of data elements can be individually selected from a tree-based hierarchical display of all archived data elements. Such sets can be stored for later re-use. Data selection is also based on time periods with one second accuracy. The visualisation component allows for the simultaneous, selective or individual display of data elements, along with other configurable options.

The SUNSAT groundsystem's Data Handling procedures share many design principles with other micro-satellite missions. For comparison, [MHHS99] reviews case studies from three recent NASA micro-satellite missions: MSX, NEAR and TIMED. These missions all employ DBMS software as their data repositories.

Though both the database schema and the Telemetry Client have received satisfactory operational evaluations, some of their design elements did impose a number of restrictions and operational difficulties. These will be discussed, along with suitable solutions, in the next section.

4.7 Operational Experience and Recommendations

Informal reviews requested of the groundstation operators, in addition to other forms of feedback on the operational experience with the SUNSAT database system and its procedures, provided several requests for enhancement. These enhancements will be briefly discussed with suitable extensions being made to the database schema, as documented in Appendix B.

Complex Data Description Many data sources, especially the Flight Software, store their telemetry as complex data structures such as records. While these can often be parsed into more atomic elements, description support for complex data elements is greatly preferred since such valuable meta-data, such as the relation between individual data elements, will be preserved.

This problem can only be satisfactorily addressed by the adoption of a suitable data description language, as will be discussed at length in section 6.4.

Fast Data Preview When searching for particular events or trends in the telemetry, data extractions of long periods must be made. Given the amount of data stored, this is a tedious process.

Each data element can be stored with an associated sequence number, repeating from zero to some arbitrary number. By only extracting elements of one or more sequences, extraction can be greatly sped up. This enhancement would also allow incremental extractions during normal data accesses, thus improving the response time of the extraction Client. A *SeqNum* attribute has been added to all the telemetry data storage tables.

Remote Data Extraction A requirement for the remote access to the SUNSAT database has been identified.

Groundsystem Software Architecture 4.7 Operational Experience and Recommendations

While this need is easily addressed in the short term by providing a web-based query interface, in the longer term, some more suitable, generic remote access interface needs to be provided. This issue will be addressed by some of the technologies explored in Chapter 6, particularly in section 6.2.

Multiple Data Sources Some of the SUNSAT telemetry, especially broadcasted telemetry, is captured and forwarded by radio amateurs. More groundstations will also come into operation in the future.

A *Source* attribute has been added to the data storage tables which is a relation to the new *DataSource* entity.

Caching of Extracted Data The Visualisation Client (see previous section) does not cache the results of prior searches, leading to unnecessary database accesses. The caching of extracted data will also speed up data zoom operations.

This issue should be addressed by a new data-model in the next revision of the Client.

Finer TimeStamp Granularity Some of the telemetry generation events, both on the satellite and the groundsystem, produce data elements at a sub-second rate. Allowances must therefore be made for finer timestamp granularity.

The addition of the *SubSec* attribute to the data storage entities allows for a hundredth of a second accuracy.

Chapter 5

Communications Software Critique with Recommendations

5.1 Introduction

During more than a year and a half of operation, the SUNSAT mission has been well served by the Flight Software covered by this thesis and elsewhere (eg. [Ste00]). It is clear that the repeated cycles of redesign and re-implementation, though costly in time and labour, played a major roll, leading to the accumulation of the necessary experience and knowledge which ultimately explains the consistently high quality of the software (as indicated by high maintainability and measured by mean time between failures).

Specifically the communications software covered by this thesis has met its operational goals. Since the first contacts were established with SUNSAT during its commissioning phase after launch in February 1999, positive feedback and evaluations have been received (refer to [GM99, p.14]). Consistently high data transfer rates and reliable connections have been characteristic, with over 710KB of data at 9600 baud being downloaded during a single groundstation contact [GM99, p.13].

However, regardless of the quality of any implementation, the suitability of the underlying specifications and design decisions always remain open to criticism. The performance and suitability of the SUNSAT communications software is particularly critical, since the overall efficient operation of the satellite mission depends upon it. The rest of the Flight Software is closely interdependent with the communications software (see section 2.3 and figure 3.1),

hence the special attention paid to the design and implementation of the communications protocol stack.

In this chapter, the specific technologies that were implemented as part of the SUNSAT communications software will therefore be reassessed according to the following criteria:

- Suitability for supporting future missions and their payloads.
- Interoperability with existing and emerging standards.
- Operational considerations such as efficiency and reliability in an embedded environment.

5.2 AX.25 Evaluation

The original requirement for the implementation of AX.25 existed *a priori*. AX.25 remains the de facto standard for amateur packet-radio today, and no true alternatives existed at the time. However, almost two decades have gone by since AX.25 was first proposed, and the state of the art in communications and embedded systems has advanced dramatically. It is clearly a suitable time to re-assess the situation and perhaps plot a new course.

The key consideration for the efficient operation of a packet-radio network is, of course, maximum exploitation of available transmission capacity. AX.25 is not considered to be optimal in this regard for a number of reasons (from [MTB95, p.5]):

- AX.25 allows for multiple repeater operation which is, strictly speaking, not the responsibility of the link-layer. Much greater flexibility and functionality may be achieved by performing the function of establishing virtual circuits at a higher (network) protocol layer.
- Similarly, source routing is not strictly the responsibility of the link-layer. It should also form part of the network layer.
- AX.25 broadcasts the amateur callsigns it uses for addressing within every frame. Since each callsign is 7 octets long, with an additional 1 octet SSID, this amounts to 16 wasted octets for every frame broadcasted.

[Kar94] follows a different approach to criticising AX.25. Firstly, AX.25 makes no allowance for the special problems commonly experienced with packet-radio use, namely partial collisions, noisy links and congestion. The basic argument is that link-layer protocols designed for wired networks (AX.25 still follows the principles of the X.25 standard) do not translate well to packet-radio. Also, AX.25 was designed for ease of implementation, with simplicity being a basic requirement. However, two decades have passed and the computational power that is now readily available, even in embedded environments, has increased by an order of magnitude, allowing the implementation of more complex algorithms, such as advanced error-correction techniques as opposed to simple CRC-based error-detection.

In the evaluation of possible alternatives, it should be noted that satellite channels have several characteristics that differ from terrestrial ones in crucial respects [AGS99, p.4]. These differences are of particular importance when adopting existing wire-packet protocols to the satellite packet-radio environment. They are as follows:

Long Feedback Loop The propagation delay to a LEO orbit ranges from several milliseconds when communicating with a satellite directly overhead, to as much as 80ms when the satellite is on the horizon [AGS99, p.2].

Large Delay Bandwidth Product The Delay Bandwidth Product (DBP) defines the amount of data a protocol should have “in flight” (data that has been transmitted, but not yet acknowledged) at any one time to fully utilise the available channel capacity.

Transmission Errors Satellite channels exhibit a higher bit-error rate (BER) than typical terrestrial networks. Wire-packet protocols (particularly TCP) would assume that such packet-loss signals network congestion, and respond inappropriately by reducing the size of its sliding window.

Variable Round Trip Times Particularly in LEO constellations, the propagation delay to and from the satellite varies greatly over time.

Intermittent Connectivity In non-geostationary orbit configurations, TCP connections must be handed over from one satellite to another, or from one groundstation to another, from time to time. These handovers will cause packet-loss if not handled correctly.

5.2.1 First Alternative – AX.25 Version 2.2

Though version 2.2 of the protocol does not address most of the points listed in the previous section, it does however address a wide range of inefficiencies and limitations associated with prior versions. A comparison between version 2.0 [Fox84] and version 2.2 [BNT97] of the specification yields the following changes and extensions:

- In response to criticism of AX.25's relation to the OSI layers (see figure 3.1) its descriptive name has been altered from "link-layer" to "link access" protocol.
- Repeater chaining has been phased out with backward compatibility being maintained with a limit of two repeaters.
- A selective rejection (SREJ) command and response have been added. The use of the SREJ mechanism results in the retransmission of only a single packet of information whenever one is lost due to corruption or frame collision. In the absence of repeated frame-loss, this addition should yield greater link efficiency.
- An exchange identification (XID) frame has been added that would cause a receiving station to identify itself and to provide characteristics to the sending station. The XID frame complies with ISO 8885.
- Further extension to the specification was made with the addition of command elements for layer-to-layer communications. This serves to formalise the interoperation of the link-layer with both higher and lower layers of the OSI.
- A total of ten new timers have been added to the existing three. These include an anti-hogging limit timer which prevents a station from monopolising the channel. A priority window timer also reserves time slots for priority frames (acknowledgements and repeater frames) which would lead to greater channel utilisation.
- Finally, version 2.2 includes a formal definition of the protocol in the form of System Description Language (SDL) diagrams. The SDL formalisation has precedence over the textual description and should ease the correct and efficient implementation of the protocol.

Though a complete re-implementation of version 2.2, closely following the SDL definition, would be preferable, even the selective implementation of some of the new extensions (in

particular the SREJ command/response pair) is highly recommended, and should lead to direct and immediate improvements in efficiency. To illustrate this improvement, consider figure 5.1. It depicts a scenario where the loss of a single frame initially causes multiple frame retransmissions. After the addition of the SREJ mechanism, only a only a single retransmission is necessary.

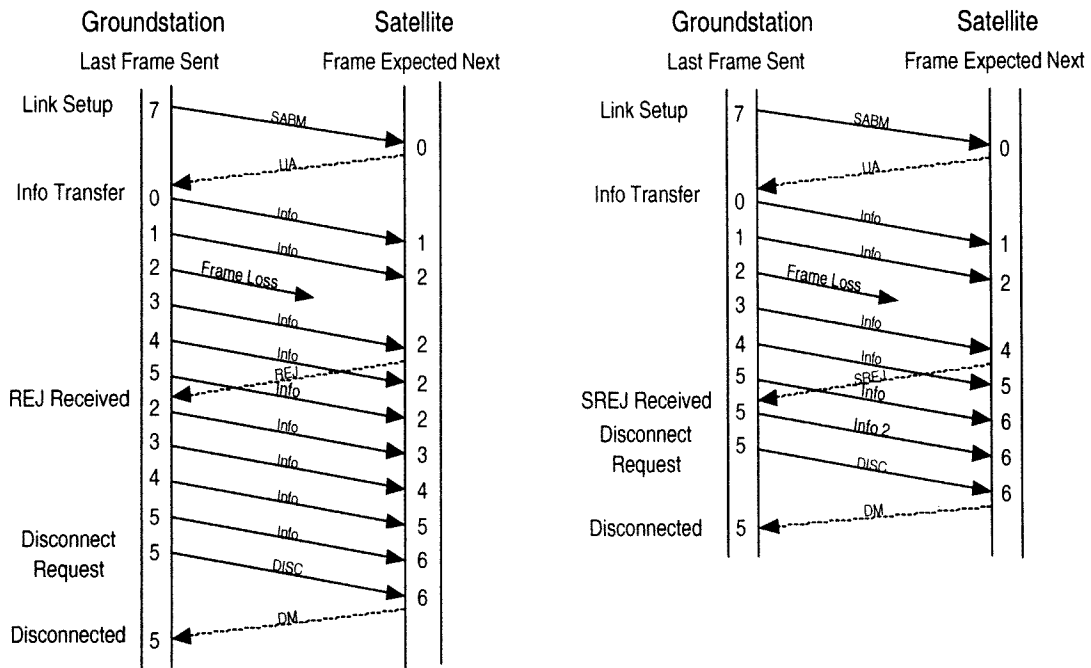


Figure 5.1: Efficiency improvement with the addition of the SREJ mechanism

Version 2.2 of the specification makes explicit provision for backward compatibility with version 2.0. It is also considered better suited for encapsulated high-layer protocol streams (such as TCP/IP) than earlier versions, where such issues were not addressed in any great detail.

Finally, the new SDL-based specification places AX.25 ahead of most international standards and will surely ease implementation and the verification of correctness.

5.2.2 Second Alternative – TCP/IP and Derivatives

The Transmission Control Protocol (TCP) and Internet Protocol (IP) [Ste94] pair is doubtless the most popular solution for packet-based networking in use, and has accumulated a

number of extensions in order to keep pace with their rapid adaptation in a number of diverse networking environments.

It has been noted [DMT96] that space communications and mobile and wireless communications environments have many similarities when viewed from the perspective of a transport protocol. Consequently, a wide body of existing techniques derived from the commercial exploitation of wireless networking are available to researchers, in addition to technology already developed by the space industry. The adaptation of TCP/IP to the space communications environment is a very active and rapidly growing field of study. The Internet Engineering Task Force (IETF) has formed a "TCP/IP for Satellite" Workgroup and two important Requests for Comments (RFCs) have recently been published [AGS99] [MA00].

Two basic approaches are followed when TCP/IP is adapted to the satellite environment: either existing TCP mechanisms are enhanced to deal with the unique requirements (see [AGS99]) or special extensions are added (see [DMT96]). Enhancements to the standard mechanisms include (see [AGS99, p.8-13]):

- sending larger packets for better channel utilisation;
- fast retransmit through selective resends;
- selective acknowledgments (SACKs) for faster recovery;
- larger TCP window sizes, to compensate for the longer Round-Trip Time (RTT); and
- alternative decision algorithms for recovery and congestion control.

[DMT96] follows the second approach by proposing the Space Communications Protocol Standard-Transport Protocol (SCPS-TP), a set of extensions (as opposed to mere enhancements) which are tailored to address all the identified problems. The exact details of these extensions are considered to be outside the scope of this thesis.

A final consideration when evaluating TCP/IP for the satellite environment is the inefficient bandwidth utilisation caused by the typically large header sizes. With the advent of IPv6, address size will increase from 4 to 16 bytes and a typical IP header will increase from 20 to 40 bytes. [DENP96] proposes a low-loss header compression scheme, specifically for wireless networks.

Performance studies that have evaluated the proposed enhancements to TCP/IP for the satellite environment [AHKO97] [Hay97] generally deliver quite favorable verdicts. The current set of suggested enhancements are considered to be adequate for achieving high throughput over satellite links. Research is, of course, still on-going. For the most recent results and suggested research, refer to RFC 2760 [MA00].

In terms of serving as an alternative to AX.25, TCP/IP is highly recommended. Even with the proposed enhancements, the protocol will still interoperate with all other existing implementations. Given the large set of current and emerging technologies based on TCP/IP, adoption is certain to bring numerous immediate benefits.

5.2.3 Third Alternative – DUAL

The Digital Unreliable Amateur Link (DUAL) Format as proposed in [MTB95] offers an entirely different approach to overcoming the limitations of AX.25. DUAL is offered as a partial, and therefore simpler, replacement of AX.25. It seeks to achieve the following:

- Provide for the transmission of packets from a range of high-layer protocol types, using link addresses suitable to these protocols, rather than amateur callsigns.
- Provide a mechanism to broadcast the association between the link addresses and the callsign of the operators only when necessary.
- Limit the overhead in the link-layer, primarily by drastically reducing the header size.
- Separate the logical link control, medium access control and physical layers in a modular way so that DUAL can be used independently of these.

The original design goal for DUAL was to create a link-layer that was better suited to the transmission of TCP/IP datagrams across amateur packet-radio connections. The DUAL specification thus adds several functions, such as IP header compression, to render it more efficient in this regard.

DUAL achieves its goals by aiming for even greater simplicity than AX.25. It is basically a strict broadcast protocol, with flow-control and other functionality being left to the higher protocol layers that it is intended to facilitate. Though DUAL has not received much attention, it does however contain a number of viable design elements that warrant further investigation.

5.2.4 Fourth Alternative – Advanced Solutions

The fourth alternative to the existing AX.25 solution is to design a new protocol specifically for the packet-radio environment. Such a protocol should divorce itself from all the concepts inherited from the wired-protocol environment, and aim at achieving efficient operation in both the traditional Amateur Radio and LEO space communications environments.

One specific feature proposed for new link-layer radio protocols [Kar94] is the use of so-called forward error correction (FEC). FEC involves the transmission of redundant information which allows the correction of some errors resulting from interference without retransmission.

A rule of thumb when employing the traditional wired-protocol sliding window and acknowledgement scheme (automatic request repeat; ARQ), is that it only remains efficient while packet-loss stays below 1%. The conditions experienced on packet-radio links are often far worse. Using smaller packets would help, but at the cost of adding considerable overhead. Physical alternatives (increasing power and/or antenna gain) are neither practical nor economical. This leaves mechanisms such as FEC as the only remaining design options.

Arguments against using FEC is that it would add a constant overhead, whether needed or not. FEC's are also computationally very expensive to decode. [Kar94] evaluates various types of FEC, provides a solution to the problem of synchronisation and proposes a suitable frame format. A number of outstanding questions are also identified.

5.3 PACSAT Evaluation

Upon evaluation, the PBP broadcast protocol does not strike one as particularly well-designed. It lacks both clarity and completeness in its specification, and makes few allowances for future extension and third-party adaptation. Though simplicity was certainly one of the design goals, several points of criticism must be raised – despite the three sets of revisions already issued to address operational difficulties. The FTL0 protocol is better specified and its implementation is greatly eased by the use of a state-machine description in its specification. Still, it is considered flawed:

Incomplete Specification The PBP specification as published is extensive, yet fails to be complete in specifying all the mechanisms that are required for its effective operation.

Several rather informal revisions and extensions to the protocol have subsequently been released as electronic text. These are in fact crucial for the correct interpretation of the original publication that has never been updated.

Outdated Design Requirements Both protocols were originally designed for implementation on an 8Mhz 16-bit, 80C186 CPU. Memory and CPU speed restrictions necessitated rather simple protocols with a bare minimum of functionality. LEO satellites currently being operated or designed contain far more powerful embedded processors which certainly warrants the design and use of more sophisticated protocols.

Dependency on AX.25 The protocols, as specified, are rendered completely dependent on the AX.25 link-layer protocol by their design assumptions. No provision is made for implementation in other environments. This is a contravention of modern protocol design principles.

Not Extensible The PACSAT File Header, though it provides for optional headers, is not extensible. This limits its effective use.

Redundant Error-Detection The PBP protocol adds additional error detection data (a 16-bit XModem CRC) to its frames, duplicating both the function of the CRC data already contained in the PFH headers, and the error-detection provided by AX.25. This feature was introduced to address possible corruption when data is traveling between the TNC and the personal computer in a typical amateur radio groundstation. Such functionality is considered redundant and unnecessary given the likelihood of such corruption. The CRC calculation is computationally expensive and operationally cumbersome. This unnecessarily complicates the system, especially in light of the fact that separate CRC values must also be calculated for the PFH fields and the actual contents of a file.

Duplicate Functionality The search capability of FTL0 duplicates the directory broadcasting performed by PBP. Due to the complexity of the FTL0 search specification, this feature has seldom been fully implemented. Also, both PBP and FTL0 provide for file downloading.

The PACSAT suite also suffers from the following limitations:

Ineffective File Management PBP only allows for the deletion of files upon expiration of their indicated lifetime.

Limited File Operations Existing files cannot be updated nor appended to.

No Security Access to the protocols cannot be restricted in any way, leaving the system open to abuse.

Given the above criticism, it is strongly recommended that new research into the design of more capable and appropriate store-and-forward protocols be conducted.

5.4 Memory Management

As has been noted in [MvdM99] and [Ste00, p.129], the communications software currently suffers from a serious design limitation where memory allocation is concerned. Due to the use of RTX mailboxes as the inter-process communications mechanism for the communications protocol stack, data copying operations are unavoidably performed each time a data frame is passed from one module to another. Also, some modules maintain their own internal frame buffers. AX.25, for example, employs a linked list of buffers as part of its flow-control mechanism. These buffers add additional memory allocation and copying overhead. This state of affairs affects both the reliability and efficiency of the communications software because of the sub-optimal use of resources and introduction of the potential for deadlock.

As an example, a data frame destined for SatFTP, received by AX.25 in its mailbox from the SCC driver, will be decoded by AX.25 and its contents placed into SSSL's receiving mailbox. Each time RTX performs a copying operation, SSSL will perform any necessary operations (such as decryption), and place the processed data frame into SatFTP's mailbox. Another copying operation is therefor performed before SatFTP receives the frame, for a total of three copying operations. While transmission requests are performed by means of procedure calls between SatFTP and SSSL (for the sake of efficiency), and again between SSSL and AX.25, two copying operations are still required: once when AX.25 places the frame into its internal buffer, and again when it is copied by the SCC driver into its own internal buffer.

A further problem can be identified from the previous example: both AX.25 and the SCC driver maintain their own internal buffers. While this is clearly unavoidable, the pre-allocation of memory for both buffer pools is wasteful and could lead to scenarios where there is a buffer shortage in one module, while buffers are still available to other modules.

It has been suggested in [MvdM99] that a global frame pool mechanism be implemented with an interface suitable for passing only frame references, and not actual frame data, between modules. The frame pool can also pre-allocate a fixed number of frames in order to limit heap memory fragmentation and reduce memory allocation overhead. A proposal for such a global frame pool is provided in Appendix C.

Chapter 6

Towards a Unified Information Model

6.1 Introduction

In Chapter 3 the communications protocol stack was introduced. This discussion was continued in Chapter 4 where the complementary ground-system architecture was introduced. The emphasis then shifted towards addressing the problem of effectively organising, archiving and distributing the telemetry data gathered on the satellite, after its transfer to the groundsystem. This chapter will address the limitations of the current groundsystem, as identified by operational experience and subsequently pointed out in the previous chapters (in particular Chapter 4).

It should be noted beforehand that, with the prospect of simultaneous satellite missions (perhaps even constellations of satellites), multiple groundstations and multiple-party mission co-operation, the efficient and effective operation of the groundsystem will become all the more important in the future.

In section 2.5 the physical organisation of the SUNSAT groundstation was introduced, and in Chapter 3, the corresponding groundsystem software architecture was discussed. In this chapter a more integrated approach to the analysis and design of a groundsystem will be followed. A so-called Unified Information Model (UIM) will be proposed, based on a systems analysis of the groundstation requirements as taught by the the SUNSAT mission experience, and analyses gathered from studies published by comparable projects, elsewhere in the world.

The problem of efficiently handling the data produced as mission results, in addition to safe

and effective mission control, will, as always, receive our primary attention. This study and proposal is not intended to be final nor complete, but rather to serve as a first conceptual and technological survey of the field. The intention is to make strong suggestions and recommendations for a future development path, as well as to identify areas of future research.

6.2 Background Study

The following areas were identified for study, based on experience dealing with the existing SUNSAT groundsystem's limitations (see section 4.7):

Future Development Language The choice of implementation language determines the available of technologies and influences cost and efficiency.

Data Description Language The need for a data description language (DDL), capable of describing complex data structures, was identified in section 4.7.

Groundsystem Architecture The centralised or distributed nature of the groundsystem is an important design distinction, especially when future extensibility and scalability is considered.

6.2.1 Future Development Language

The Java programming language, first described in [GM95], is rapidly being adopted into a wide variety of development environments. Our concern is whether Java is suitable for deployment as the primary execution platform for a future groundsystem. The use of Java for the development of Flight Software must also be considered.

[MCL98] describes the development and deployment of the Jswitch/Jsat system at NASA's Goddard Space Flight Center. This project was the result of a drive towards the use of so-called COTS (commercial off the shelf, see section 2.2.1) technologies. Jswitch provides the network infrastructure, including security measures such as encryption and access control, while Jsat implements the primary Data Handling functions of data extraction and exportation. The project was judged a success, with some of the listed advantages of employing Java being: Java's suitability for distributed computing, code portability, easier

software maintenance and ease of deployment. It is concluded that Java fits very well with inexpensive, small satellite rapid development.

[MFKC98] analyses the incorporation of Java into a legacy C++ programming language environment. This study, published by researchers from the Lockheed Martin Space Missions Systems Services division, allows for the comparison of these two object oriented languages in a flight operations center environment. The emphasis of the paper is on Java's performance and interoperability, as well as how Java's unique features can benefit and enhance the existing system. While Java's performance and imperfect portability does raise concern, these problems are not considered insurmountable, especially given the rapid advance of the Java platform. In conclusion, Java is considered suitable, especially for interface development, and it is found that Java provides a unique set of abilities that the existing C++ system can not provide.

[ASD99] evaluates Java's suitability within the constraints of execution as an embedded real-time system. Java's strengths and shortfalls for embedded and real-time applications are discussed in detail. Concerns include stability given the language's relative youth, poor performance as compared to the C language and its non-deterministic memory management profile because of the garbage collection mechanism employed. Java's implementation on top of a real-time kernel is also considered, with experiments on the VxWorks platform being compared to execution on Sun Microsystem's Solaris platform. The study concludes that the Java technology tested is mature enough for the environment considered, and that adequate solutions can be found for the problems encountered, leaving Java as a realistic and cost-effective option.

The Java platform is rapidly maturing with broad support for diverse technologies being made available, including those considered in the rest of this chapter. The experience gathered prototyping these Java technologies during the background study is in agreement with the published results and Java is recommended as the implementation language of choice for future micro-satellite groundsystems. Further experimentation is recommended before considering Java for the implementation of Flight Software.

6.2.2 Data Description Language (DDL)

The emergence of the Internet's HTTP hypertext transport protocol and HTML markup language as a universal and near ubiquitous data and information conduit serves as an illustration of the potential data description languages hold. Given the volume and complexity of the data accumulated during scientific missions, particularly space operations, numerous data description languages have been developed to assist with the organisation and distribution of mission results.

Most of the such DDLs developed in the past were custom-designed for a particular mission or project. [olSS97], however, compares some of the more generalised DDLs currently being employed in satellite mission operations. Some of the criteria used to judge the suitability of a DDL include syntax support for complex data structures, semantic support for high-level data modeling, and software support. From the results of the survey it is clear that a suitably generalised, platform independent DDL was not to be found at the time.

Developed by the World Wide Web Consortium in association with a wide representation from the media and information technology industries, the Extensible Markup Language (XML) [BPSMM00] promises to address the need for a universal markup language, allowing applications-specific data description languages to be derived from a common base. The prospects that XML holds for the space mission environment will be considered in section 6.4.

6.2.3 Groundsystem Architecture

In the early 1980s the consequence of the continuous decrease in the cost of computing power and the rapid increase in the applications of digital communications was already apparent at such institutions as Xerox PARC. Three distinct phases of computing were subsequently identified, namely:

- Host-Centric
- Desktop-Centric
- Network-Centric

Since we are already well into the network computing phase, the ramifications of this shift on specifically software engineering is already appreciated as being enormous. This section will

briefly examine how a network-centric design approach can assist in improving the efficiency and reliability of a groundsystem architecture.

Besides the many benefits that increased network awareness has brought to computing environments in general, the need for such developments in micro-satellite groundstations in particular are driven by the following factors:

1. Increased multi-party cooperation in mission development and executions. Such scenarios necessitate a distributed approach to mission management.
2. The commercialisation of payloads requires the creation of facilities for third-party access to mission data and results.
3. The need to reduce cost leads to pressure for greater standardisation and the integration of, or at least interoperability between, previously distinct systems.
4. The great cost benefits which can be brought on by the automation of processes, especially with regard to entirely eliminating operator involvement, requires connecting physically separated systems for data exchange.

In addition, the shift towards a network model will have the following general benefits:

1. The creation of a transparent and pre-existing link between systems for the expansion of network services and the addition of future systems.
2. Increased flexibility and more opportunities for redundancy which in turn leads to greater reliability and future scalability.
3. The spatial distribution of systems and services, possibly over great distances, which allows for the better utilisation of system resources.
4. The automation of system processes in particular allows for the possibility of automatic fault detection and recovery leading to extended system usage.

[Via96] measures the potential impact of emerging distributed middleware technologies on satellite groundsystems. Technologies such as the Common Object Request Broker Architecture (CORBA) and the Distributed Computing Environment (DCE) are considered and compared. A prototype implementation of a distributed control center was made and

positive results were obtained. Some of the advantages listed are: greater redundancy and therefore reliability can be achieved, and legacy systems can be easily integrated with newly developed systems.

[SWP96] discusses the suitability of World Wide Web (W3) technologies, such as Java applets, for the development of a satellite payload monitoring system. A general design concept, built around a central database, is evolved. It is concluded that W3 technologies and their associated tools offer significant cost savings, in addition to improving maintainability.

[Mon98] presents a generic event handling subsystem, based on CORBA, for use at satellite operational control centers. Its many advantages – platform independence, scalability and robustness – are emphasised, and it is recommended that the service be extended to an open, distributed framework for spatial mission and data processing.

The advantages of a distributed approach, over the current centralised model (see section 4.2), is evident and the further investigation of suitable distributed protocols is recommended.

6.3 Characterisation of a UIM

For the purposes of this analysis the terms of the phrase *Unified Information Model* will be defined as follows:

Unified The equal treatment of all elements of similar function or description. Denotes completeness and consistency.

Information Contextualised data realised by the addition of meta-data such as relation, type and origin. Implies a level of interpretation actualised by its representation.

Model A conceptual structure suggesting organisation according to role or function, and association by means of relations.

In more practical terms, a Unified Information Model for a satellite groundsystem is understood to be characterised by the interoperability of all its software and the interchangeability of all data. This can be achieved by the establishment of a communications

mechanism common to all groundsystem components, and a data description language suitable to all data.

These characteristics have been discussed, in one form or another, in the previous sections. The final two sections will explicitly deal with the two main requirements for the achievement of a UIM:

- A suitable data description language and associated protocols and mechanisms.
- A suitable Mission Control Architecture for the execution of the Mission Plan for the groundsystem.

6.4 The Extensible Markup Language (XML)

The Extensible Markup Language [BPSMM00] is in fact a subset of the existing Standard Generalised Markup Language (SGML, ISO 8879). Thus, all XML documents are also conforming SGML documents. XML is in fact no more than a specification for the delimiting of textual data and was as such developed under the auspices of the W3C for use on the Internet. The design goals included:

- Support for a wide variety of applications
- Compatibility with SGML
- Human-legibility and reasonable clarity
- Formal specification and conciseness
- Ease of creation and of use

An XML document is considered to be well-formed if it meets all the constraints set by the XML specification. In addition, a XML document is considered valid if it adheres to the further constraints imposed by its associated Data Type Definition (DTD). One of the characteristics of XML, that sets it apart from HTML, is that there are no pre-defined XML tags, making it highly suited for deriving document formats from.

Since its first publication in 1996, a wide body of protocols and document formats have been based on XML. Some of the technologies relevant to our discussion include:

Document Object Model (DOM) The DOM, partially specified in [BC00], is intended as a platform- and language-neutral programming interface that would allow programs to access and update the content of documents. It is applicable to HTML, Cascading Style Sheets (CSS) and XML documents.

Simple Object Access Protocol (SOAP) SOAP [DB00] is a lightweight protocol for the exchange of information in a decentralised, distributed environment. It is an XML-based protocol providing for the description of message contents, the encoding of application-defined data types and includes conventions for remote procedure calls and responses. SOAP can therefore be described as a RPC mechanism for XML-based data.

XML Schema The purpose of the XML Schema specifications [Fal00] is to replace the existing DTD XML document formulation with a more capable and flexible scheme. XML Schema therefore allows for the definition of data types, in addition to more complex data structure definitions. XML Schema provides a number of standard, built in, data types, data models and data structures.

From this short introduction it should be evident that XML holds the potential of fully addressing our needs for a data description language, in addition to serving as a stepping stone towards more advanced technologies such as the SOAP distributed protocol. The further investigation of XML, and the development of a suitable definition for the description of satellite telemetry, is therefore highly recommended.

6.5 Mission Control Architecture

Figure 6.1 illustrates the results of an analysis performed in order to describe a fully-featured micro-satellite groundsystem according to the principles of a UIM. The resulting Mission Control Architecture (MCA) presents a network component topology consisting of a set of distinct services organised by function. The MCA is in this respect an extension of the functions introduced in section 2.2. This analysis made extensive use of the concepts developed in a paper [TS99] published by researchers from the Jet Propulsion Laboratory, entitled *A Mission Operations Architecture for the 21st Century*.

[TS99] reached its conclusions by performing a detailed analysis of the mission requirement for a groundsystem, when all the functions of a Mission Operations System are analysed in terms of descriptions of network services.

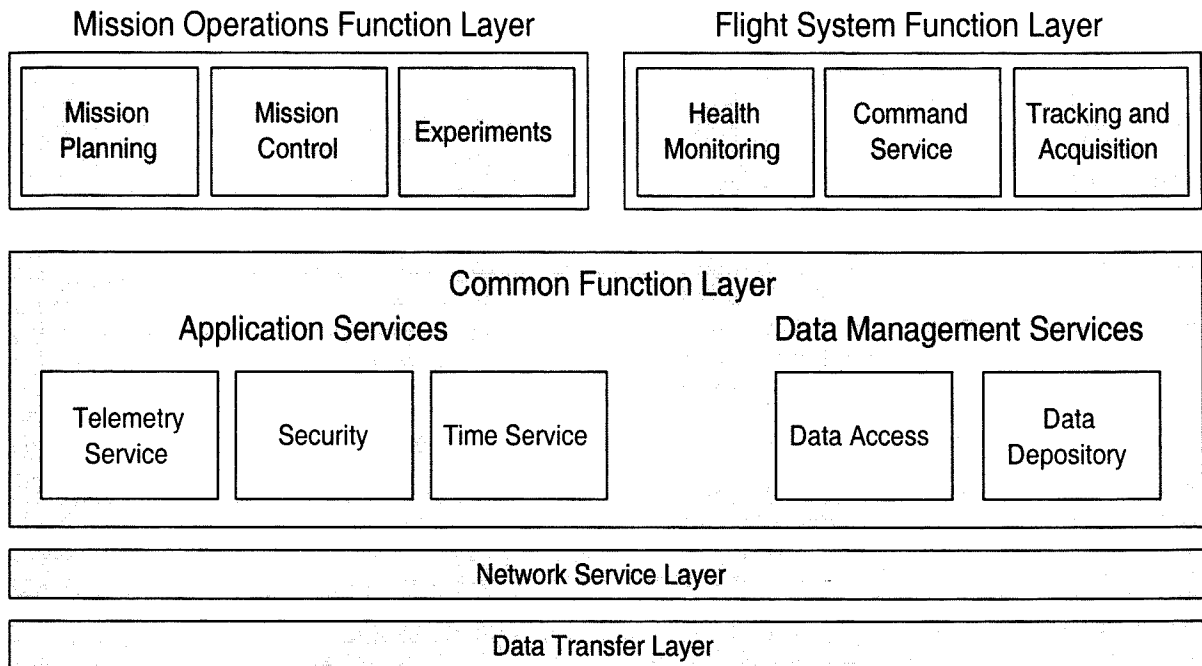


Figure 6.1: *Service System Architecture – A layered View*

The following are detailed descriptions of the identified services, when adopted to a micro-satellite groundsystem environment:

Mission Planning Since micro-satellites are only in contact with their groundstations for brief periods, detailed mission event sequences need to be created and uploaded to the satellite for execution at specific future time instances. This service encompasses the mission planning software and interfaces required for the creation of such event sequences.

Command Service During actual contact with the satellite, specific control sequences (via the telecommands system) may need to be sent to the satellite, in addition to the uploading of mission plans. This service provides such a command interface to the groundstation operators. This functionality is currently provided as part of the Command and Control Client (see section 4.2).

Data Depository Telemetry, error and command logs, science experiment results and other data need to be stored for future analysis. This system provides a comprehensive database for the storage and management of all mission data. The data should be described by a common data description language and be made accessible by means of common communication mechanisms.

Time Service For the accurate operation of the satellite and its scientific payload, the system clock on the satellite needs to be regularly and accurately synchronised with the groundstation. Interfacing to its satellite counterpart is an obvious requirement.

Tracking and Acquisition This service involves the control of the radio antennae and other ground communication equipment. It is also the responsibility of the Acquisition Service to notify the Mission Control service when a satellite enters the groundstation's footprint.

Mission Control The efficient and preferably automated operation of the groundstation requires a central command service which will initiate the various operation sequences of all the other services.

Telemetry Service After transferring all recorded telemetry and other data from the satellite, such data typically needs to be pre-processed and analysed before storage in the data depository.

Experiments Individual science payloads on the satellite may require specific maintenance or operational procedures.

Health Monitoring This would involve real-time analysis of critical satellite systems during groundstation contact, particularly the thermal and power systems of the satellite, with an operator alarm capability being provided for emergency intervention.

Data Access This service provides for the distributed and possibly remote access to depository data stored by the other services and by remote third party processes.

Data Transfer This service provides efficient and reliable communications and data transfer with the micro-satellite. It primarily consists of an implementation of a radio link-layer protocol, but would also offer file-transfer and broadcast capture capabilities.

Network Service The network service encompasses the common communications protocol and all associated network management functions such as address resolution and routing, but also such facilities as directory functions.

Security The security of the satellite, the integrity of the data depository and the possible commercial value of the scientific data, require that data access may be restricted on the level of individual services and parties to specific areas of the data depository and network services. The security service would therefore have to allow profile-based network identities and associated privileges.

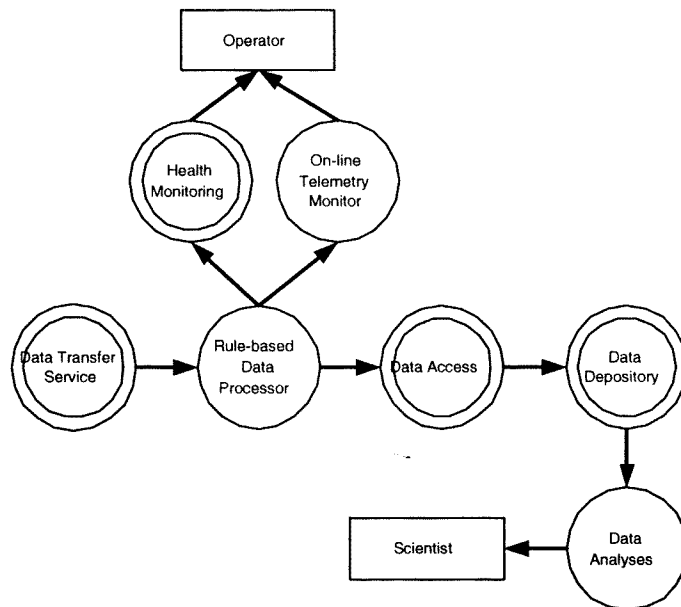


Figure 6.2: A Dataflow diagram centered on Telemetry and associated Services

Each network service, in turn, can be further analysed in greater detail in terms of its own specific functions and operation, particularly in relation to the other network services. Figure 6.2 provides such an illustration centered on the Telemetry Services and including the end-user.

Chapter 7

Results and Conclusion

7.1 Thesis Results

The thesis produced the following primary results, as judged by the initial problem statement in section 1.1:

Reliable and Efficient Communications A set of communications protocols was implemented and optimised to provide a reliable, efficient and secure means of communications between the groundsystem and the SUNSAT satellite. The modules implemented include AX.25, SSTL, SatFTP and PACSAT.

Orderly and effective Mission Control Support mechanisms such as the SatFTP transfer protocol, the SSTL protocol for security and the SFS file system were either implemented or enhanced to address the necessary Mission Control support requirements. In addition, many aspects of the groundsystem, such as the Visualisation Client, provide analysis and decision making support.

Effective and Efficient Data Handling Effective procedures and mechanisms for the storage of telemetry data on the satellite were provided, primarily by the SFS module. The efficacy of the satellite Data Handling function was demonstrated by the results achieved in successfully operating the satellite, particularly by making available the wide range of valuable engineering and scientific data collected.

The Data Transfer function is performed by the entire communications protocol stack. A suitable database schema for the orderly and efficient storage of decoded telemetry

data was developed and the necessary mechanisms were implemented. Finally, a Data Extraction and Visualisation Client was developed, which also provides data export functionality for telemetry analysis by third-party tools.

The following secondary results are also noteworthy:

Protocol Evaluations The communications protocol stack was evaluated for the suitability of some of its constituent modules and recommendations for improvement were made.

Memory Management An efficient scheme for the improvement of the memory management mechanisms in the Flight Software was proposed and specified.

Database Extensions Suitable extensions to the existing database schema were identified in order to address operational limitations.

Technology Review Various emerging technologies were reviewed in order to address current limitations and future requirements.

7.2 Future Work

In addition to the results achieved, the following areas of study have been identified for further investigation and research:

Next generation Link-Layer Protocol Introductions were provided to various alternatives to the existing AX.25 version 2.0 implementation. This area requires further investigation.

Groundsystem Distributed Protocol An initial survey of suitable distributed protocols for the development of a network-centric groundsystem was conducted. This area requires further research.

Data Description Language A recommendation was made for the adoption of XML as the data description language for the SUNSAT groundsystem. Further investigation of the various emerging technologies based on XML, such as the SOAP protocol, should be conducted.

Groundsystem Architecture A complete design specification for a next generation groundsystem, based on the principles of a Unified Information Model, should be created.

7.3 Conclusion

As was noted in the introduction, and further stipulated in the previous chapter, this thesis ultimately represents an attempt at unifying a complex set of goals, mechanisms and interfaces into a uniform set of functional services.

The most important thesis result, in this respect, was the establishment of a conceptual framework, the so-called *Unified Information Model*, the characterisation of which is intended to provide a clear path for the future development of the groundsystem architecture and methodologies. This should ultimately lead to the achievement of a fully automated Mission Operations System.

Appendix A

SSTL Protocol Specification

A.1 Constants and Definitions

Declared Algorithms:

RC4 The standard RC4 algorithm, as specified in [Sch96a, p.397-398], with a 128 bit seed and a 256 bit secret key size.

XOR encryption A simple XOR operation performed on every byte in a packet, against the values supplied from an array of byte values.

Declared Constants:

RC4KeySize = 16 The seed value for the RC4 algorithm in bytes.

SyncInterval = 25 The number of information frames per link direction before a resynchronisation is forced.

ConnectDelay = 10 The delay (in seconds) between connection attempts to ensure random re-seedings.

ConnectTimeOut = AX25_T3_TimeOut * 3 The SSTL timeout value is initialised to a value three times the value of the AX.25 T3 (inactive link) timer.

SecKeyBlockSize = 24 The size (in blocks) of a secret key.

SecKeyBlocks = 10 The number of blocks in a secret key set.

LocalKeys An array of size *SecKeyBlocks* to arrays of size *SecKeyBlockSize* containing the random secret key bytes values for the local key set.

RemoteKeys An array of size *SecKeyBlocks* to arrays of size *SecKeyBlockSize* containing the random secret key bytes values for the remote key set.

The following state records is declared:

Field	Type	Size	Purpose
X_Index	Byte	1	The X index into the RC4 seed vector
Y_Index	Byte	1	The Y index into the RC4 seed vector
RC4Seed	Byte	256	The RC4 Secret Key

Table A.1: *RC4State - RX4 State Record Definition*

Field	Type	Size	Purpose
Token	Unsigned Integer	4	Handshake token for authentication
SeqNum	Unsigned Integer	4	Sequence number initialisation value
RC4Seed	Array of Bytes	RC4KeySize	Seed values for RC4 algorithm

Table A.2: *ChallRespPkt - Challenge/Response Packet Type*

Field	Type	Size	Purpose
SeqNum	Unsigned Integer	4	New sequence number initialisation value
RC4Seed	Array of Bytes	RC4KeySize	Seed values for RC4 algorithm

Table A.3: *SyncPkt - Synchronisation Packet Type*

Declared Variables:

LockKeyIndex The position in a local key set for the next synchronisation encryption.

RemKeyIndex The position in the remote key set for the next synchronisation encryption.

LocSeqNum The sequence number of the next information frame to be sent.

RemSeqNum The sequence number of the next information frame to be received.

RC4LocState The local RC4 state record of type *RC4State*.

RC4RemState The remote RC4 state record type *RC4State*.

Term Definitions:

Client The connecting party that initiates the SSTL connection attempt.

Server The connection receiving party that responds with the authentication challenge.

A.2 SSTL Process Registration

All processes that intend to use SSTL as their transport mechanism must register with SSTL. All (unique) AX.25 Protocol Identifiers that a process intends to use, must also be registered, while indicating whether frames of a particular PID should be encrypted.

A.3 AX.25 Connection and Disconnection Procedures

When an AX.25 connection from a SSTL client to a server occurs, as addressed by the regular AX.25 callsign and the assigned SSTL secondary station identifier (SSID), AX.25 will notify the SSTL process of the establishment of a AX.25 connection.

Whenever either party wishes to break the connection, the regular AX.25 link disconnect procedure is followed.

A.4 SSTL Challenge and Response Mechanism

Upon notification of a AX.25 connection to its SSID, a SSTL server process will respond with an authentication challenge. The client process will await the challenge packet. Should no response be received by the SSTL server within *ConnectTimeOut* seconds, the AX.25 link will be disconnected. Should a AX.25 connection be established within *ConnectDelay* of the last connection attempt, the AX.25 link will also be disconnected by the SSTL Server in order to ensure proper re-seeding of its random number vectors.

A SSTL Challenge is a packet of type *ChallRespPkt* with all the fields filled with random byte values. The Challenge is then XOR encrypted with the first secret key of the *RemoteKeys* set.

Upon receiving a Challenge, a SSTL client will decrypt the packet using the first secret key in its *RemoteKeys* set. It will then respond with a Response Packet of type *ChallRespPkt* with the token value supplied from the Challenge packet, with the rest of the Response Packet being filled with random byte values. After XOR encrypting this packet with the *second* key in its *RemoteKeys* set, the Response packet is sent. The Client will now initialise its *LocSeqNum* and *RemSeqNum* variables with the values sent or received, respectively. It will also initialise its *RC4RemState* record with the byte values sent in the Response packet and its *RC4LocState* record with the byte values received in the Challenge record. The *LockKeyIndex* value is incremented and the *RemKeyIndex* value is set to a value of two.

Upon receiving a Response packet, the SSTL server will decrypt the packet using the *second* key in its *LocalKeys* set. Should the token received match the one sent, it will also initialise its *RC4LocState* and *RC4RemState* records with the byte values either sent or received. It will also increment its *RemKeyIndex* variable, set its *LockKeyIndex* variable to 2 and initialise its *LocSeqNum* and *RemSeqNum* to the values sent or received, respectively.

If no AX.25 link disconnect notification is received, the Client can assume that its authentication was successful.

A.5 Information Frame Exchange

All information frames with a PID registered for encryption, are first encrypted by the RC4 cipher using the *RC4RemState* record's state information, for either party. Each encrypted information frame is also prepended by a sequence number from the *RemLocNum* variable, which is incremented afterwards. The sequence number value is also included in the RC4 encryption operation.

After decrypting each encrypted information frame (as determined by PID value), using the *RC4LocState* record's state information, the sequence number received is compared to the *RemSeqNum* value expected. If a mismatch occurs, the AX.25 link is disconnected immediately, else the *RemSeqNum* variable is incremented.

A.6 Resynchronisation Procedure

Each time *SyncInterval* packets have been sent, a party will follow it with a packet of type *SyncPacket*, filled with random byte values and XOR encrypted with the *RemKeyIndex* key from its *RemoteKeys* set. Afterwards, the *RemKeyIndex* variable is incremented, the *RC4LocState* record is updated with the values sent. The *LocSeqNum* variable is also updated with the value sent. A resynchronisation packet is also prepended with an encrypted sequence number from the *RemSeqNum* variable, to guard against packet insertion attacks.

Appendix B

SUNSAT Database Schema

B.1 List of Defined Entities

The following entities are defined for the SUNSAT database schema:

- Telemetry Source (TImSrc)
- Telemetry Element Entries (TImElem)
- Unit Measure (TImUnit)
- Index of data type definitions (DataTypes)
- Session log storage entity (Session)
- Data storage entity for each type definition (TImByteData etc.)
- Calibration Meta-data (TImCalib)
- Index of calibration Procedures (TImCalibTypes)
- Calibration Lookup Definitions (CalibIndex)
- Indexed Lookup Tables (CalibIndexes)
- Index of stored session data (SessionIndex)

B.2 Entity Definitions

Each entity's attributes and other meta-data will be defined in the following sections. The *size* values are in bytes, while *null* indicates whether the actual field value can be left unspecified.

TImSrc

The TImSrc entity allows for the hierarchical organisation of all telemetry sources, such as satellite subsystems and Flight Software modules.

Attribute	Type	Size	Null	Key	Description/Purpose
SrcID	Unsigned Integer	4		Primary	Unique Identifier for Telemetry Source element
SrcName	Character String	20			Descriptive Name
SrcDesc	Character String	128	Yes		Long Description
SrcParent	Unsigned Integer	4	Yes	Foreign	Relation to TImSrc.SrcID for indicating hierarchical relationships

TImElem

Each telemetry element has its own entry in the TImElem table, which provides descriptive meta-data, such as the measuring unit, and indicates such attributes as whether the element should be calibrated.

Attribute	Type	Size	Null	Key	Description/Purpose
ElemID	Unsigned Integer	4		Primary	Unique Identifier for Telemetry element
ElemName	Character String	32			Descriptive Name
ElemShtName	Character String	20			Short Name for Export Description
ElemDesc	Character String	128	Yes		Long Description
ElemUnit	Unsigned Integer	2	Yes	Foreign	Relation from TImUnit.UnitID
ElemDataType	Unsigned Integer	2		Foreign	Relation from DataType.DataTypeID
ElemTImSrc	Unsigned Integer	4		Foreign	Relation from TImSrc.SrcID
Calibrated	Unsigned Integer	1			Bit 1 set if calibrated

TImUnit

This entity defines a list of standard measuring units.

Attribute	Type	Size	Null	Key	Description/Purpose
UnitID	Unsigned Integer	2		Primary	Unique Identifier for Unit measure
UnitName	Character String	8			Unit abbreviation
UnitDesc	Character String	64	Yes		Long Description

Data Type

Each data storage entity has an entry in this table, which also provides some descriptive meta-data on the type definitions.

Attribute	Type	Size	Null	Key	Description/Purpose
DataTypeID	Unsigned Integer	2		Primary	Unique Identifier for DataType
DataTypeName	Character String	20			Descriptive Name
DataTypeDesc	Character String	128	Yes		Long Description

Session

This table provides a detailed set of log entries for all the actions and events that occurred during a session, as defined by the SessionIndex entity.

Attribute	Type	Size	Null	Key	Description/Purpose
SessionID	Unsigned Integer	4		Foreign	Relation from SessionIndex.SessionID
Time_Stamp	Unsigned Integer	4			Time value in UNIX utime (in seconds)
SubSec	Unsigned Integer	1			Hundredth of a second time value
TImElem	Unsigned Integer	4		Foreign	Relation from TImElem.ElemID
DescStr	Character String	128	Yes		Description of action/event

Telemetry Data Tables

These entities, optimised for each element data type that the database schema allows, are the actual storage tables for the telemetry elements. With the exception of the *TImValue* attribute, all the telemetry data tables share the same structure:

Attribute	Type	Size	Null	Key	Description/Purpose
Time_Stamp	Unsigned Integer	4		Composite	Time value in UNIX utime (in seconds)
SubSec	Unsigned Integer	1		Composite	Time value in hundredth of a second
Source	Unsigned Integer	1		Composite	Relation from DataSource.DataSrcID
TImElem	Unsigned Integer	4		Composite	Relation to TImElem.ElemID
SeqNum	Unsigned Integer	1			Sequence number for variable extraction granularity
TImValue	Unsigned Integer	1			Attribute for TImByteData
	Unsigned Integer	2			Attribute for TImWordData
	Unsigned Integer	4			Attribute for TImLongData
	Double Precision Floating Point	8			Attribute for TImRealData
	Unsigned Integer	Undef.			Attribute for TImVarByteData
	Character String	Undef.			Attribute for TImTextData

DataSource

This entity defines a number of data sources from which individual telemetry elements were received.

Attribute	Type	Size	Null	Key	Description/Purpose
DataSrcID	Unsigned Integer	1		Primary	Unique Identifier for Data Source
DataSrcName	Character String	20			Short descriptive name
DataSrcDesc	Character String	128	Yes		Long Description

TImCalib

This table provides the calibration definitions for each telemetry element. Attributes include such value declarations as polynomial coefficients.

Attribute	Type	Size	Null	Key	Description/Purpose
TImElem	Unsigned Integer	4		Composite	Relation to TImElem.ElemID
TimeStart	Unsigned Integer	4		Composite	Valid time period start
TimeEnd	Unsigned Integer	4		Composite	Valid time period end
CalibType	Unsigned Integer	2		Foreign	Relation from TImCalibTypes.CalibID
IndexID	Unsigned Integer	2	Yes	Foreign	Relation from TImCalib.Indexes
MultValue	Floating Point	8			Multiplication coefficient
DivValue	Floating Point	8			Division coefficient
PolyA	Floating Point	8			Value for coefficient a of $x = ax^2 + bx + c$
PolyB	Floating Point	8			Value for coefficient b
PolyC	Floating Point	8			Value for coefficient c

TImCalibTypes

This entity defines the set of calibration procedures allowed by the database schema.

Attribute	Type	Size	Null	Key	Description/Purpose
CalibID	Unsigned Integer	2		Primary	Unique Identifier for procedure definitions
CalibName	Character String	20			Descriptive Name

CalibIndexes

Some telemetry elements must be calibrated by way of lookup procedures into indexed tables. This entity defines the set of lookup indexes.

Attribute	Type	Size	Null	Key	Description/Purpose
IndexID	Unsigned Integer	2		Primary	Relation from CalibIndexes.IndexID
IndexName	Character String	20			Descriptive Name

CalibIndex

The actual reference and lookup values for indexed calibration are defined in this table.

Attribute	Type	Size	Null	Key	Description/Purpose
IndexID	Unsigned Integer	2		Foreign	Relation from CalibIndexes.IndexID
IndexValue	Unsigned Integer	4			Lookup Value
RefValue	Floating Point	8			Reference Value

SessionIndex

Each period of activity on the ground-system is called a Session. This entity allows for defining such session periods in the database.

Attribute	Type	Size	Null	Key	Description/Purpose
SessionID	Unsigned Integer	4		Primary	Unique Identifier for Session
TimeStart	Unsigned Integer	4			Valid time period start
TimeEnd	Unsigned Integer	4			Valid time period end

Appendix C

Global Frame Pool

C.1 Global Frame Pool (GFP) Module

The function of this module is to pre-allocate a set of frames, provide the memory address of a single frame on request, and to keep track of unused frames. The only further requirement is that these operations be performed with the minimum overhead, both in terms of CPU clock cycles and memory usage.

C.2 Frame Pool Representation

The maximum size of an AX.25 frame is 274 bytes (see table 3.2), since the flag octets are dropped by the SCC driver. This is therefore also the size of each frame in the pool. For the effective management of free and allocated frames, a bitmap representation is proposed, as is already employed by the SFS file system for block allocation. By allocating multiple 35072-byte memory blocks, each bit in a 16-byte bitmap would translate to the offset of one of 128 frames in the memory block. This memory block size is determined by the 65KB limit imposed by the 16 bit OBC1 memory architecture.

A bitmap representation is considered more effective than, for example, a linked list. For a linked list at least 16 bits (for addressing the next node in the list) per frame would be required, compared to 1 bit for a bitmap representation. A bitmap would also require fewer memory accesses for searching (each byte represents eight frames), than a linked list. Given the 8-bit memory bus of OBC1, this is important.

C.3 Frame Request Mechanism

A process such as the SCC driver would place a function call to the GFP module to request a frame. The Modula-2 definition for such a function would be:

```
PROCEDURE GFP.RequestFrame() : SYSTEM.ADDRESS;
```

SYSTEM.ADDRESS is the definition of the memory address type in Modula-2.

The GFP module now searches its bitmaps for a byte with a value less than 256, i.e. a byte with not all its bits set, thus one with a bit which translates into a free frame's address. After determining the free bit, calculating the memory address of the free frame is trivial:

$$\text{max_frame_size} = 274$$

$$\text{frame_bit_number} = (\text{bitmap_byte_offset} * 8) + \text{bit_offset}$$

$$\text{frame_address} = \text{memory_block_address} + (\text{frame_bit_number} * \text{max_frame_size})$$

If no free frame is available, a value of NIL (zero) will be returned.

C.4 Frame Reference Passing Mechanism

To pass a frame reference from one RTX process to another, the following interface is provided by the GFP module:

```
PROCEDURE GFP.SendFrame( recipient      : RTX.channel;
                        offset         : CARDINAL;
                        frame_address  : SYSTEM.ADDRESS;
                        length         : CARDINAL) : CMAP_API.ObjectCode;
```

CMAP_API.ObjectCode is a reference to the Flight Software global code definition module's data type definition.

The parameter definitions are:

recipient The mailbox for the receiving RTX process.

offset The offset within the frame where the segment intended for the receiving process begins.

frame_address The memory address as received from either the GFP.RequestFrame() procedure or from a RTX mailbox.

length The *remaining* number of bytes in the frame segment intended for the receiving process.

The GFP module will then calculate the memory address as intended for the receiving process (`frame_address + mess.offset`), and place this value, along with the `length` value, in the `recipient` mailbox. When the receiving process is scheduled by RTX, it will use the regular RTX.Receive procedure to retrieve the message and to determine the address and length of the referenced frame.

C.5 Frame Deallocation Mechanism

When the final recipient in the protocol stack (such as SatFTP or the Command Processing Module) is done with a received frame, it must call the following GFP function to have the frame deallocated:

```
GFP.FreeFrame( frame_address : SYSTEM.ADDRESS ) : CMAP_API.ObjectCode;
```

The GFP module now simply calculates the correct bit position in its bitmaps and sets the bit to zero:

$$\text{frame_bit_number} = (\text{address} - \text{memory_block_address}) \text{ DIV } \text{max_frame_size}$$

Where the DIV function performs a divide operation which returns an integer, discarding any remainder.

Bibliography

- [Ack91] M.C. Ackerman, *RTX programmer's reference manual*, 1.1 ed., 1991.
- [Ack93] M.C. Ackerman, *A kernel for temporally correct reactive systems*, Master's thesis, Department of Computer Science, University of Stellenbosch, 1993.
- [AGS99] M. Allman, D. Glover, and L. Sanchez, *Enhancing TCP over satellite channels using standard mechanisms*, Request for comments: 2488, IETF, January 1999.
- [AHKO97] M. Allman, C. Hayes, H. Kruse, and S. Ostermann, *TCP performance over satellite links*, 5th International Conference on Telecommunication Systems, 1997.
- [ASD99] L. Apvrille, P. Senac, and M. Diaz, *A Java framework for spatial embedded systems*, 13th Annual AIAA/USU Conference on Small Satellites, 1999.
- [BC00] et al. B. Chang, *Document object model (DOM) requirements*, Working draft, W3C, December 2000.
- [BNT97] W.A. Beech, D.E. Nielsen, and J. Taylor, *AX.25 link access protocol for amateur packet radio*, 2.2 ed., Tucson Amateur Packet Radio Corporation, 1997.
- [Boo96] J. Boot, *Implementation and integration of ground and space segment communication software for the SUNSAT micro-satellite*, Master's thesis, Department of Electronic Engineering, University of Stellenbosch, December 1996.
- [BPSMM00] T. Bray, J. Paoli, C.M. Sperberg-McQueen, and E. Maler, *Extensible Markup Language (XML) 1.0*, Recommendation, W3C, October 2000.

- [BW93] A. Burns and A. Wellings, *Real-time systems and their programming languages*, Addison-Wesley, 1993.
- [Car97] A. Cardoza, *SUNSAT groundstation automation*, Tech. report, ESL, University of Stellenbosch, September 1997.
- [Car99] A. Cardoza, *A GSM FTP protocol for remote LEO gateways*, 2nd Annual Proceedings of the South African Telecommunications Networks and Applications Conference, pp.270-274, September 1999.
- [Cer82] V. Cerf, *Packet satellite technology reference sources*, Request for comments: 829, IETF, November 1982.
- [Cod70] E.F. Codd, *A relational model of data for large shared data banks*, Communications of the ACM, Vol. 13, No. 6, pp.377-387, June 1970.
- [Com93] Ameritech Mobile Communications, *Cellular digital packet data system specifications: Part 406: Airlink security*, Tech. report, CDPD, July 1993.
- [DA99] T. Dierks and C. Allen, *The TLS protocol version 1.0*, Request for comments: 2246, January 1999.
- [dB95] J.P. du Buson, *The integration of a link layer protocol on a satellite subsystem*, Honours report, Department of Computer Science, University of Stellenbosch, February 1995.
- [DB00] et al. D. Box, *Simple object access protocol (SOAP) 1.1*, Note, W3C, May 2000.
- [Dei90] H.M. Deitel, *Operating systems*, second ed., Addison Wesley, 1990.
- [DENP96] M. Degermark, M. Egan, B. Nordgren, and S. Pink, *Low-loss TCP/IP header compression for wireless networks*, 2nd ACM International Conference on Mobile Computing and Networking (Mobicom), 1996.
- [DMT96] R.C. Durst, G.J. Miller, and E.J. Travis, *TCP extensions for space communications*, 2nd ACM International Conference on Mobile Computing and Networking (Mobicom), 1996.

- [Fal00] D.C. Fallside, *XML schema part 0: primer*, Candidate recommendation, W3C, October 2000.
- [Fox84] T. Fox, *AX.25 amateur packet-radio link-layer protocol version 2.0*, ARRL, October 1984.
- [GM95] J. Gosling and H. McGilton, *The Java language environment: a White Paper*, <http://java.sun.com/whitepaper/java-whitepaper-1.html>, Sun Microsystems, 1995.
- [GM99] et al. G.W. Milne, *SUNSAT - launch and first six months' orbital performance*, 13th Annual AIAA/USU Conference on Small Satellites, 1999.
- [Gre98] E. Greenberg, *File transfer in the deep space environment: issues, desirable features, and protocol design*, SpaceOps, 1998.
- [Hay97] C. Hayes, *Analyzing the performance of new TCP extensions over satellite links*, Master's thesis, College of Engineering and Technology, Ohio Technology, August 1997.
- [Jac86] J. Jacobson, *YAPP protocol for packet radio binary file transfer*, Electronic document, version 1.1, June 1986.
- [Kar94] P. Karn, *Toward new link layer protocols*, QEX, pp.3-10, June 1994.
- [KG98] N.R. Kuo and E. Greenberg, *On-board file management and its application in flight operations*, SpaceOps, 1998.
- [IR96] P. le Riche, *Proposed implementation of diary and command & control software for SUNSAT*, Tech. Report 1.1, ESL, University of Stellenbosch, November 1996.
- [MA00] et al. M. Allman, *Ongoing TCP research related to satellites*, Request for comments: 2760, February 2000.
- [MCL98] A.H. Maury, A. Critchfield, and J. Langston, *Jswitch/Jsat: real-time and offline world wide web interface*, 12th Annual AIAA/USU Conference on Small Satellites, 1998.

- [MFKC98] R.J. Moore, K.P. Fregeolle, L.H. Kaye, and N. Clabough, *An analysis and implementation plan for incorporating Java into a heritage C++ flight operations center*, SpaceOps, 1998.
- [MHHS99] P.L. McKerracher, H.S. Han, D.B. Holland, and J.A.H. Stock, *Database applications in science data systems for low-cost satellite missions*, 13th Annual AIAA/USU Conference on Small Satellites, 1999.
- [Mic95] Micon, *Verfahren und Strukturen beim Flugbetrieb von Raumfahrzeugen*, Sponsored study, DARA, April 1995.
- [Mon98] F.J. Monrozier, *Common services in space ground systems - an event handler based on CORBA*, SpaceOps, 1998.
- [MTB95] O. Mohammadi, W. Toomy, and Lawrie Brown, *DUAL - A packet format for a new amateur radio link layer*, Sponsored Study, Department of Computer Science, The University of New South Wales, February 1995.
- [MvdM99] S. Mostert and B. van der Merwe, *Architecture and methodologies for micro-satellite communication software*, IEEE Africon, September 1999.
- [oISS97] Working Group on Information Systems and Services, *Data description languages*, Technical report, CEOS, 1997.
- [Par72] D.L. Parnas, *On the criteria to be used in decomposing systems into modules*, Communications of the ACM, Vol. 15, No. 12, pp.1053-1058, December 1972.
- [PR85] J. Postel and J. Reynolds, *File transfer protocol (FTP)*, Request for comments: 959, IETF, October 1985.
- [PW90a] H.E. Price and J. Ward, *PACSAT broadcast protocol*, ARRL 9th Computer Networking Conference, August 1990.
- [PW90b] H.E. Price and J. Ward, *PACSAT protocol suite: an overview*, ARRL 9th Computer Networking Conference, August 1990.

- [PWWL96] W. Polites, W. Wollman, D. Woo, and R. Langan, *Experiments with a simple file transfer protocol for radio links using enhanced trivial file transfer protocol (ETFTP)*, Request for comments: 1986, IETF, August 1996.
- [RC93] P. Rob and C. Coronel, *Database systems*, Wadsworth, 1993.
- [RF99] E. Rabenau and Dr. H. Fischer, *Analyses and evaluation of operational concepts for spacecraft with special focus on small satellites and low-cost options*, SpaceOps, 1999.
- [Sca88] E.L. Scace, *Various AX.25 state machines*, 7th Computer Networking Conference, 1988.
- [Sch96a] B. Schneier, *Applied cryptography*, second ed., John Wiley & Sons, Inc, 1996.
- [Sch96b] I. Scholl, *The design and implementation of the archive for the SOHO data*, SpaceOps, 1996.
- [SMM⁺97] A. Schoonwinkel, G.W. Milne, S. Mostert, W.H. Steyn, and K. van der Westhuizen, *Pre-flight performace of the communications payloads on SUNSAT, South Africa's first micro-satellite*, 8th Bi-annual Conference and Exhibition on Telecommunications in Southern Africa, 1997.
- [Sta94] W. Stallings, *Data and computer communications*, fourth ed., Macmillan, 1994.
- [Ste94] W.R. Stevens, *TCP/IP illustrated Volume 1*, Addison-Wessley, 1994.
- [Ste00] N.L. Steenkamp, *Development of the on board computer flight software for SUNSAT 1*, Master's thesis, Department of Electronic Engineering, University of Stellenbosch, December 2000.
- [SWP96] C. Sun, M. Windrem, and L. Picinich, *Application of world wide web (w3) technologies in payload operations*, SpaceOps, 1996.
- [Tri97] C.C. Tribelhorn, *A flexible environment for software development targeted for embedded systems*, Master's thesis, Department of Electronic Engineering, University of Stellenbosch, 1997.

- [TS99] W. Tai and D. Sweetnam, *A mission operations architecture for the 21st century*, SpaceOps, 1999.
- [vdM98] B. van der Merwe, *The AX.25 and SatFTP protocols: design and implementation*, Honours report, Department of Computer Science, University of Stellenbosch, February 1998.
- [Via96] P. Viallefont, *Distributed middleware for future ground segment architectures*, SpaceOps, 1996.
- [War90] J.W. Ward, *Store-and-forward message relay using micro-satellites: the UoSat-3 PACSAT communications satellite*, 4th Annual AIAA/USU Conference on Small Satellites, August 1990.
- [WL99] J.R. Wertz and W.J. Larson, *Space mission analysis and design*, third ed., Kluwer Academic Publishers, 1999.
- [WP90a] J. Ward and H.E. Price, *PACSAT file header definition*, ARRL 9th Computer Networking Conference, August 1990.
- [WP90b] J. Ward and H.E. Price, *PACSAT protocol: file transfer level 0*, ARRL 9th Computer Networking Conference, August 1990.