Web Accessible Real-Time Geospatial

Operations Via Satellite Communications

by

Alex Lemin Wu

TR03-159, June 03

Faculty of Computer Science
University of New Brunswick
Fredericton, N.B. E3B 5A3
Canada

Phone: (506) 453-4566
Fax: (506) 453-3566
E-mail: fcs@unb.ca
www: http://www.cs.unb.ca

Abstract

This research investigates and tests two satellite communication links for application to real-time seismic survey operations. The research includes an investigation of satellite system physical characteristics, a cost model and the software and system integration required to integrate satellite communications with real-time geospatial operations.

After intensive investigation of different satellite systems, Iridium and Orbcomm were chosen as candidates for experimental validation. An XML cost model was developed to estimate costs for various seismic survey operational scenarios involving Iridium and Orbcomm satellite systems. Both satellite systems were tested to determine their operational characteristics. Three different experiments transmitting packets of 108 bytes (simulating helicopter GPS information) were carried out at different frequencies, time of day and transceiver movement (static and dynamic).

Satellite communication system cost is about 10 times less per unit of data transmitted for Orbcomm compared to Iridium. Orbcomm has a much higher latency and lower update rate than Iridium. Our experiments determined an average latency for Iridium of 1465 ms while Orbcomm's average latency was 12 minutes 45 seconds given the same data transmission parameters.

Simple Object Access Protocol (SOAP) was also explored in this research for its possible role in satellite link geospatial information systems. Our results show that SOAP has good potential for use in client – server web applications for satellite cost estimation as part of satellite-based field operations.

Acknowledgments

First of all I would like to take this chance to express my many thanks to Dr. Brad Nickerson, my supervisor for his support, patience in guiding my research work at every step. Without his support and guidance it would not possible for me to finish my program.

I am also thankful to Jim McLellan, Peter Srajer and Dave Huff of Eagle Navigation Systems Inc. for their support in providing Kodiak Office, RGU software and seismic survey GPS data for experimental testing.

My thanks go to Dr. Peter Dare and the Geodesy and Geomatics Engineering Department of UNB for their generous permission to use the Hydrography Lab for our research.

I would like to thank Dr. Bernd J Kurz and all members of the examining board for taking time to evaluate my research work.

I appreciate all the technical support from the system support team of UNB Computer Science including Ivan Sears, Troy Cable and Sean Seeley. My appreciation also goes to Mary Kaye, Bruce Miller and Adam Wilson of Department of Electrical and Computer Engineering for the consultation on QNX 4.25 system and supply of equipment.

We also thank Dr. Claudia Iturriaga who provided financial support to purchase hardware necessary to complete our research experiments.

Table of Contents

List of Figures

List of Tables

List of Acronyms and Abbreviations

ACK --- Acknowledgment
API --- Application Programming Interface
CDMA --- Code Division Multiple Access
CDPD --- Cellular Digital Packet Data
COM --- Component Object Model
CPU --- Central Process Unit
CRC --- Cyclic Redundancy Check
DGPS --- Differential Global Positioning System
DTD --- Document Type Definition
FDMA --- Frequency Division Multiple Access
FTP --- File Transfer Protocol
GEO --- Geostationary Earth Orbit
GES --- Gateway Earth Station
GPS --- Global Positioning System
HTTP --- Hypertext Transfer Protocol
IDE --- Integrated Development Environment
IEEE --- Institute of Electrical and Electronics Engineers, Inc.
ISU --- Iridium Subscribing Unit
LAN --- Local Area Network
LEO --- Low Earth Orbit
MEO --- Medium Earth Orbit
MTS --- Multiplex Timing Serial
NCC --- Network Control Center
NTP --- Network Time Protocol
OEM --- Original Equipment Manufacturer
OMG --- Object Management Group
POSIX --- Portable Operating System Interface
PSTN --- Public Switched Telephone Network
RGU --- Remote Guidance Unit
SBM --- Short Burst Messaging
SC --- Subscriber Communicator
SDK --- Software Development Kit
SIM --- Subscriber Identity Module
SNTP --- Simple Network Time Protocol
SOAP --- Simple Object Access Protocol
TCP/IP --- Transmission Control Protocol/Internet Protocol
TDMA --- Time Division Multiple Access
TNC --- Threaded Neill Concelman
UHF --- Ultra High Frequency
UML --- Unified Modeling Language
VHF --- Very High Frequency
W3C --- World Wide Web Consortium
WAP --- Wireless Application
WML --- Wireless Markup Language

WMLScript --- Wireless Markup Language Script
XML --- eXtensible Markup Language
XSL/XSLT --- extensible Stylesheet Language/extensible Stylesheet Language Transformations
WSDL --- Web Service Description Language

Chapter 1. Introduction

Wireless communication has entered its fastest growth period in history since the development of technologies permitting wide spread deployment [Rappaport, 1996]. Real-time wireless information management in energy and resource operations requires fast and seamless information flow for data acquisition, management and operational decisions. Due to the swift advance of internet technology, communication over the web has become an efficient alternative to many currently applied communication methods, which is particularly advantageous for applications like real-time mobile information management [Nickerson and Shan, 2001].

In mobile distributed environments, applications related to current location often need to send and receive data dynamically. In geophysical surveys in remote areas, equipment movement by using helicopter to geophone locations as well as dynamic real-time information exchange between the pilot, base station and main office is necessary to guide the helicopter to the right location successfully and efficiently [Chatenay, 2000]. These tasks can be accomplished by wireless mobile telephone (cellular phone), radio modems or satellite communication link. In most cases, using radio modems is less expensive and more convenient since cell phone coverage may not be available. The disadvantage of using radio modems is that the maximum range of communication using commonly available licensed radios (with a 5 to 40 watt transmitter) is limited to around 40 kilometers (with repeaters). A satellite communication link becomes a good alternative choice in real-time geospatial operations [McLellan, 2001]. Recent technological advancements allow the deployment of satellite networks that provide voice and data transfer capabilities to any isolated corner of the globe [Ha, 2001].

1.1 Previous work

A simulated wireless real-time geospatial operations system was achieved successfully by Ying Shan of UNB during her graduate research [Shan, 2001]. A three-tier working prototype was designed and implemented using WAP, WML, WMLScript, XML, JavaServlet, Java Applet, HTML and JDBC techniques. An experiment using recorded helicopter flight data from Eagle Navigation Systems, Inc. demonstrated that the system could send from a WAP device simulator to the on-line central office the real-time geospatial information of a helicopter employed for the purpose of drilling holes and delivering equipment for seismic exploration operations. The movements and detailed information of this helicopter can be displayed, in real-time, from a web browser with a delay of less than one second. The system can also store all the received data in an Oracle database and therefore any existing historic flight path can be selected and reviewed by internet users from a web browser [Shan, 2001]. Kodiak flight path data files (KFP) were reformatted into XML before data was used in the simulator. This research builds on Ying Shan's completed research to investigate the middle layer using satellite systems for communications.

1.2 Research objectives

In the current Kodiak system, there is a limitation including a range limit of approximately 40 km of the radio frequency communication between the helicopter and base station and the need for expensive setup of radio modem transmission towers (e.g. when radio repeaters are required) [Eagle, 1998].

To improve the communications within the existing Kodiak system based on Ying Shan's research work, a satellite communication link becomes a potential choice for the

connections between helicopter, base station and head office. The objectives explored in this research are in three parts – physical system characteristics, cost and feasibility and software and system integration.

In the physical system characteristics part the focus is on finding if the selected satellite communication systems are able to handle (maintain a reliable data link) the high dynamics of aircraft (fixed wing and rotary) navigation, provide error checking functionality and keep data link "always on" without initiating a call (e.g. like CDPD (Cellular Digital Packet Data) for cell phone data service).

The second task is to explore if there is an appropriate active satellite system for real-time global geospatial operations. This task includes building a cost model and determining the long-term viability and target market for individual satellite communication systems.

The third part of the objectives is to integrate software and systems with the satellite system modems including experimental testing of satellite communication data link systems. The role of Simple Object Access Protocol (SOAP) in defining a software architecture for real-time geospatial operations is also a main topic in this part.

1.3 Thesis overview

The chapters in the thesis are organized as follows:

Chapter 2 is background introduction including Kodiak system architecture, satellite systems overview, XML and SOAP tools for the satellite cost model. In Chapter 3 the satellite cost model is described in detail including comparison of satellite systems, cost model design and implementation. Satellite data communication experiments for both Iridium and Orbcomm are introduced in Chapter 4 with system design and software

development. Chapter 5 presents satellite data transmission latency testing and analyzes the test results. Chapter 6 concludes the research and indicates future work.

Chapter 2. Background

A system called Kodiak, for real-time mobile geospatial operations has been designed and built by Eagle Navigation System, Inc. of Calgary, Alberta. Section 2.1 gives a brief introduction to the Kodiak system and section 2.2 describes a simulation experiment using the Kodiak system.

2.1 Kodiak system architecture overview

The architecture of Kodiak system is shown in Figure 2.1.

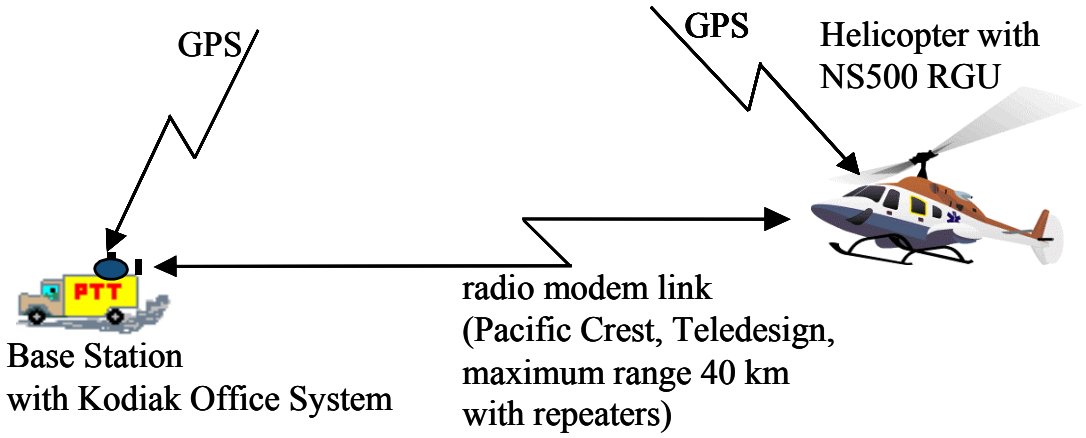


Figure 2.1 Architecture of the Kodiak system with radio frequency modems.

Kodiak is primarily used for helicopter guidance and management during seismic survey operations. The system consists of two major components, namely:

1) Kodiak Office software installed on an industrial computer with Windows 98, flat panel screen, data radio modem, UPS, UHF/VHF antenna cabling and a 12 volt power supply. The Kodiak Office base station generates missions for the remote helicopters, displays the current helicopter positions, and provides reports on activities [Eagle, 1998].

2) The NS500 Kodiak navigation system installed in a helicopter. The NS500 Remote Guidance Unit (RGU) consists of a vehicle-mounted DGPS navigation and guidance system connected via radio frequency links back to a single controlling base station

(Kodiak Office) [Eagle, 2001]. The NS500 system has a NovAtel 12-Channel GPS card working with an Omnistar OEM card providing DGPS corrections. The NS500 runs on an embedded 586-class computer using a QNX real-time operating system. There is a custom-built remote display for the pilot and a keypad for menu operation. The NS500 system allows the pilot to fly directly to the target without having to follow a map and confirm terrain features. The system also reduces time in "Dead Man's Curve" (i.e. when the helicopter has little forward motion making positional control more difficult.), minimizes the radio communication between pilot and base station and allows pilots to mark waypoints (such as a fuel cache) and return to them later [McLellan, 2001].

2.2 Simulation of Kodiak Office 2.5.1 and RGU500

Simulated radio frequency communication between Kodiak Office 2.5.1 and NS500 RGU was conducted on a Windows 2000 computer named Sylvius and a QNX 4.25 machine named QNX1 with the architecture shown in Figure 2.2. Kodiak Office 2.5.1 running on Sylvius sends task points with latitude, longitude and height information to a simulated helicopter running NS500 RGU on QNX1. Meanwhile, the simulated helicopter sends its own position information back to Kodiak Office 2.5.1. A virtual GPS simulator running on a computer named Ltempor with Windows 98 operating system is connected to QNX1 to generate GPS positions for the simulated helicopter. Figure 2.3 shows the NS500 RGU screen while a simulated mobile agent is approaching the target task point from Kodiak Office 2.5.1. Figure 2.4 shows the helicopter position information displayed on a Kodiak Office 2.5.1 screen. A VirtualGPS screen shot is given in Figure 2.5. Connections between Sylvius and QNX1, Ltempor and QNX1 are serial null modem

cables (Laplink Serial cable), which allow two ends to communicate with each other without a real modem.

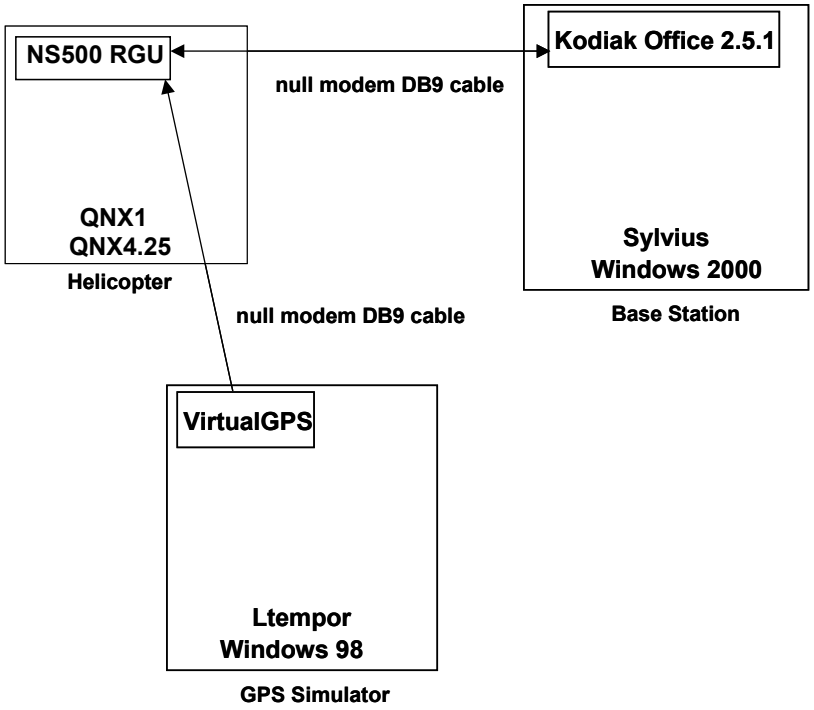


Figure 2.2 Architecture of Kodiak Office 2.5.1 and NS500 RGU simulation.

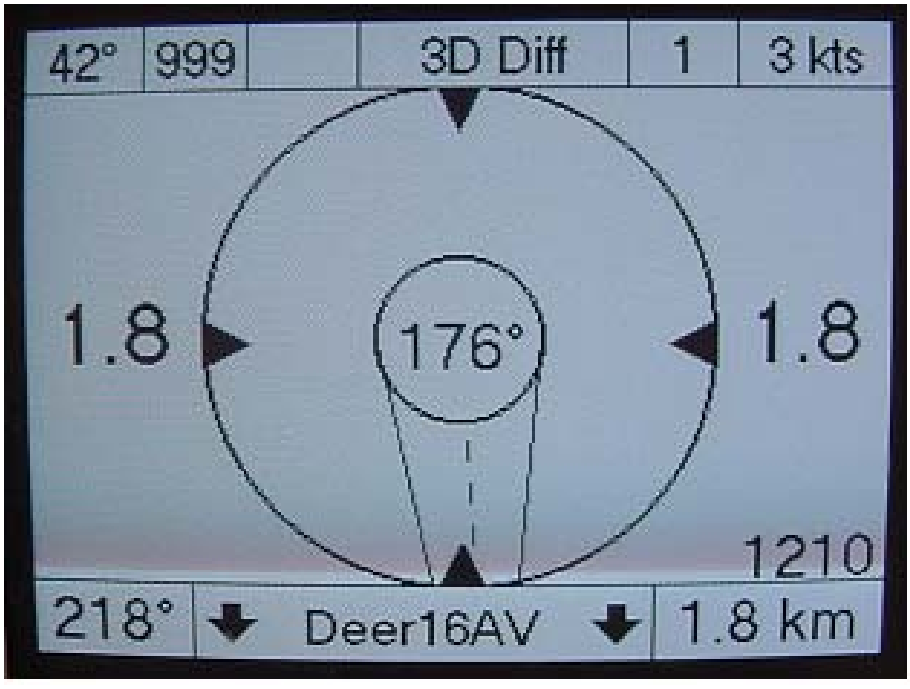


Figure 2.3 NS500 RGU software running on QNX1.

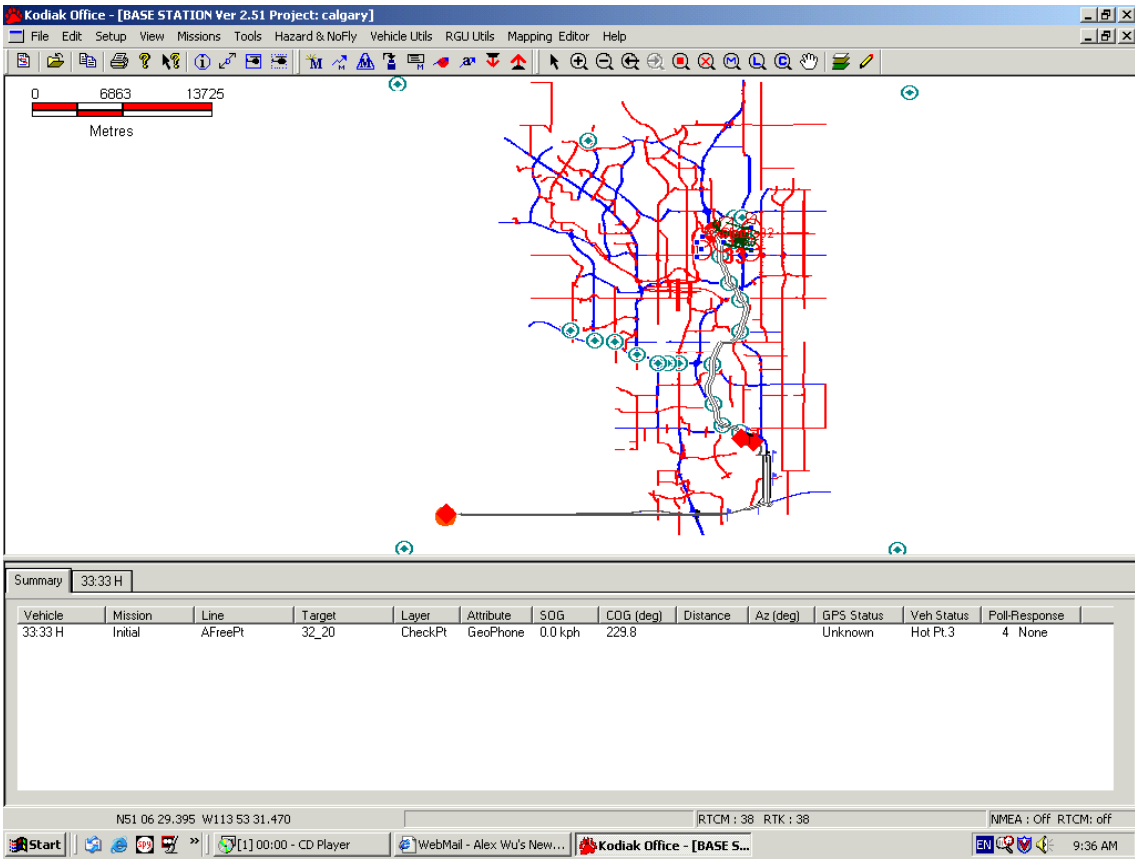


Figure 2.4 Kodiak Office2.5.1 with mobile agent position information.

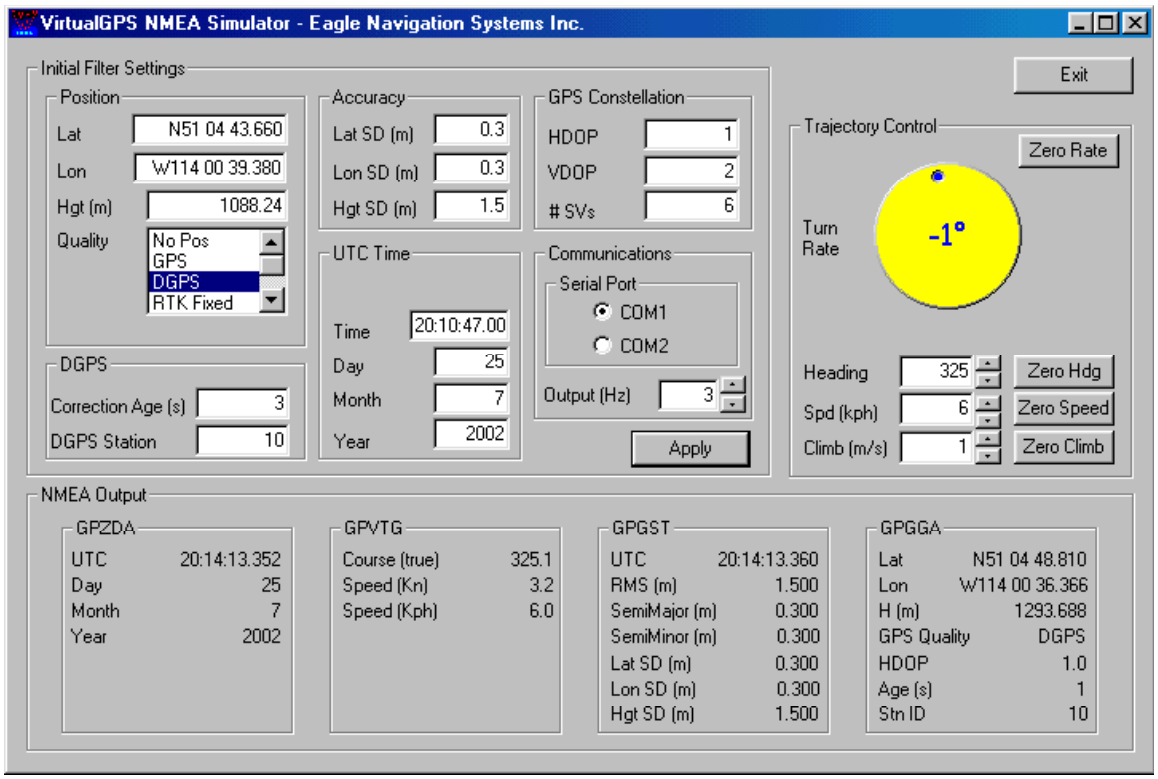


Figure 2.5 VirtualGPS software screen shot.

2.3 Satellite systems

Satellite systems are primarily categorized by their earth orbit characteristics. This section gives a brief introduction to each different satellite system and its typical technical elements.

2.3.1 Low earth orbit systems

LEO stands for Low Earth Orbit satellite constellation. LEO systems have a large number of satellites, flying in an orbit of a few hundred kilometers above the earth. Any location on the earth is able to see one of the satellites and the system works as a cellular phone system with a moving receiver/transmitter. With a relatively short distance to earth, round trip latency of transmission is theoretically low (but not in practice due to different satellite operational characteristics) [Compass, 1999].

Current operating LEO systems include Iridium, Orbcomm and Globalstar. Teledesic was scheduled to come into the market in 2005, but the Teledesic's project of high speed Internet service vial a constellation of satellites has been postponed due to financial and marketing reasons [Jung, 2002].

2.3.2 Geostationary earth orbit systems

A GEO (geostationary earth orbit) satellite has a 36,000 km high orbit and it circulates the earth directly over the equator. The satellite remains over the same earth location since it takes 24 hours for a round trip and it can see nearly 40% of the Earth's surface (due to its altitude). Weather satellites are usually geostationary. MSAT and Inmarsat are GEO satellite communication systems.

2.3.3 Medium earth orbit systems

MEO satellite systems have earth orbits with altitudes between a few hundred km to a few thousand km. Ellipso and ICO are two MEO systems. Individual MEO satellites can cover more of the Earth's surface since their orbit is higher than LEO satellites and therefore MEO can cause latency longer than LEO, but less than GEO [Compass, 1999]. Some MEO satellites have an elliptical orbit with their perigee (lowest altitude) significantly less than their apogee (greatest altitude).

2.3.4 Satellite systems technical elements

Major satellite systems currently in operation are Iridium (LEO), Orbcomm (LEO), Globalstar (LEO) and MSAT (GEO). Technical elements of these satellite systems are listed in tables 2.1 and 2.2. Further details can be found in Nickerson and Wu [2002].

Table 2.1 Technical elements of Iridium and Orbcomm satellite systems.

satellite system	orbit	Transmission service band	data rate	one-way propagation delay in ms (min, max)
Iridium (LEO)	780 km	1616.0 - 1626.5 MHz transmit, 1616.0 - 1626.5 MHz receive TDMA, FDMA	2,400 bps sustained, up to 10,000 bps burst with Direct Internet Data Service	(2.6, 8.2)
Orbcomm (LEO)	825 km	Uplink: 148.0 – 150.05 MHz downlink: 137.0-138.0 MHz, Packet Data	Transmit: 4800 bps	2.7

satellite system	first launched	Range	current market	comments
Iridium (LEO)	First: 1997, commercial service starts in 1998. 8 years designed satellite life	Global	aviation, construction, disaster relief/emergency, government, leisure travel, maritime and media.	Iridium contracted with Boeing for its network operations
Orbcomm (LEO)	First: 1991, 4th launch in 2000, 4 year designed satellite life	Global, near continuous between the polar circles	ocean vessel track and control, avionics weather	Orbcomm LLC is the managing company since Aug 2001.

Table 2.2 Technical elements of Globalstar and MSAT satellite systems.

satellite system	orbit	Transmission service band	data rate	one-way propagation delay in ms (min, max)
GlobalStar (LEO)	1,410 km	2483.5 - 2500.0 MHz transmit, 1610.0 - 1626.5 MHz receive, CDMA	7,200 bps sustained	(4.6, 11.5)
MSAT (GEO)	36,000 km	1530-1559 MHz transmit 1631.5-1660.6 MHz receive	4,800 bps	(270, 400)

satellite system	first launched	Range	current market	comments
GlobalStar (LEO)	1999, designed satellite life: 7.5 years	Within +/- 68° latitude of the equator	High-quality voice, SMS, packet-switched and asynchronous data.	Finalized agreement on debt restructuring and new business model, made Chapter 11 filing on Feb 15, 2002
MSAT (GEO)	Apr.20, 1996, designed satellite life: 12 years	N/C America, northern S America, Caribbean, Hawaii, up to 250 km offshore	Transportation, utility, oil & gas, government, maritime, and resource industries	secure and reliable voice.

2.4 XML and SOAP

XML is a markup language designed to describe and carry data with user-defined tags defined in a Document Type Definition (DTD) or Schema. Several XML tools have been used during the development of XML satellite cost estimation model software, and these are described briefly. Simple Object Access Protocol (SOAP) is also explored in this research for its possible role in software designed for estimating and exchanging costs associated with satellite communication use in field operations.

2.4.1 XMLSpy

XMLSpy is a well-developed and widely used XML development environment. XMLSpy is an industry-standard tool for designing, editing and debugging in various

XML related technologies and protocols including XML, XML Schema, eXtensible Stylesheet Language/eXtensible Stylesheet Language Transformations (XSL/XSLT), SOAP, Web Service Description Language (WSDL) and Web Service technologies. Version 5 has functions for HTML-XML conversion with C, C++ and Java code generation [Altova, 2002].

In this research XMLSpy IDE was used to test XML and SOAP file generation and editing. The XMLSpy IDE reads from a DTD file provided by the user and generates an XML file structure with tag names. With the editor, the user fills in all the tag values accordingly. XMLSpy IDE can create SOAP requests by reading a WSDL file from a SOAP server instead of a DTD. XMLSpy IDE provides a straightforward, flexible and functional user interface and it was very useful in creating satellite cost model XML and SOAP files.

2.4.2 Xerces XML Parser

Xerces XML Parser from Apache provides several XML functions such as counting the number of tags or levels and parsing XML files according to a DTD to validate XML files. Xerces Parser is written in portable C++ and follows the W3C XML 1.0 specification. Error messages generated by the Xerces Parser are very clear, which helps a user to locate and correct errors easily and efficiently [Apache, 2001]. Xerces Parser was used in this research to validate both the basic satellite cost elements XML file and the satellite project cost XML files generated by CostXML_Generator, the software we developed for satellite system cost (see section 3.3.2).

2.4.3 SOAP exploration

SOAP is a lightweight information exchange protocol. SOAP has been used in a decentralized, distributed environment [BEKL, 2001]. SOAP uses an XML structure to generalize the file format for information exchange between applications using HTTP on the Internet. There are three parts in a SOAP message – envelope, header and body. The envelope is mandatory and it defines what is in a message and who should deal with it. An optional SOAP header has information describing the SOAP message. The SOAP body is mandatory and it contains SOAP requests and responses. SOAP is used in communications over HTTP but it can potentially be used in combination with a variety of other protocols [BEKL, 2001] [W3C, 2002]. SOAP must use a SOAP Envelope Namespace and SOAP Encoding Namespace [W3C, 2002].

SOAP is simple and extensible, as well as platform, technology and programming language independent. Due to its character-based structure, SOAP is compatible with most existing firewalls and is a W3C standard. SOAP seems promising for widespread use in business information exchange on the Internet [W3C, 2002]. DreamFactory Web Services of DreamFactory Software, Inc. at Lost Gatos, CA and Enterprise Web Services of WebMethods at Fairfax, VA are two commercial products currently using SOAP 1.1 [Winer, 2002].

The SQLData SOAP Server from SQLData Systems, Inc. was downloaded and tested in this research for SOAP information exchange. By connecting to a SQLData SOAP Server and fetching its WSDL file, a client enters parameter values to create a SOAP request according to the WSDL file structure. The client then sends the request to the SOAP server through an Internet browser. A SOAP response is fed back to the client with

the information requested [SQLData, 2002]. Further details of the SOAP architecture and operations are given in section 3.3.4.

Chapter 3. Cost estimation model for satellite survey operations

3.1 Satellite communication components

After extensive research on currently active satellite systems, Iridium and Orbcomm, both LEO systems, were chosen as candidates for satellite communication experiments in this research [Nickerson and Wu, 2002]. Iridium represents a low latency data transmission service over a continuous dedicated circuit switched voice link using TDMA and FDMA technologies. Iridium has several kinds of antennae for data transmission, including the Motorola external magnetic antenna and Sensor Systems S67-1575-90 Iridium/GPS antenna. Both of these antennae are small and easily mounted on a mobile agent such as a helicopter.

Orbcomm is the sole existing satellite system that provides only packet data transmission service. Orbcomm also has one of the lowest operational costs of any satellite communication system, which makes Orbcomm worthy of consideration for functionality testing in this research.

3.1.1 Iridium system

With 66 active satellites in a constellation, Iridium Satellite provides global mobile satellite voice and data services with complete coverage of the Earth. Iridium Satellite LLC, the new owner since November 2000 (after the previous owner Iridium LLC filed for bankruptcy in August 1999), is owned by private investors including Baralonco NV of Netherlands Antilles (24.3%, controlled by Saudi Prince Khalid bin Abdullah bin Abdulrahman), Bareena Holdings Party Ltd of Australia (26.9%, owned by Michael Boyd), Milport Associates SA of Panama (8.9%, owned by Inepar, Brazil) and Syndicated Communications Inc of USA (26.9%, controlled by Herbert Wilkins). These

four main shareholders hold 87% of Iridium Satellite LLC [Hopkins, 2001]. Iridium Satellite LLC has a contract with the Boeing Company to operate and maintain the Iridium satellite constellation and network. Motorola continues to be the major equipment supplier to Iridium under commercially acceptable terms. Iridium Satellite LLC has signed a US$72 million contract with the US Department of Defense, under which 20,000 government employees will have unlimited usage of airtime over the Iridium network for three years [Goldstein, 2000] [Analysis, 2002].

3.1.1.1 Iridium equipment

Manufactured by Motorola exclusively for the Iridium satellite system, Motorola 9500 and 9505 satellite phones are the most commonly used Iridium equipment in both Iridium voice and data services. The Motorola 9500 was used in this research.

The Iridium Data Kit provides the hardware and software required to establish an Iridium data call with a Motorola 9500 or Motorola 9505 portable satellite phone. The Iridium data kit includes a data adapter and DB9 serial cable to provide a connection to a Motorola 9500 or 9505 portable satellite phones. The Iridium World Data Services CD contains all software and documentation required to install and configure Iridium World Data Services on a computer.

Iridium has other more expensive transceivers for mobile aircraft communication. Airsat I, developed by Honeywell for Iridium, provides reliable and high quality two-way satellite communications (voice only) for light aircraft over a single digital channel in Iridium's global satellite system (without support for data transmission). Airsat I includes one transceiver unit, a handset and a specially designed Satcom blade antenna. With its powerful RF output (6 watts, much higher than the 0.57 watts of a Motorola 9500 or

9505) Airsat I is so far the best choice for communication under heavy blade rotation in aircraft. Airsat I's equipment and installation cost is very high (e.g. US$29,900 for equipment and US$20,000 for installation per aircraft). SatTalk II is another Iridium product for aircraft satellite communication using both voice and data. SatTalk II was developed by Icarus and is used with a Motorola 9505 satellite portable phone. SatTalk II provides clear telephone communications and Internet access in an aircraft cockpit and cabin. SatTalk II costs approximately US$7,000 to purchase and install for one aircraft [Nickerson and Wu, 2002].

Different types of Iridium modems are available for satellite data transmission using a modified Motorola L-band transceiver (LBT). The 9500 Iridium Modem has an Internal Subscriber Identify Module (SIM) Card Reader (CDM9500I35-I). Ruggedized modem A00002LA-E has an external SIM Card Reader and modem A00002LA-I comes with an internal SIM Card Reader. The Iridium modems use Time Division Duplex as a duplexing method, TDMA/FDMA as a multiplexing method and have a standard RS-232 (AT command) interface. Average output power is from 0.60 W to 0.62 W.

Two kinds of antennae were used in our Iridium data transmission testing. The Motorola external magnetic auxiliary antenna works as an accessory for one Motorola 9500 phone. A Sensor Systems S67-1575-109 Iridium/GPS antenna manufactured by Sensor Systems Inc., Chatsworth, CA is connected with a second Motorola 9500 phone. Both antennae can be installed on a mobile agent such as a helicopter. The Sensor Systems antenna is designed to work in aircraft and operate under rotating helicopter blades. The Sensor Systems antenna is connected to a Motorola 9500 phone through a 10 foot (3 m) antenna with a standard TNC connector at each end to satisfy the requirement

of total maximum 3dB signal loss between satellite antenna and transceiver. Figure 3.2 shows the equipment used in Iridium system testing.

Iridium data kit

Sensor Systems S67-1575-109
Iridium/GPS Antenna.

Iridium Motorola 9500

Motorola 9500 with data adapter

Figure 3.1 Iridium equipment used for our research.

3.1.1.2 Iridium technologies and services

The Iridium satellite system provides both voice and data services. Dial-Up Data Service from Iridium allows a user to send data from a computer to an end user with a computer, a corporate network/LAN or Internet Service Provider (ISP) by dial-up connection. The data transmission is routed through the Iridium satellite network and the maximum data rate is 2,400 bps [Iridium, 2002b]. Roundtrip system latency for circuit switched Dial-Up data is approximately 800 ms. Most of this latency is due to Global System Mobile (GSM) processing and only a few tens of milliseconds is due to propagation delay [Nickerson and Wu, 2002].

Iridium provides Direct Internet Data Service in which data is sent from a computer directly to the Internet via an Iridium satellite phone and dedicated servers at the Iridium

gateway. The data transmission rate for Direct Internet Data Services can burst up to 10,000 bps by using transparent compression technology [Iridium, 2002a].

The Iridium Mobile Terminated Data Service shown in Figure 3.2 is the architecture we explored in this research for Iridium system data communication latency. Mobile Terminated Data Services provides a data connection between two computers. Data calls originate from a computer connected to an Iridium phone with a data adapter. On the receiving side data calls terminate on a computer connected to either an Iridium phone or a land phone on the Public Switched Telephone Network (PSTN). This research explores the first scenario of Iridium Mobile Terminated Data Service. The data rate for Iridium Mobile Terminated Data Service is 2,400 bps [Iridium, 2002c].

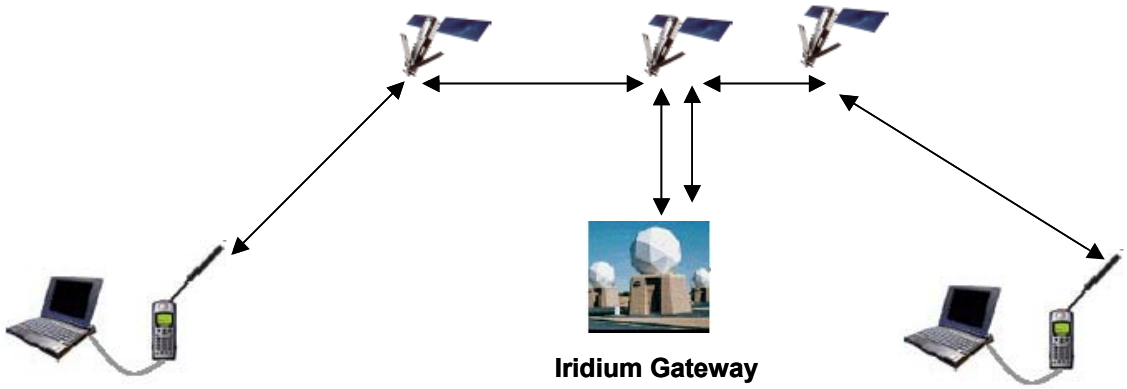


Figure 3.2 Iridium Mobile Terminated Data Service [Iridium, 2002c].

Iridium is currently developing new Short Burst Messaging (SBM) services that will provide low latency two-way messaging from small data messaging terminals (message size of 50-75 bytes). SBM service is targeted for unattended sensor, alarm and control applications [Iridium, 2002d], and is planned to start service early 2003. The SBM service requires the use of a new software data kit and a Motorola 9522 L-Band

transceiver (called a short burst data terminal). SBM will not work with Motorola 9500 and 9505 Iridium telephones.

Short Burst Messaging data requires different AT commands to dial up (circuit switched data). Latency for Short Burst Messaging data messages from a mobile device to the Gateway in Arizona is expected to be in the order of ten seconds or less. Delivery of a short burst data message from one mobile device to another will incur at least double the single hop latency. Additional latency from Internet traffic routing of the message will apply. Mobile terminated messages will not be immediately delivered, but held in a Gateway mailbox until the mobile device either polls the Gateway or sends a message [Nickerson and Wu, 2002].

Iridium data service charges include a monthly fee, an activation fee and per-minute charge. The per-minute charge only applies to the calling side. Iridium service in this research was subscribed from Preferred Communications in Creedmoore, NC. With a package of 1000 minutes airtime prepaid at US$0.68 per minute for data communication from ISU to ISU, the activation fee is US$15 and the monthly fee is US$19.95 per handset. The package is valid for 12 months and remaining airtime minutes can be rolled over to the next 12-month period [Nickerson and Wu, 2002].

3.1.2 Orbcomm system

Orbcomm provides global 2-way data services via low earth orbit (LEO) Satellites and ground infrastucture. Data is first sent from a satellite subscriber communicator (SC) to the Orbcomm satellite. An Orbcomm satellite then sends data to a Gateway Earth Station (GES). The GES relays the data message to the Network Control Center (NCC) either through satellite or ground line. NCC routes the message to the recipient by email,

phone line or fax. Figure 3.3 shows how a data message is transmitted through the Orbcomm system [Orbcomm, 2002].

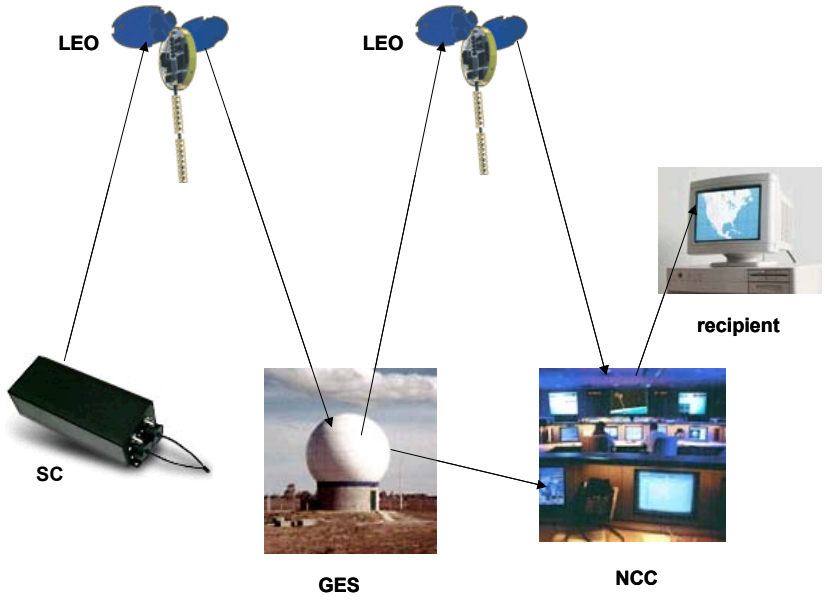


Figure 3.3 Orbcomm system data communication routine (from [Orbcomm, 2002]).

Orbcomm currently has 35 satellites in orbit, and is licensed by the FCC to launch and operate up to 48 satellites. Orbcomm is designed for short packet (0.5 second) transmission.

Orbcomm filed for Chapter 11 bankruptcy protection in September 2000. In August 2001 International Licensees LLC, a consortium of Orbcomm licensees and affiliates, purchased the business and assets of Orbcomm Global, L.P. and its other entities. The consortium includes OHB Systems GMBH, Orbcomm Asia Ltd. and other private investors. The new company was incorporated as Orbcomm LLC [Orbcomm, 2002].

3.1.2.1 Orbcomm equipment

There are several kinds of Subscriber Communicators (SCs) in the current market manufactured for Orbcomm system data communication. The Panasonic KX-G7101

satellite communicator with GPS is currently available in the market. A newer version, the Panasonic KX-G7201, is also available with a Software Development Kit (SDK). An OEM board for the Panasonic satellite communicator was scheduled to be on the market in January 2003.

The Magellan GSC-100 is another Orbcomm SC with a choice of antennae including roof mount, truck mount and magnetic mount. The GSC-100 development kit is also available with one Magellan Satellite Modem OEM Board, power supply interface board, choice of fixed or mobile antennae, data-power cable extension, AC power adapter, satellite PC software, evaluate® software, interface cables and user manual and reference guide.

The Q1500 development kit from Quake Global, Inc. was used in this research for Orbcomm system data communication testing. The Q1500 is designed for remote monitoring and control applications using the Orbcomm system with a very rugged military grade packaging. The Q1500 module combines high performance with a reasonable price and is a good solution for developers who need to integrate a satellite-based communications transceiver into customized applications. The Q1500 satellite transceiver (one OEM board and one adapter) unit included in the development kit provides two serial ports for communication with the host application. One serial port works as a Multiplex Timing Serial (MTS) port and fully supports the Orbcomm Serial Interface Specification. The second port is called a Logger port and can be custom programmed to support application specific communications or used as a monitoring and debugging port. Power at 12 V +/- 10% is required by the Quake 1500 modem with strict adherence to this voltage requirement over the full 3 A range. Both serial ports on the

Quake 1500 data adapter were connected to the serial ports of the testing computer (Broca or QNX1) through a DB9 cable. A magnetic whip antenna provided in the Quake 1500 development kit can be mounted on a helicopter or other mobile device. Figure 3.4 shows the equipment used for Orbcomm testing.

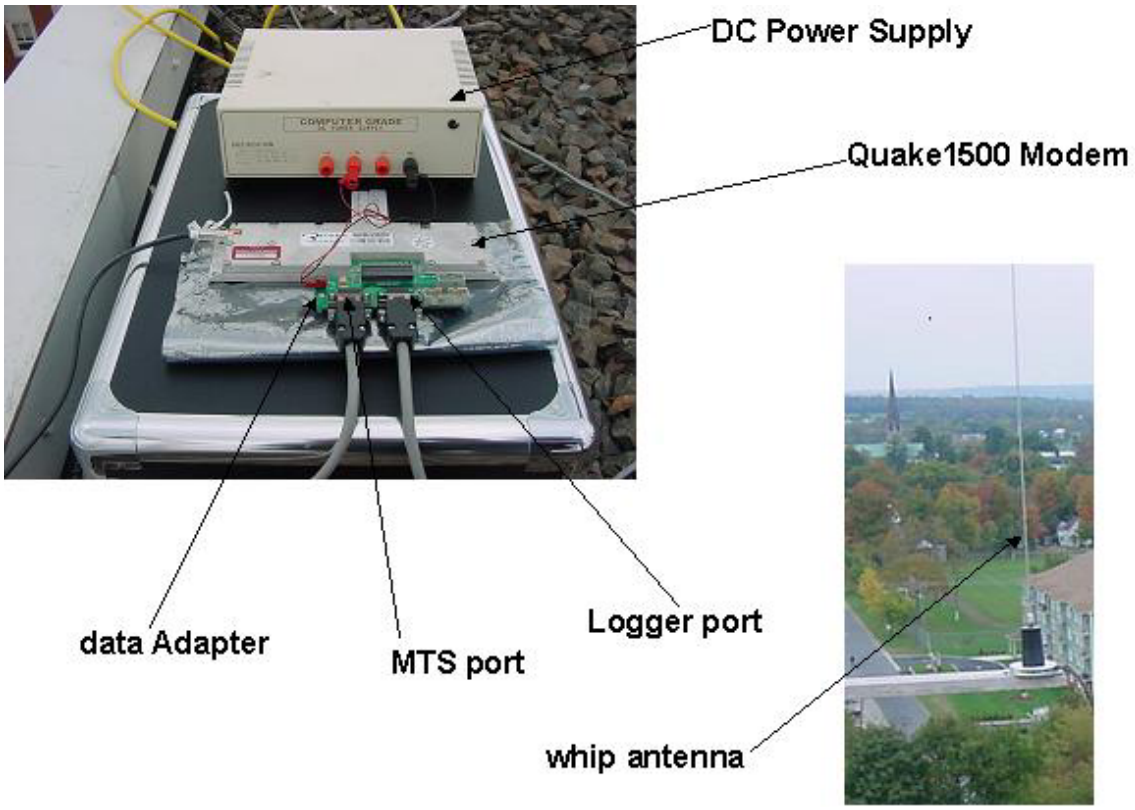


Figure 3.4 Orbcomm equipment.

3.1.2.2 Orbcomm technologies and services

Orbcomm only provides small packet data transmission. The practical maximum Orbcomm message size is 2000 bytes. If a message needs to be stored in the satellite and then forwarded (Orbcomm Globalgram Service) when the satellite is not in view of a ground station, each message can contain up to 229 characters for sending and 182 characters for receiving.

Orbcomm service charges include a monthly fee, an activation fee and a charge per byte. Charge per byte applies to both incoming and outgoing messages. Magellan GSC-100 or Steller ST2500 subscription services for the Orbcomm satellite system do not have per-byte charges, but the sending frequency is as low as one message every 5 to 10 minutes.

The Orbcomm data communication service used in this research is from SkyTrac, Vancouver, BC. In our case charges included a monthly fee ($70 + tax) and an activation fee ($95 + tax) for unlimited bytes of data transmission.

3.1.3 Summary of equipment and subscription cost

A significant amount of Iridium and Orbcomm satellite equipment and accessories were used in our research with computers and cables. Table 3.1 shows a summary of all capital equipment and satellite system subscription costs that we used during the course of this research.

Table 3.1 Summary of equipment and subscription costs for this research (includes taxes, licensing and shipping).

category	equipment	quantity	unit price		cost
Iridium	Motorola 9500 kit (new)	1	$1,167.83		$1,167.83
	Motorola 9500 kit (used)	1	$951.43		$951.43
	Sensor Systems Iridium Aircraft Antenna	1	$1,069.58		$1,069.58
	Data kit for Motorola 9500	2	$329.83		$659.66
	Motorola Data Adapter SYN 7023A	1	$87.01		$87.01
	10 foot low-loss coax cable assembly	1	$172.08		$172.08
	60 foot serial cable	2	$40.38		$80.76
	Subtotal (capital)				**$4,188.35**
	activation fee	2	$22.89		$45.78
	monthly access fee	5 months	$70.38		$351.90
	prepaid airtime package	1000 minutes	$1.04		$1,040.00
	Motorola phone repair	1	$448.65		$448.65
	Subtotal (service)				**$1,886.33**
	Total Iridium				**$6,074.68**
Orbcomm	Quake 1500 Development kit	1	$1,967.49		$1,967.49
	12V stable DC power supply	1	0		0
	Subtotal (capital)				**$1,967.49**
	activation fee	1	$104.79		$104.79
	Deactivation fee	1	$49.99		$49.99
	monthly fee	4 months	$78.60		$314.39
	Subtotal (service)				**$469.17**
	Total Orbcomm				**$2,436.66**
Computers	Laptop (IBM A20m, PIII, 667 MHz)	1	0		0
	Broca (IBM 300GL PII, 400 MHz)	1	0		0
	Sylvius (IBM 300GL PII, 450 MHz)	1	0		0
	Ltempor (Dell OptiPlex PIII, 500 MHz)	1	0		0
	QNX1 (IBM 300GL PII, 166 MHz)	1	0		0
	10 foot serial cable	1	$10.00		$10.00
	10 foot laplink serial cable	2	$18.00		$36.00
	power inverter to run laptop in a car	1	$44.11		$44.11
	Total Computers				**$90.11**
Grand Total					**$8,601.45**

3.2 Cost estimation model design

A satellite system cost estimation model was designed and implemented in C++. This cost model is designed to present the suitability of satellite system used in seismic survey operations from a business perspective. Given the basic cost elements of satellite systems such as monthly charges, per-minute cost, helicopter hourly cost, the satellite system cost

estimation application calculates the total system cost for different seismic survey operations and creates a project cost file in XML.

3.2.1 Cost estimation model equations

The satellite system cost estimation calculations are based on a series of equations designed according to various seismic project scenarios. The total cost *S* for one seismic project involving one or more helicopters is a function of *d*, the number of operational days per project. The cost *S* (in Canadian dollars) is as follows:

$$S(d) = A + B + C + D \tag{3.1}$$

where *A* = satellite communication system cost, *B* = helicopter cost, *C* = company support cost and *D* = initial project setup cost.

There are three parts in cost *A*:

$$A(d) = A_1 + A_2 + A_3 \tag{3.2}$$

where A_1 = the satellite communication system cost in helicopter(s), A_2 = the satellite communication system cost at base station and A_3 = the satellite communication system cost at head office (only standard Internet connection is required at head office for Orbcomm satellite communication with helicopter(s) or base station). They are computed (in Canadian dollars) as follows:

$$A_1(d) = (1+T)\sum_{i=1}^{s}\left(R_{si}\left(M\left\lceil \frac{d_i}{31} \right\rceil + d_i\left(60C(h_i - y) + r\left(60h_i(2f)p - b\right)\right)\right) + R_{ei}d_i\left(\frac{E}{7} + \frac{G}{n}\right)\right) \tag{3.3}$$

$$A_2(d) = (1+T)\sum_{i=1}^{2}\left(R_{si}\left(M_i\left\lceil \frac{d}{31} \right\rceil + d\left(60C_i(h - y_i) + r\left(60h(2f)p - b\right)\right)\right) + R_{ei}d\left(\frac{E_i}{7} + \frac{G_i}{n}\right)\right) \tag{3.4}$$

$$A_3(d) = (1+T)\left(X\left\lceil \frac{d}{31} \right\rceil\right) \tag{3.5}$$

where M = monthly cost, C = air-time calling cost per minute, h = number of operational hours per day, y = free airtime in hours per day included in monthly fee**,** r = cost per byte additional to bytes included in monthly cost, f = number of message transmissions per minute in one direction (multiply by 2 for bi-directional charge), p = number of bytes of each message, b = daily free bytes included in monthly fee, d = number of total operational days per project, d_i = number of operational days for satellite transceiver i, E = equipment weekly rental cost, G = equipment purchase cost, X = monthly fee for standard Internet access, n = number of days in capital period, T = tax rate, s = number of satellite transceivers, R_{si} = currency exchange rate for service charge and R_{ei} = currency exchange rate for equipment purchase.

The second cost B of flying the helicopter (s) is computed as follows (in Canadian dollars):

$$B(z) = (1+T)\sum_{i=1}^{z} d_i\left(Lh_i + Fh_i + Hh_i + I + Y\right) \tag{3.6}$$

where L = hourly cost for hiring pilot(s), F = hourly fuel cost, H = hourly helicopter rental cost, I = daily insurance cost, d_i = number of days of flying helicopter i, h_i = number of daily hours of flying helicopter i, Y = daily maintenance cost and z = number of helicopters.

Assuming that a commercial company carries out a seismic survey project operation, the equation for calculating the third cost C of the commercial company support (in Canadian dollars) is

$$C(d) = (1+T)(d)\left(Q + \sum_{i=1}^{m} h_i\left(P_i\right)\right) \tag{3.7}$$

where P_i = the individual commercial company's hourly personnel cost for person i,

Q = the commercial company's daily equipment cost, d = number of working days, h_i = number of daily working hours for person I and m = the number of employees.

The cost D of initial setup for one project has four parts as follows:

$$D = D_1 + D_2 + D_3 + wU \tag{3.8}$$

where D_1 = the initial setup cost in helicopter(s), D_2 = the initial setup cost at the base station, D_3 = the initial setup cost at head office, U = cost to set up one repeater station for radio modems and w = number of repeater stations. They are computed (in Canadian dollars) as follows:

$$D_1 = (1+T)\left(\sum_{i=1}^{k}(I_i + V_i R_i)\right) \tag{3.9}$$

$$D_2 = (1+T)\left(\sum_{j=1}^{2}(J_j + V_j R_j)\right) \tag{3.10}$$

$$D_3 = (1+T)K \tag{3.11}$$

where I_i = the installation cost for helicopter i, V_i = the activation cost of satellite transceiver/radio modem i in helicopter, J_j = the installation cost for satellite transceiver/radio modem j at the base station, V_j = the activation cost of satellite transceiver/radio modem j at the base station, K = the installation cost at head office, k = number of satellite transceivers/radio modems in helicopter(s), base station or head office, R = currency exchange rate for activation fee, i = 1,2,…k and j = 1 or 2 as only two satellite transceivers are likely for the base station.

3.2.2 Cost elements and latency comparison of active satellite systems

Key cost elements in equations (3.3), (3.4), (3.9) and (3.10) for different satellite transceivers of Iridium, Orbcomm, Globalstar, MSAT and radio modem were compared (see Table 3.2) to guide satellite system selection for research testing. M is monthly cost, C is air-time calling cost per minute, r is cost per byte additional to bytes included in monthly cost, y is free airtime in hours per day included in monthly fee, G is equipment purchase cost and V is the activation fee (all prices in Table 3.2 are in Canadian dollars unless they are indicated by "US$").

Table 3.2 Comparison of satellite communication system cost and latency.

Satellite system		M	C	r	y	G	V	latency
Iridium	Motorola 9500	US$19.95	US$0.68	$0	0	$1,440	US$ 20	Data over dial-up: 800 ms round trip; data over Direct Internet Data: 5-10 s to 5 -10 min
	Motorola 9505					$2,395		
	SatTalk II					US$4,495		
	Airsat I					US$ 29,995		
GlobalStar	Qualcomm GSP 1600	$365	$1.39 per extra minute	$0	0.2	$1,495	$50	voice: 900 ms
MSAT	ST211 Land Mobile Satphone	US$39.99	US$0.99	$0	0	US$2,300	US$ 50	500 ms [MSAT, 2002]
Orbcomm	Technisonic OSAT-100 SkyTrac	$70 for 5,000 bytes	$0	$0.015	0	US$8,495 + US$800 for antenna	$95	less 60 s to 90 minutes
	Panasonic KX-7101	$0	$0	$0.015	0	US$725	$95	less than 60 s to 90 minutes
	Magellan GSC-100	US$29.95	$0	$0	0	US$850 antenna included	US$ 49.95	5 to 90 minutes
	Stellar ST-2500	US$29.95	$0	$0	0	US$335 + US$45 for antenna	US$ 49.95	5 to 90 minutes
	Quake1500	$70	$0	$0	0	US$1,150	$95	less than 60 s to 90 minutes
Radio modem	Pacific Crest 35 watt	$0	$0	$0	0	US$1,900		10 ms

Iridium and Orbcomm were selected for our research experiment since they represent two different typical types of data service and are most cost effective among all listed systems. The Motorola 9500 for Iridium and Quake1500 for Orbcomm were chosen as test equipment.

3.2.3 High accuracy seismic survey

The first satellite system cost estimation is for a high accuracy seismic survey. The Iridium satellite communication system is used between a helicopter and base station since the Iridium system has low system latency. The communication between base station and head office is carrying project information for which low latency is not a critical requirement. Thus, an Orbcomm system with higher system latency and lower update rate, which is more cost-effective, is adopted in communication between base station and head office.

Assuming the seismic survey field site is in Alberta at 57 degrees north, one helicopter with one satellite transceiver installed is deployed, and two satellite transceivers are installed at the base station, one for helicopter transmission (Iridium) and one for head office communication (Orbcomm). A standard Internet connection is used at head office to communicate with the base station. Figure 3.5 shows the architecture of a high accuracy seismic survey.

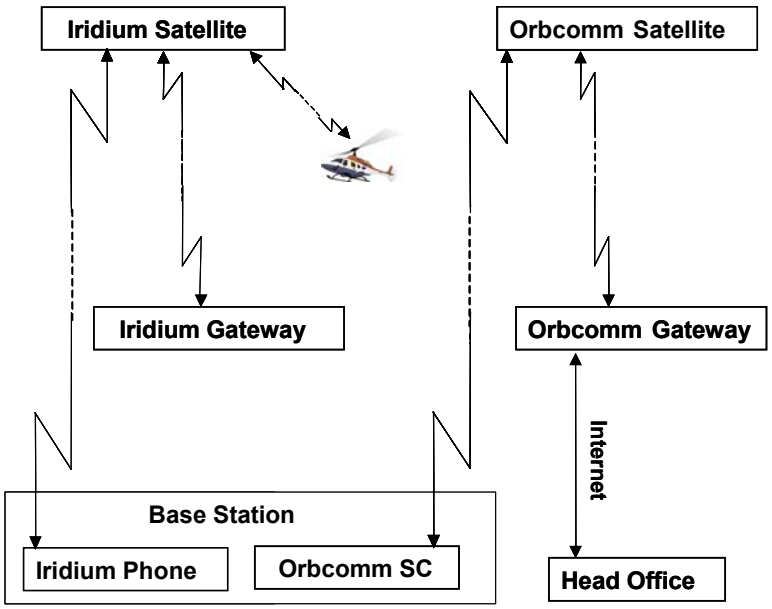


Figure 3.5 High accuracy seismic survey model using satellite communication.

3.2.4 Forest fire operations

In forest fire operations, low latency data transmission is not as necessary. The Orbcomm system is used in the helicopters. Since the necessary Internet access is not available at the base station, helicopters communicate with the base station via head office. In forest fire cost model 1, helicopters send data to head office. Head office then transfers the data to an Iridium ground gateway via an Internet connection. From the Iridium ground gateway there are two choices for data transmission to an Iridium satellite phone at the base station: (a) Mobile Terminated Data (2,400 bps) or (b) Direct Internet Data (10,000 bps). Data rates for both Iridium data services are high enough for transmission of KFP files (assuming each KFP file is 76 bytes and 15 transmissions per minute for each helicopter, a resultant 190 bytes per second assuming 10 helicopters are transmitting simultaneously). Mobile Terminated Data has a lower latency (on the order of 200 ms as it uses a voice circuit) while Direct Internet Data has a higher data rate and higher latency (which could be up to 10 minutes as shown in Table 3.2). The extra bandwidth of the Direct Internet Data connection could be used to carry transmissions besides KFP files (e.g. text E-mail messages).

We assume the operation field site is in Montana at 50 degrees N with 60 second (or less) Orbcomm system latency. We also assume 10 helicopters are deployed in one operation, 5 for the full 10 operational days, 3 for 6 operational days and 2 for 2 operational days. Each helicopter has one Orbcomm satellite transceiver installed. One Motorola 9500 Iridium satellite phone is used at base station. Figure 3.6 shows the communication connections of cost model 1 for forest fire operations.

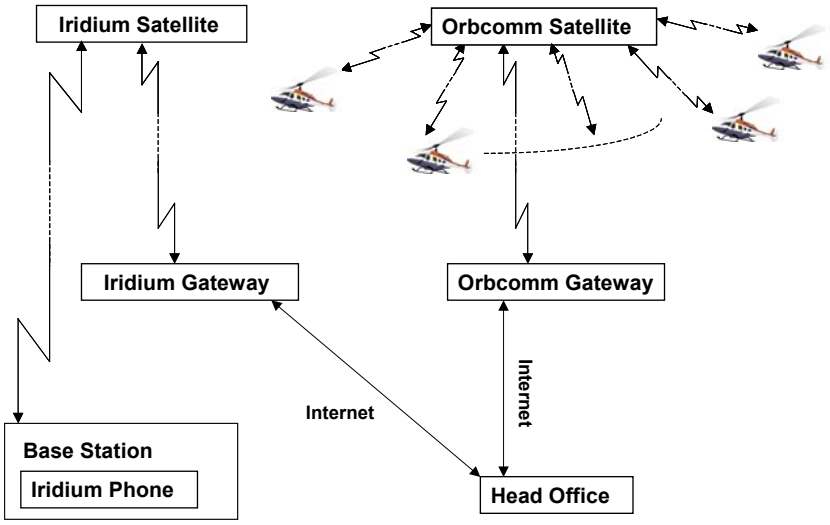


Figure 3.6 Connections for forest fire operations model #1 (Iridium plus Orbcomm).

In the 2nd forest fire operational model, every component is kept the same as model 1 except the connection between head office and base station. An Orbcomm satellite transceiver is used at base station instead of an Iridium satellite phone. After receiving data from helicopters, the head office transfers data to the base station through the Orbcomm system and vice versa. Figure 3.7 shows the communication connection of this forest fire operational model.

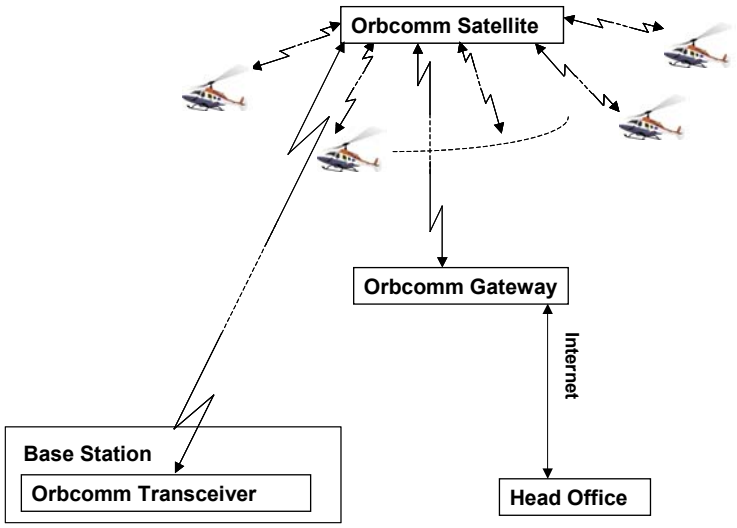


Figure 3.7 Connections for forest fire operations model #2 (all Orbcomm).

3.2.5 Radio modem model

In this satellite system cost model, the currently deployed radio modem [Eagle, 2001] is kept as a link between helicopter and base station, and the Orbcomm satellite system is used between the base station and head office with an assumption that two radio repeaters have been built to relay the radio transmission between the helicopter and the base station. Figure 3.8 is the architecture for the radio model. The purpose of designing a radio modem model is to explore seismic survey system service and cost changes compared with all-satellite communication models.

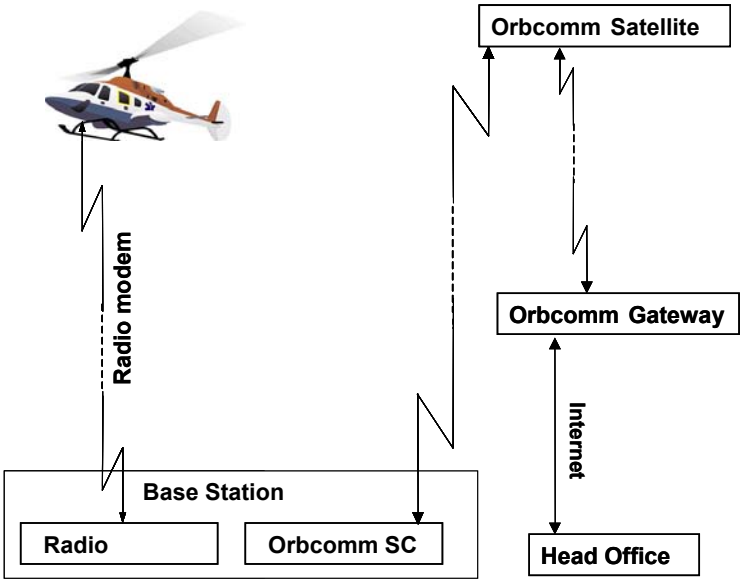


Figure 3.8 Radio model with Orbcomm link.

3.3 XML version

XML is used to define a file format for the satellite system cost estimation application. The application is designed and implemented in C++ with an object-oriented model created using Rational Rose [StPo, 2000].

3.3.1 Cost estimation model

The Unified Modeling Language (UML) is an OMG standard object-oriented design language useful for design of object-oriented software systems. Software engineers use UML to give a clear picture of the system and ensure the system design is useful, usable, reliable, flexible, affordable and available before any implementation and essential costs occur [StPo, 2000]. Figure 3.9 is a UML model for the survey with satellite system cost estimation application.

Our design for satellite system cost estimation is a typical object-oriented design. Class objects of orbcommCost, internetCost and iridiumCost are part of a satCommCost class that represents cost *A* in the satellite system cost calculation equation. Objects heliCost, comSupportCost and installCost are constructed as cost elements *B*, *C* and *D*. These cost element objects can be used in any other classes during calculation of total seismic survey project cost and XML file generation. A fileRead class works as a tool to read basic satellite cost element files and retrieve element values for satellite system cost calculations.

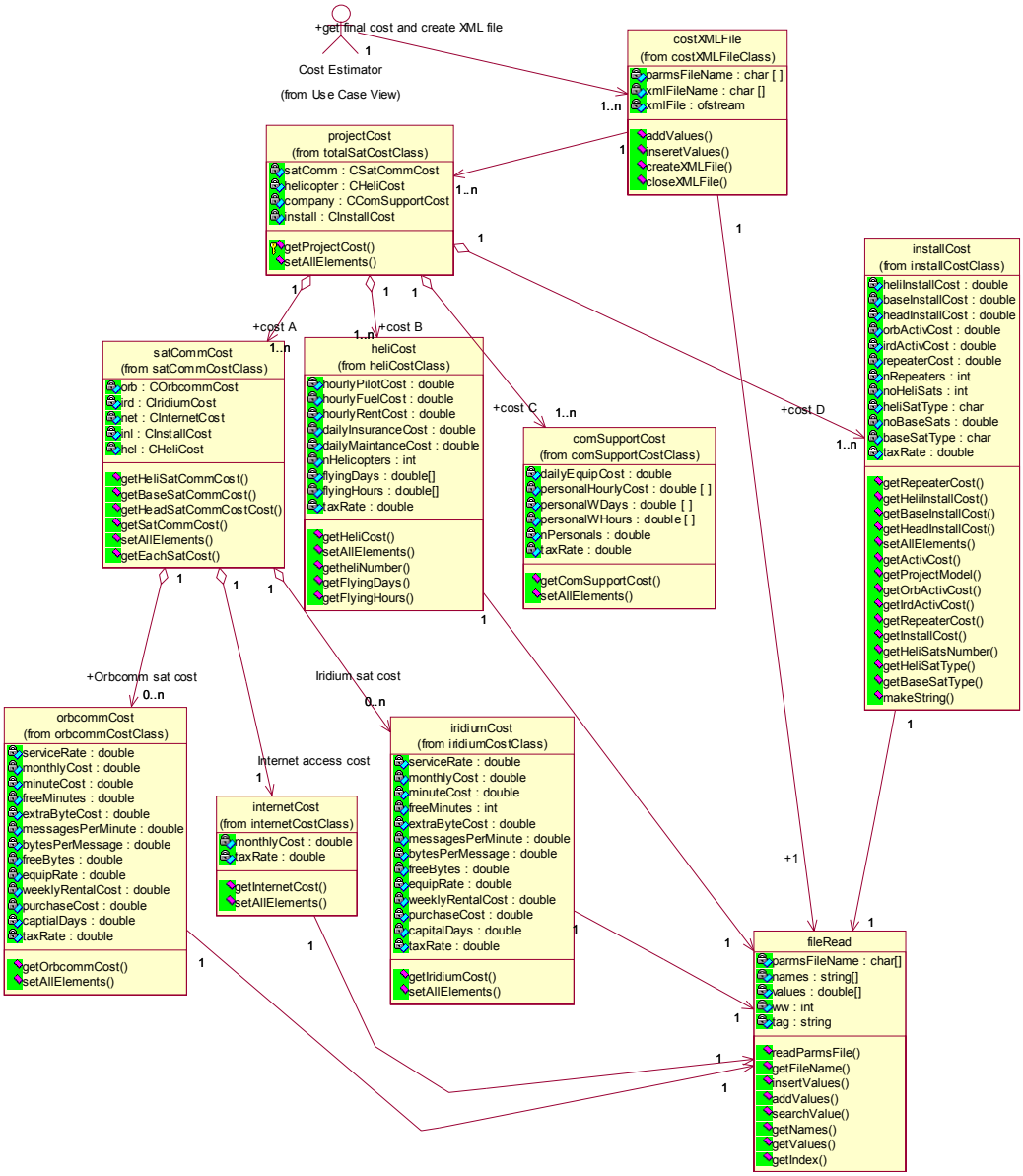


Figure 3.9 The survey with satellite project cost estimation model in UML.

A user only needs a simple main method to call CostXMLFile class methods to obtain a project cost XML file. A CostXMLFile class object is created with two parameters – source satellite cost elements file name and project cost XML file name. The

CostXMLFile object then calls the createXMLFile method to generate a project cost XML file. Figure 3.10 shows pseudocode of a typical main method.

```
Read the parms file name and cost XML file name from
   standard input.
Construct CostXMLFile object with two file names.
Call createXMLFile method to create project XML file.
Close project XML file.
```

Figure 3.10 Pseudo-code of the main method for project cost estimation.

3.3.2 Software implementation

According to the object-oriented model shown above, C++ was used to implement the satellite system cost estimation application.

A project cost XML file ProjectCost.xml provides user satellite system costs for different seismic survey operations in detailed layers. ProjectCost.xml is generated using the satellite cost estimation application CostXML_Generator by reading from both a CostParms.xml file and a ProjectCost.dtd file. Figure 3.11 shows the architecture of the CostXML_Generator program.

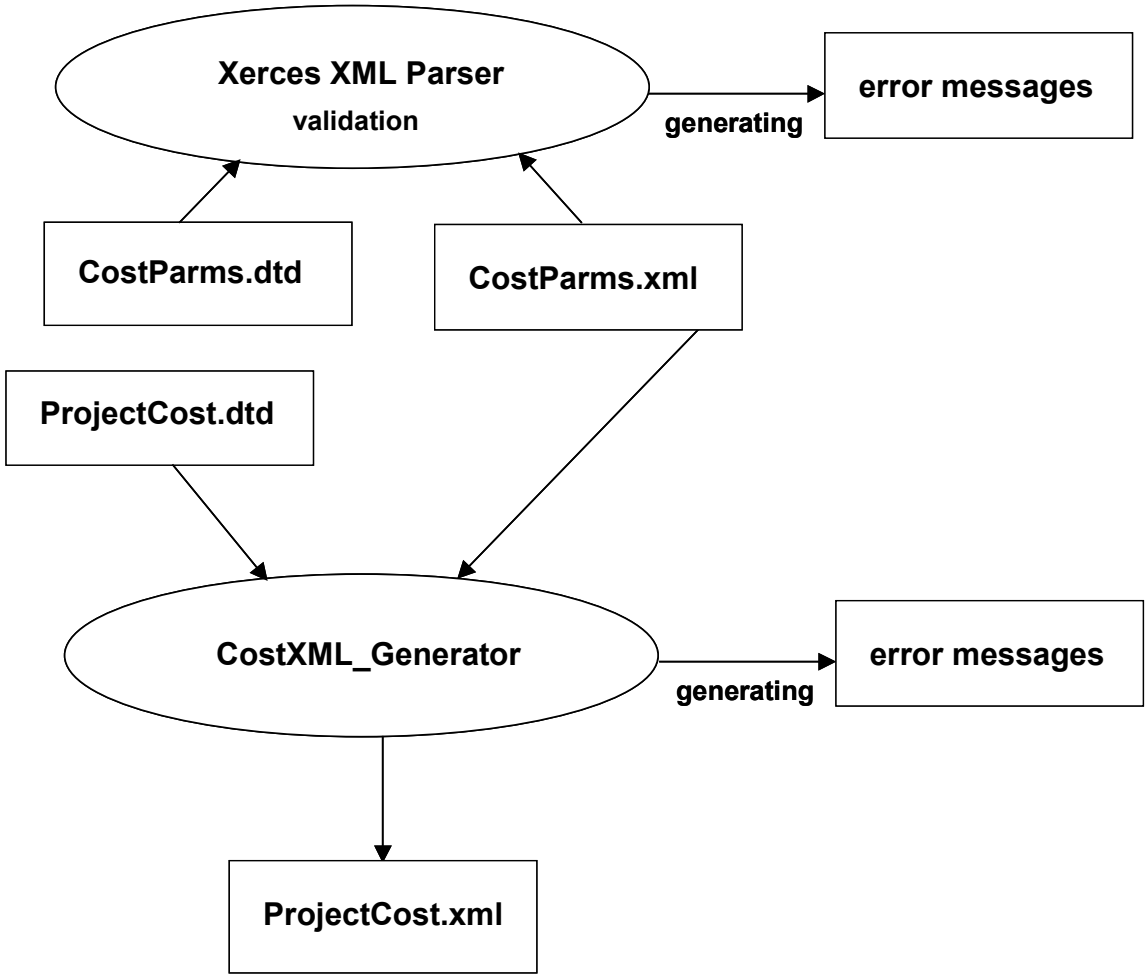


Figure 3.11 Architecture of the CostXML_Generator program.

3.3.2.1 Satellite system basic cost elements source file

An XML format satellite system source file contains basic cost elements of satellite systems required for the system cost calculation equations. The satellite system source file CostParms.xml is generated in XML format according to the grammar defined in its XML DTD file CostParms.dtd. Figure 3.12 shows part of a CostParms.dtd file. The complete CostParms.dtd file is given in Appendix I.

```
!ELEMENT Orbcomm
(service_rate_o,monthly_cost_o,minute_cost_o,free_minutes_o
,cost_per_byte_o,messages_per_minute_o,bytes_per_message_o,
free_bytes_o,equipment_rate_o,weekly_rental_cost_o,equipmen
t_purchase_cost_o,capital_days_o) >
<!ELEMENT service_rate_o (#PCDATA) >
<!ELEMENT monthly_cost_o (#PCDATA) >
<!ELEMENT minute_cost_o (#PCDATA) >
<!ELEMENT free_minutes_o (#PCDATA) >
<!ELEMENT cost_per_byte_o (#PCDATA) >
<!ELEMENT messages_per_minute_o (#PCDATA) >
<!ELEMENT bytes_per_message_o (#PCDATA) >
<!ELEMENT free_bytes_o (#PCDATA) >
<!ELEMENT equipment_rate_o (#PCDATA) >
<!ELEMENT weekly_rental_cost_o (#PCDATA) >
<!ELEMENT equipment_purchase_cost_o (#PCDATA) >
<!ELEMENT capital_days_o (#PCDATA) >
```

Figure 3.12 Part of the CostParms.dtd file (for the Orbcomm cost elements part).

Based on CostParms.dtd, there are seven major parts in the basic satellite cost elements file - CostParms.xml. The seven parts are defined under tag names of "Scenario", "Taxrate", "Orbcomm", "Iridium", "Internet", "Helicopters", "Companysupport" and "Install". Each part includes basic cost elements for the particular part of the satellite system cost model defined in the calculation equations (3.1) to (3.11). Various survey operation's CostParms.xml have different basic cost elements except cost elements in "Orbcomm" and "Iridium". Figure 3.13 shown the "Install" part of a CostParms.xml file for fire operation model #1. According to CostParms.dtd, there are eleven tag values within the "Install" section as follows:

(i) `heli_install_cost` = installation cost of each satellite transceiver in helicopter.

(ii) `base_install_cost` = installation cost of each satellite transceiver at base station.

(iii) `head_install_cost` = head office installation cost.

(iv) `orb_activation_cost` = activation cost of one Orbcomm satellite transceiver.

(v) `ird_activation_cost` = activation cost of one Iridium satellite transceiver.

(vi) `repeater_install_cost` = each repeater installation cost.

(vii) `repeater_number` = number of repeaters.

(viii) `heli_sat_type` = type of satellite transceiver in a helicopter.

(ix) `heli_sat_number` = number of total satellite transceivers in helicopter(s).

(x) `base_sat_type` = type of satellite transceiver at the base station.

(xi) `base_sat_number` = total number of satellite transceivers at the base station.

```
<Install>
  <heli_install_cost>2500</heli_install_cost>
  <base_install_cost>5000</base_install_cost>
  <head_install_cost>2000</head_install_cost>
  <orb_activation_cost>50</orb_activation_cost>
  <ird_activation_cost>24</ird_activation_cost>
  <repeater_install_cost>1600</repeater_install_cost>
  <repeater_number>0</repeater_number>
  <heli_sat_number>10</heli_sat_number>
  <heli_sat_type>0</heli_sat_type>
  <base_sat_number>1</base_sat_number>
  <base_sat_type>1</base_sat_type>
 </Install>
```

Figure 3.13 Part of an example CostParms.xml file for forest fire operation #1.

In the forest fire operation CostParms.xml file, “0” represents an Orbcomm satellite transceiver, “1” indicates an Iridium satellite transceiver and “2” means a combination of “Orbcomm” and “Iridium” is installed at the base station or in a helicopter. In forest fire operation model #1, ten helicopters are deployed in the project, each with one Orbcomm transceiver (“0”). One Iridium phone (“1”) is working at the base station. These satellite system basic cost elements provide information to the system cost model software in installation cost (cost *D*) calculations. Some basic cost elements in the “Install” part such

as type of satellite transceiver and number of satellite transceivers are also necessary to calculate other parts of the project cost.

3.3.2.2 Project cost XML file

ProjectCost.dtd defines the structure of a project cost XML file. Information from the CostParms.xml file (validated by the Xerces XML Parser with CostParms.dtd) gives the values of all elements needed by the calculation equations. The CostXML_Generator determines the seismic survey scenario and conducts different calculations based on basic cost information. Figure 3.14 shows the "Install" part of a ProjectCost.dtd file and Figure 3.15 shows the "Install" part of a ProjectCost.xml file generated according to CostXML.dtd for a high accuracy seismic survey. The complete ProjectCost.dtd file is given in Appendix II.

During the ProjectCost.xml file generation, each tag defined in ProjectCost.dtd is printed in the ProjectCost.xml according to the basic cost element value and type of seismic survey operation. After calculation of each installation cost, D_1, D_2, D_3 are displayed in the project cost file with detailed tag values. Total installation cost and its percentage of total project cost is shown under the tag name of "Install_cost".

```
<!ELEMENT Install_Cost
(Heli_Install_Cost,Base_Install_Cost,Head_Install_Cost,Repeater_Ins
tall_Cost?) >
<!ATTLIST Install_Cost cost_D CDATA #REQUIRED percentage_D CDATA
#REQUIRED >
<!ELEMENT Heli_Install_Cost (Heli_Install_Details) >
<!ATTLIST Heli_Install_Cost cost_D1 CDATA #REQUIRED >
<!ELEMENT Heli_Install_Details
(heli_install_cost,heli_sat_activation_cost?) >
<!ATTLIST Heli_Install_Details heli_sat_type (0|1|2|3|9999) "0"
heli_sat_number (0|1|10) "0" >
<!ELEMENT heli_install_cost (#PCDATA) >
<!ELEMENT heli_sat_activation_cost (#PCDATA) >
<!ELEMENT Base_Install_Cost (Base_Install_Details) >
<!ATTLIST Base_Install_Cost cost_D2 CDATA #REQUIRED >
<!ELEMENT Base_Install_Details
(base_install_cost+,base_sat_activation_cost+) >
<!ATTLIST Base_Install_Details base_sat_type (0|1|2|3|9999) "0"
base_sat_number (0|1|2) "0" >
<!ELEMENT base_install_cost (#PCDATA) >
<!ELEMENT base_sat_activation_cost (#PCDATA) >
<!ELEMENT Head_Install_Cost (Head_Install_Details) >
<!ATTLIST Head_Install_Cost cost_D3 CDATA #REQUIRED >
<!ELEMENT Head_Install_Details (head_install_cost) >
<!ATTLIST Head_Install_Details head_sat_type CDATA #REQUIRED >
<!ELEMENT head_install_cost (#PCDATA) >
<!ELEMENT Repeater_Install_Cost (Repeater_Install_Details) >
<!ATTLIST Repeater_Install_Cost cost_D4 CDATA #REQUIRED >
<!ELEMENT Repeater_Install_Details (repeater_install_cost) >
<!ATTLIST Repeater_Install_Details repeater_number (0|1|2|3) "0" >
<!ELEMENT repeater_install_cost (#PCDATA) >
```

Figure 3.14 CostXML.dtd for high accuracy seismic survey (`Install` part only).

```
<Install_Cost cost_D="15619.9" percentage_D="6.76251">
  <Heli_Install_Cost cost_D1="2700.68">
    <Heli_Install_Details heli_sat_type="1" heli_sat_number="1">
      <heli_install_cost>2500</heli_install_cost>
      <heli_sat_activation_cost>24</heli_sat_activation_cost>
    </Heli_Install_Details>
  </Heli_Install_Cost>

  <Base_Install_Cost cost_D2="10779.2">
    <Base_Install_Details base_sat_type="2" base_sat_number="2">
      <base_install_cost>5000</base_install_cost>
      <base_sat_activation_cost>50</base_sat_activation_cost>
      <base_sat_activation_cost>24</base_sat_activation_cost>
    </Base_Install_Details>
  </Base_Install_Cost>

  <Head_Install_Cost cost_D3="2140">
    <Head_Install_Details head_sat_type="0">
      <head_install_cost>2000</head_install_cost>
    </Head_Install_Details>
  </Head_Install_Cost>

  <Repeater_Install_Cost cost_D4="0">
    <Repeater_Install_Details repeater_number="0">
      <repeater_install_cost>1600</repeater_install_cost>
    </Repeater_Install_Details>
  </Repeater_Install_Cost>
</Install_Cost>
```

Figure 3.15 Part of a sample ProjectCost.xml file for a high accuracy seismic survey (`Install_Cost` only).

3.3.2.3 Cost XML File Generator

CostXML_Generator completes the project costs (using equations (3.1) to (3.11)) and shows the results in an XML format ProjectCost.xml file. CostXML_Generator is coded in C++ based on the object-oriented design model given in section 3.3.1. The fileRead class is used in each level of satellite system cost calculation class including orbcommCost, iridiumCost, helicopterCost, comSupportCost and projectCost in reading CostParms.xml for basic cost element values. The fileRead program reads the satellite basic cost elements source file name (e.g. CostParms.xml) and a keyword that indicates which particular section of the satellite basic cost elements source file will be read in the file access. For instance, if the keyword input is “Orbcomm”, the `readParmsFile` method in the fileRead class skips other parts of the CostParms.xml file, only reads and returns the tag values within “`Orbcomm”` section. Information provided to

CostXML_Generator to decide the type and number of satellite transceivers, total number of helicopters and total number of repeaters is stored in the "`Install`" section of the CostParms.xml file. At top of the CostParms.xml file, there is a "`Scenario`" tag which CostXML_Generator reads to determine the type of survey operation (0 for high accuracy seismic survey, 1 for forest fire #1, 2 for forest fire #2 or 3 for seismic survey with radio modem).

As shown in Figure 3.13, CostXML_Generator reads from the "`Install`" section and determines that there are 10 helicopters in the operation, each with one Orbcomm satellite transceiver. There is one Iridium satellite transceiver installed at the base station. The tag value from "`Install`" also tells CostXML_Generator that no repeater is involved in the operation. After reading the tag value of "`Scenario`", CostXML_Generator concludes the operation is fire operation #1 and chooses the calculation equations accordingly. Figure 3.16 is part of the code from CostXML_Generator to calculate installation cost D_1 of satellite transceivers in one helicopter.

```
double CInstallCost::getHeliInstallCost(){
 double activCost=0;
 double heliCost=0;
 if(heliSatType == 0)
   activCost = orbActivCost;
 else if(heliSatType ==1)
   activCost = irdActivCost;
 else if(heliSatType==3)
   activCost=0;

     heliCost=(1.0+taxRate) * noHeliSats *
     (heliInstallCost + activCost);

 if(nRepeaters!=0)
     heliCost=heliCost+heliInstallCost*(1.0+taxRate);
 return heliCost;
}
```

Figure 3.16 Example of source code from CostXML_Generator to compute cost element D_1 from equation (3.9).

A key part of the CostXML_Generator is the CostXMLFile class. The function of this class is to generate a ProjectCost.xml file according to data type definitions in the ProjectCost.dtd file. By reading each line of the ProjectCost.dtd file, CostXML_Generator determines the tag names and data type defined for each tag in the XML tree structure of the ProjectCost.xml file. CostXML_Generator must determine if the tag just read is a parent element name, an attribute name or a child element name. CostXML_Generator also needs to know how many children each parent tag has from reading each line of the ProjectCost.dtd file. CostXML_Generator then searches satellite system cost results (calculated and stored in an array in an early part of the program) to find the value of the tag and write it into the XML file at its appropriate position. Figure 3.17 is a pseudo-code description of the `createXMLFile()` method of the `CostXMLFile` class.

```
Open CostParms.xml file and read tag values.
Calculate each part of survey with satellite system cost
    and store them in an array for later use.
Open project ProjectCost.dtd and ProjectCost.xml files.
While not at the end of ProjectCost.dtd file, keep reading
    each line of the dtd file.
    Check each token read from a single line to see if it
        is parent or children. If it is parent, store its
        name and number of its children in arrays for later
        XML file generation.
    For each token of a single line, find its value in the
        calculation results array.
        Use information stored in the parent arrays to
            determine the position of the token (tag) in
            the XML structure, adjust the tab value and
            print token name and value to the
            ProjectCost.xml file.
```

Figure 3.17 Pseudo-code description of the `createXMLFile()` method (`CostXMLFile` class).

CostXML_Generator issues an error message when tag names in a CostParms.xml file do not match tag names in the ProjectCost.dtd file. An error message tells the user the location of the error in the CostParms.xml file and the correct tag name. CostXML_Generator also provides the user flexibility by allowing a certain degree of modifications in both the CostParms.xml and ProjectCost.dtd files. Changing tag names, adding a new tag or deleting a tag in ProjectCost.dtd will require appropriate changes in CostParms.xml. Simple modifications such as these are read by the CostXML_Generator and reflected in the ProjectCost.xml file generated by the CostXML_Generator without changing and recompiling its source code.

3.3.3 Example satellite survey costs

Based on the cost estimation model equations (section 3.2.1), we calculated the example costs for different types of 10-day surveys introduced in sections 3.2.3 to 3.2.5. Table 3.3 shows the comparison of survey costs using various satellite systems. As we

can see, the Orbcomm satellite system is less expensive. In fact, cost component *A* (satellite system cost) is about 10 times less for Orbcomm compared to Iridium.

Table 3.3 Costs comparisons of different satellite survey (number in ( ) after each subtotal indicating % of total cost excluding cost *B*).

Cost element	**Description**		**High accuracy seismic survey**		**Forest fire operations**	
			satellite	radio modem	**#1**	**#2**
A_1 *	Helicopter sat cost	Orbcomm	0	0	10	10
		Iridium	1	0	0	0
		radio modem	0	1	0	0
		cost	$4,259.37	$0	$3,416.02	$3,416.02
A_2 *	base station sat cost	Orbcomm	1	1	0	1
		Iridium	1	0	1	0
		radio modem	0	1	0	0
		cost	$4,733.28	$474.45	$4,259.37	$474.45
A_3	head office sat cost	Internet service	1	1	1	1
		cost	$32.10	$32.10	$32.10	$32.10
A	total sat cost		**$9,025** (19)	**$506** (1)	**$7,707** (12)	**$3,922** (7)
B	Flying helicopter cost	helicopter (s)	1	1	10	10
		helicopter type	Bell 205	Bell 205	Bell 206B	Bell 206B
		flying days	10	10	**	**
		daily fly hours	6	6	6	6
		cost	**$182,970**	**$182,970**	**$420,176**	**$420,176**
C	Company support cost	personnel	2	2	2	2
		daily hours	8	8	8	8
		cost	**$23,361** (48)	**$23,361** (55)	**$23,361** (35)	**$23,361** (38)
D_1	Helicopter installation cost	Orbcomm	0	0	10	10
		Iridium	1	0	0	0
		radio modem	0	1	0	0
		cost	$2,700.68	$2,675.00	$27,285.00	$27,285.00
D_2	base station installation cost	Orbcomm	1	1	0	1
		Iridium	1	0	1	0
		radio modem	0	1	0	0
		cost	$10,779.18	$10,754	$5,375.68	$5,403.50
D_3	head office install cost	Internet service	1	1	1	1
		cost	$2,140.00	$2,140.00	$2,140.00	$2,140.00
WU	Repeater install cost	repeater (s)	0	2	0	0
		Cost	0	$3,424.00	0	0
D	**Total install cost**		**$15,619** (33)	**$18,992** (44)	**$34,800** (53)	**$34,828** (55)
A+C+D	**Subtotal**		**$48,005**	**$42,859**	**$65,868**	**$62,111**
S	**Total**		**$230,977**	**$225,831**	**$486,046**	**$482,289**

* Costs A_1 and A_2 include capital costs of equipment purchase, monthly fee and airtime minutes listed in Table 3.1 for Iridium and Orbcomm systems.

** 5 helicopters fly 10 full days, 3 helicopters fly 6 days and 2 helicopters fly 2 days.

The cost for each survey varies because different satellite system configurations have been used in each operation. In the high accuracy seismic survey with satellite system, one Iridium satellite transceiver is used in the helicopter and one Iridium transceiver plus one Orbcomm transceiver are installed at the base station. In the radio modem high accuracy seismic survey, we replace the Iridium transceivers in both helicopter and base station with radio modems and the total survey cost is reduced. Forest fire operation #1 differs from #2 in that, at the base station, one Iridium transceiver is installed in model #1 and one Orbcomm transceiver is used in model #2. The total survey cost including the satellite cost subtotal of model #1 is higher than model #2 because the satellite system cost (*A*) for Iridium is higher than the cost for Orbcomm. A complete set of ProjectCost.xml files generated by Cost_XML_Generator for the examples in Table 3.3 is given in Appendix III.

In the Orbcomm system communication cost, there is an extra per-byte cost for the transmitted data when the data bytes are more than the free bytes included in the monthly fee. The price used in the calculation is offered by Rom Communications Inc. (Kelowna, BC). For every extra 25,000 data bytes, the cost is $91.95 ($0.003678 per byte). This cost did not apply to our research experiments with the Orbcomm system because of the special deal offered by SkyTrac Systems Ltd. (Penticton, BC).

3.3.4 SOAP compatibility

As introduced in section 2.4.4, SOAP works as a communication protocol between a SOAP server and client. A SOAP server provides a simple way to build web services. Web services include creating COM objects and generating a WSDL file. A SOAP server has functions to retrieve requests from HTTP and process the parameters of the requests.

A SOAP server then parses the WSDL file, constructs input parameters for SOAP response, invokes specified SOAP methods and sends a response back to the client through HTTP [W3C, 2002] [SQLData, 2002].

SQLData3.0 SOAP Server was used in this research to test how a SOAP server communicates with a client for information exchange over HTTP. SQLData3.0 is a SOAP server package from SQLData Systems, Inc. SQLData SOAP server is implemented in C++ with high performance and low footprint. It is an efficient and reliable bottom-up implementation of the SOAP 1.1 standard with build-in HTTP/HTTPS support. The server has a flexible WSDL parser, a SOAP Actor to fulfill SOAP requests, a dynamic SOAP client and a smart XML Cache Manager. The SQLData SOAP server can perform as a SOAP client or intermediary. Services from one SOAP server can be sent to another SOAP server for processing that is transparent to the SOAP server acting as the client. SQLData SOAP server also converts database service requests from a client such as ODBC to automatically access SOAP server services without any additional programming [SQLData, 2002]. Figure 3.18 shows an example SOAP response message from a SQLData3.0 SOAP server to a client.

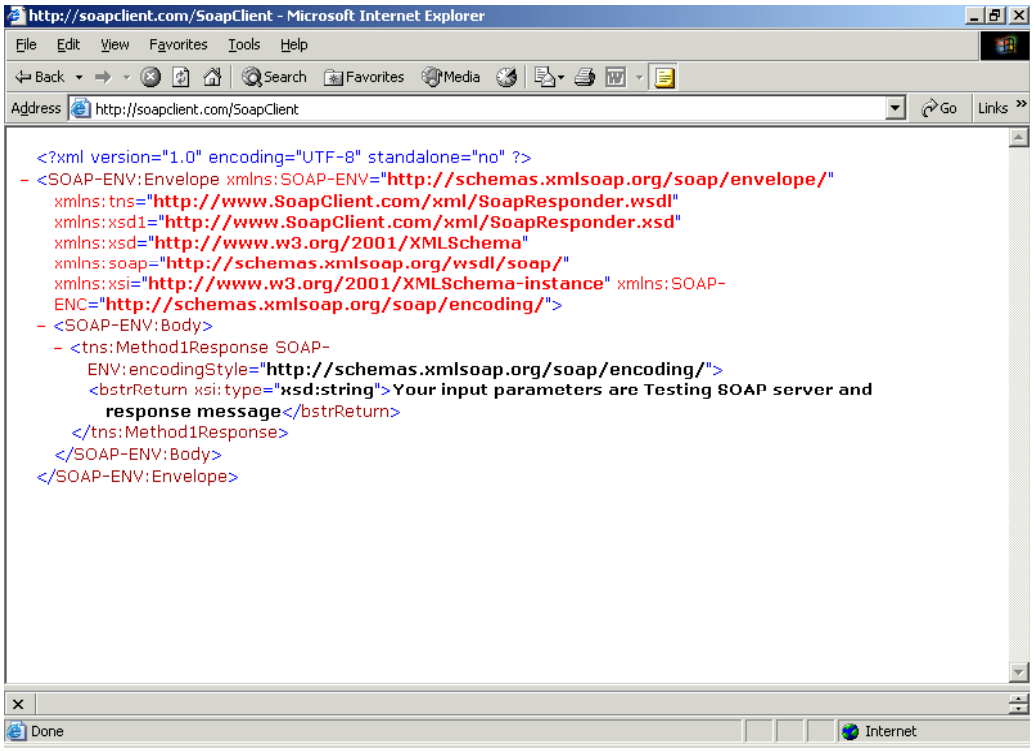


Figure 3.18 Example SOAP response from SQLData3.0 SOAP Server [SQLData, 2002].

Our experiments with SQLData3.0 SOAP Server have shown SOAP features that can be an asset to the satellite cost model application. A satellite seismic services cost SOAP server (which we call the SatSOAP server) containing the basic cost elements of satellite systems (CostParms.xml) could be established. Customers who are interested in satellite services for seismic survey operations can send SOAP requests through HTTP in any web browser to the SatSOAP server to obtain satellite service costs. A SOAP client request would contain parameter values such as type of seismic survey and length of the project indicating a customer's requirements for a project using satellite communications. Methods defined in the SatSOAP server will process the SOAP request, calculate satellite project cost for the specific operation and respond to the customer with satellite system cost in a detailed XML format. Figure 3.19 shows the proposed architecture of a SOAP

enabled project cost estimation application. Figure 3.20 shows a sample SOAP response file including information of the head office satellite communication cost.

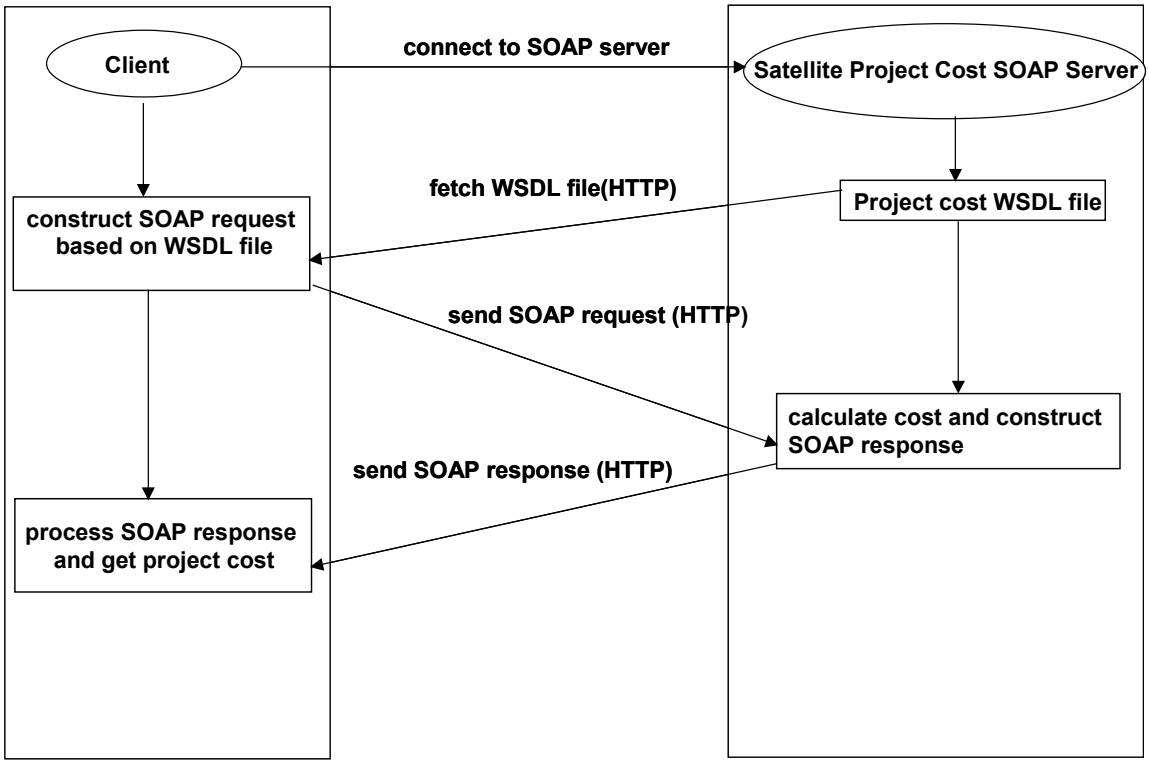


Figure 3.19 Proposed architecture of a SOAP-enabled project cost estimation application.

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<SOAP-ENV:Envelope xmlns:SOAP-
ENV=http://schemas.xmlsoap.org/soap/envelope/
   xmlns:tns=http://www.SatProject.com/xml/SatProjectCo
   st.wsdl
   xmlns:xsd1="http://www.SatProject.com/xml/SatProject
   Cost.xsd"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-
   instance" xmlns:SOAP-
   ENC="http://schemas.xmlsoap.org/soap/encoding/">
   <SOAP-ENV:Body>
         <m:HeadSatCostResponse  xmlns:
         m="http://www.SatProject.com/SatProjectCost/">
               <m:Head_Sat_Cost _A3> 32.1
               </m:Head_Sat_Cost _A3>
               <m:Head-Sat_Details>Orbcomm</m:Head-
               Sat_Details>
               <m: internet_monthly_cost>30
               </m:internet_monthly_cost>
         </m:HeadSatCostResponse>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 3.20 Example SOAP response of satellite communication cost at head office.

Chapter 4. Satellite communication experiments

4.1 Experiment design

Satellite link communication experiments were designed and implemented in this research on both Iridium and Orbcomm systems to test system latency on data transmission. The simulated satellite testing system is built with one Windows 2000 platform (base station) and one QNX 4.25 platform (helicopter) communicating through a satellite link.

4.1.1 System architecture

The architecture of our satellite communication testing system is shown in Figure 4.1. Our satellite testing system is based on previous research [Shan, 2001] and focuses on real-time communication using satellite data transceivers (modem) and the software components (Sender and Receiver) necessary to run the transceivers under computer control. In our research, satellite link software components were written for Windows 2000 and QNX 4.25. We used an Iridium Motorola 9500 phone and a Quake 1500 Orbcomm modem as satellite transceivers in our testing.

Our research focused on software development for satellite system data communication latency testing. Elements affecting satellite data communication performance include transferred data size, transmission frequency, transmission time period and transmission environment (static or dynamic). We investigated all of these variables in our experiments.

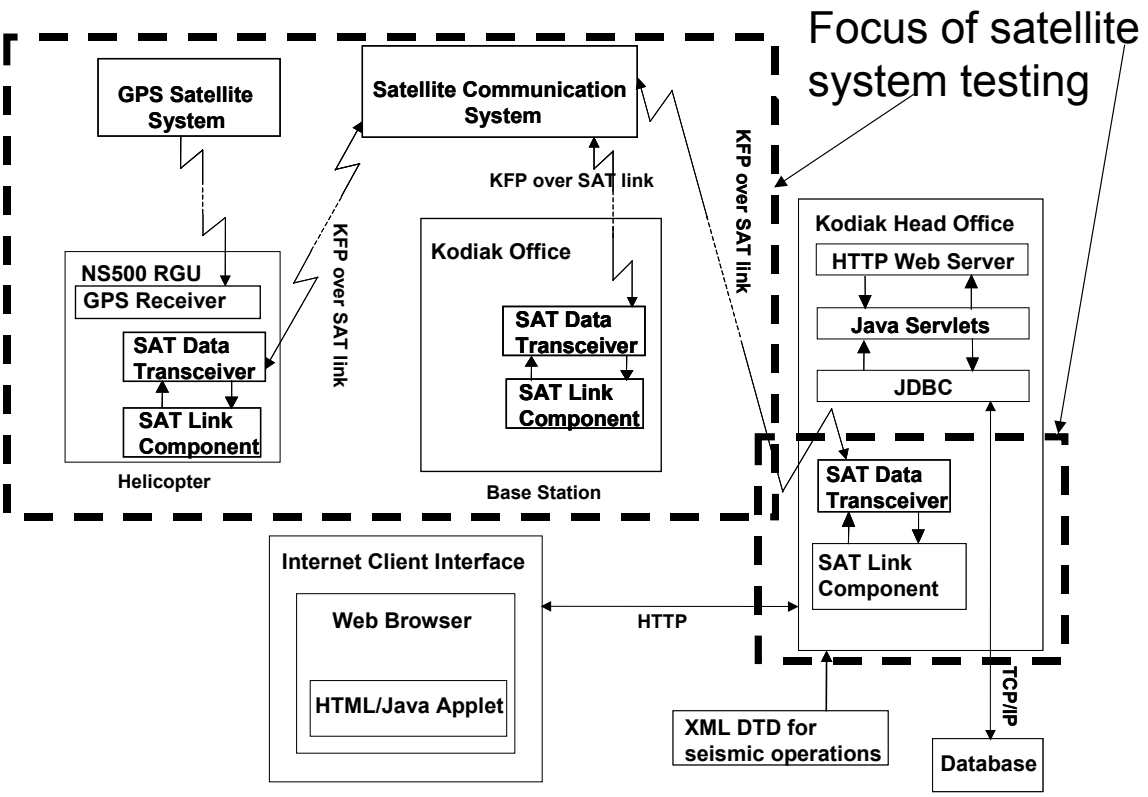


Figure 4.1 Architecture of satellite communication experiment system.

4.1.2 QNX 4.25

The QNX 4.25 real-time operating system was installed on a Pentium 166 IBM workstation with 31 MB extended system memory and 2.56 GB hard disk (QNX1) acting as if it was installed in a helicopter. Version 4 of QNX provides a reliable and scalable operating system for embedded real-time systems. QNX is a true microkernel operating system with simplicity and efficiency. The path through the operating system of an application is very short with only a few system instructions. The QNX system can work for ROMable embedded applications since the system is very small. It also has the power to run a distributed network with several hundred processors. With these special features, the QNX system uses only a very small part of a processor's energy on the operating system. Most of the processor's focus is on running the application [QNX, 2002].

As a member of the QNX family, QNX 4.25 is a real-time system with key characteristics such as multitasking, priority-driven preemptive scheduling and fast context switching. QNX 4.25 is a small, simple and efficient operating system with modularity. Two fundamental principles of QNX 4.25 are its microkernel architecture and message-based interprocess communication. The user is also allowed to modify the QNX 4.25 system according to their own criteria [QNX, 1996].

4.1.3 Embedded serial communication programming

An embedded system is designed to execute a special function. It contains computer hardware and software. In some special cases an embedded system can also include additional sections such as mechanical parts [Barr, 1999]. Embedded system programming involves close interaction between software and hardware.

Embedded serial communication programming is an important technology in this research for satellite system latency testing. Successful data communication between the computer serial port and externally connected satellite transceiver is essential for this research. In serial port communication, data is transferred one bit at a time through a computer serial port [Sweet, 1999]. The serial port communicates with external devices such as a keyboard, network devices or satellite transceiver (modem). A serial port has variables of baud rate, data bits, stop bit, parity and flow control, which determine the behavior of a serial port. Baud rate is the most important element that controls the transmission rate of a serial port. In this research, the communication between satellite data transceiver (modem) and satellite link component (computer) are accomplished through a serial port on both the Windows 2000 and QNX 4.25 workstations. The serial port setup and communications on these two different operating systems are very similar.

The first step of serial port communication is to open and configure the serial port. A basic configuration requires setup of baud rate, data bits, parity, stop bit and flow control. In order to communicate with external satellite data transceivers using modem commands, serial port input is set as raw input. With the raw input option, a serial port passes input characters exactly as they are received without additional processing [Sweet, 1999].

Modem commands are sent through a serial port to the external satellite transceiver (modem) to accomplish satellite transceiver configuration and to establish a satellite data link between the sending side and the receiving side.

After configuring both serial ports and the satellite transceiver, the satellite system is ready for data communication by writing data to the output serial port and reading data from the input serial port. Figure 4.2 shows the architecture of the satellite serial communications software.

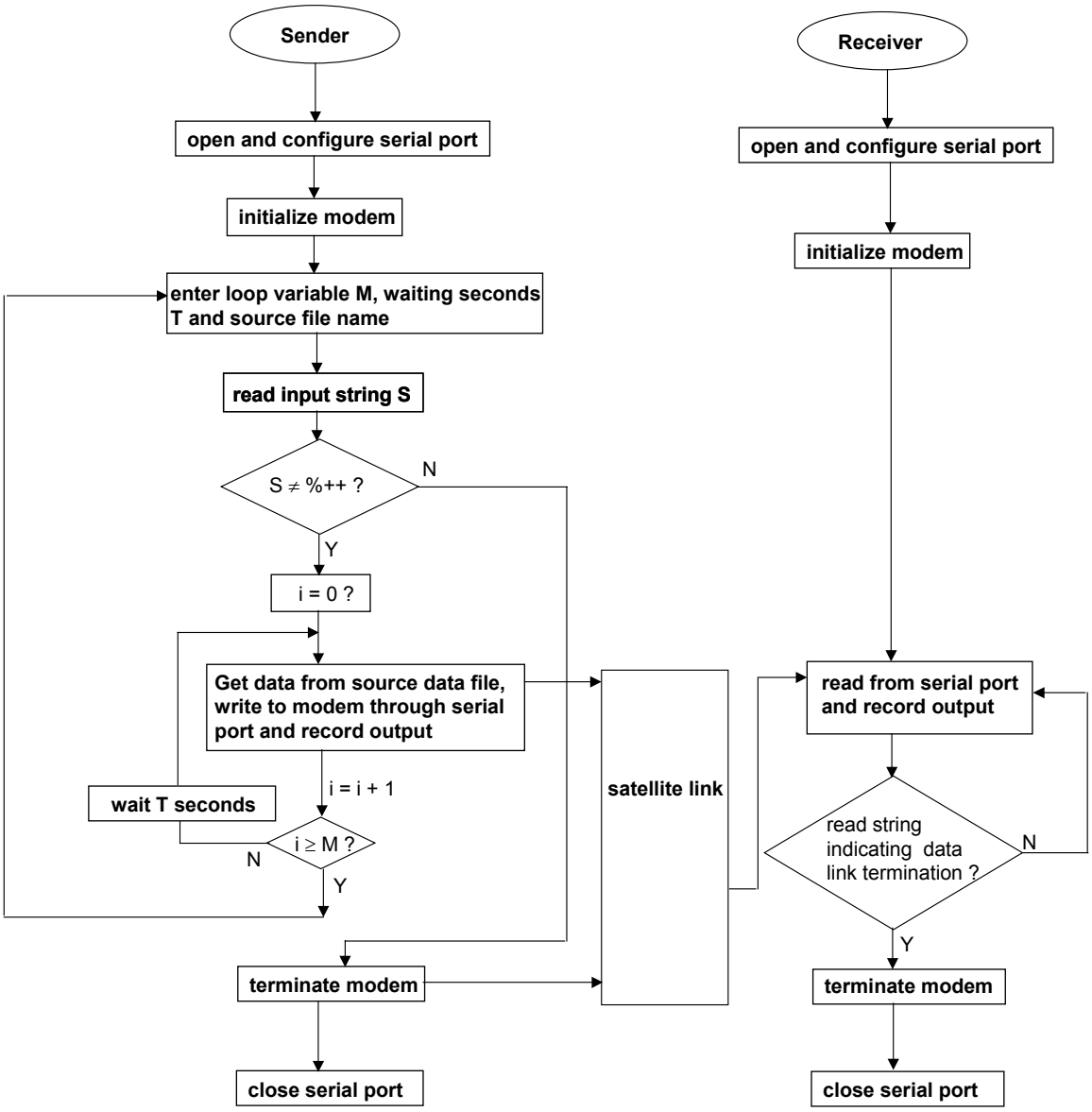


Figure 4.2 Software architecture of the satellite serial communication experiments.

4.1.4 Satellite transceiver (modem)

The satellite transceivers used in this research act as external modems connected to the serial port of sending and receiving computers. Modem commands are sets of unique data strings to configure and initialize a satellite modem to set up the data transmission link. Termination of a satellite data link is also done by modem commands. Different satellite systems are designed to work with unique modem commands.

The Iridium system uses traditional Hayes AT commands. An Iridium satellite transceiver unit is designed to be Hayes AT command compatible. The unit switches between command mode and data mode by responding to Hayes AT commands. Basic commands used to communicate with an Iridium satellite transceiver (modem) are from Hayes AT commands with slight modification by Iridium and Motorola (see Table 4.1). Figure 4.3 shows sample AT commands and responses from the sending side to configure an Iridium Motorola 9500 phone, set the data link and send test data.

The escape sequence "+++" is used to initiate command mode for disconnecting the data link. As shown in Figure 4.2, when the user input string is "%++", the Sender program stops sending data and sends "+++" to the Iridium modem for data link disconnection. A guard time period (usually one second) where no data is transmitted before the "+++" is received and following the "+++" ensures that transmitted data does not inadvertently place the modem into command mode.

The Orbcomm satellite transceiver (modem) is not Hayes AT command compatible and it communicates with its own command language embedded in Orbcomm data strings. The procedure for modem command communication is different for the Iridium and Orbcomm systems. The configuration and initialization of an Iridium satellite transceiver (modem) needs to be done before any serial port data sending or receiving can occur. The Orbcomm system transceiver coded using the Orbcomm Serial Interface accepts one small packet of data which includes modem commands, transferred data and all necessary information for data transmission. Orbcomm packet data is composed in hexidecimal format. Table 4.2 shows the SC-Originated (Subscriber Communicator

Originated) message packet data structure of the Orbcomm system with a small example message.

Table 4.1 Common Iridium AT commands and responses [MPCS, 2000].

AT command (ending with '\r'=carriage return)	Description	Response	Description
AT	attention	OK	executed
AT+CBST= <speed><name><ce>	configure for bearer service (send)		
ATS0=N	configure to answer incoming call automatically	ERROR	rejected
+++	Initiate disconnect		
ATH	disconnect call		
P0	Turn off phone		
ATDT phone number	Dial phone number	OK	executed
		ERROR	rejected
		CONNECT	connection established
		NO CARRIER	connection terminated
		BUSY	busy signal detected
		NO ANSWER	no answer
		RING	receiving call

```
AT
ok
AT+CBST=6,0,1    //set bearer service type values
ok
ATDT 00881631416987  //dial satellite phone number
connect 19200
Testing data   //send testing data
+++                 //send escape sequence to transfer
                    to command mode
ok
ATH        //disconnect data link
ok
```

Figure 4.3 Example AT command and response sequence for the Iridium Motorola 9500 (sending side) [Iridium, 2002c].

Table 4.2 SC-Originated message of Orbcomm with an example packet data from the Quake 1500 satellite transceiver [KBMW, 1999].

byte No.	hex value	Description
0	85	packet header byte (0x85 or 0x86)
1	06	packet data type, 0x06
2	23	length byte 0
3	00	length byte 1
4	00	number of times this packet has been re-sent/sequence number
5	01	Orbcomm gateway id (01 - gateway in USA)
6	00	polled by Orbcomm gateway (01) or initiated by SC (00)
7	02	ACK level (02-delivery to gateway ACK)
8	01	priority level (01 – normal)
9	00	message body type (00 – text)
10	00	DTE assigned to identify among multiple messages
11	01	number of recipients
12	00	subject indicator (00 – no subject)
13	01	pre-defined Orbcomm recipient (01)
14	05	message data type (05 – text)
15 to 32	54 68 69 73 20 69 73 20 61 6E 20 65 78 61 6D 70 6C 65	message body (hex ASCII value for message text “This is an example”)
33	4B	Fletcher Checksum byte 0
34	4D	Fletcher Checksum byte 1

4.2 Software development

Software development for satellite data transmission testing was carried out using C and C++. In the Windows 2000 environment we used the Visual C++ 6.0 compiler, and on QNX 4.25 we used the Watcom Systems version 10.6 C complier.

Sender and Receiver are the two major components in the system test software. Sender is responsible for opening a data file, fetching data from the data file and sending the data through a serial port to the satellite link. Receiver listens to the serial port and handles the incoming data stream. Data stream handling involves checking for the end of each data transmission, checking data errors, printing data to the monitor and writing data

to an output record file. Both Sender and Receiver have common functions of modem configuration, initialization and termination. Accurate system time recording for both Sender and Receiver is important to measure the latency between the time a message is sent and the time it is received. The basic design and structure for both Sender and Receiver is the same for both Windows 2000 and QNX 4.25. Detailed coding is slightly different for Windows 2000 and QNX 4.25.

4.2.1 Serial library

Serial port programming interacts with the serial ports of a computer system. The QNX 4.25 operating system is IEEE Portable Operating System Interface (POSIX) compliant, which provides a POSIX terminal user interface for configuration and communication of a serial port [Sweet, 1999]. This feature of POSIX terminal I/O makes it relatively straightforward to develop software using QNX 4.25 for serial port communication. Other operating systems such as Windows 2000 require the developer to use the Win32 API, and serial port communication software development is more challenging.

A serial library written by Ramon de Klein [Klein, 2000] provides basic serial port operational functions and simplifies serial communications programming for a developer on these non-POSIX operating systems. De Klein's serial library is written in C++, and provides the object classes and methods shown in Table 4.3. An object of CSerial type is instantiated, which is then used to call the methods to open a serial port, set a serial port, write to a serial port and read from a serial port.

Table 4.3 Classes and methods in de Klein's serial library (from [Klein 2000]).
(An * to the left indicates that we used this method for our research.)

	Methods ()	Description
*	CheckPort	Check if serial port is already open
*	Open	Open serial port
*	Setup	Configure serial port parameters
	GetRLSD	
	GetRing	
*	WaitEvent	Wait for serial event (receiving)
*	SetupHandshaking	Enable hand shaking function
*	SetupreadTimeouts	Set time out value for reading
	GetBaudRate	Get baud rate value
	GetDataBits	Get data bits value
	GetParity	Get parity value
	GetStopBits	Get stop bits value
	GetEventMask	Get vent mask value
	GetEventChar	Get event character value
	GetHandshaking	Get handshaking status
*	Write	Write data bit to serial port
*	Read	Read data bit from serial port
	Flush	
*	GetEventType	Save event
	GetError	
	GetCTS	
	GetDSR	
*	Close	Close serial port
	SetEventChar	
	SetMask	

Figure 4.4 shows sample code to open and configure a serial port using Klein's serial library [Klein, 2001]. In our research, we used de Klein's serial library in the software development of Sender and Receiver on Windows 2000.

```
CSerial serial;
LONG    lLastError = ERROR_SUCCESS;

// Attempt to open the serial port (COM2)
lLastError = serial.Open(_T("COM2"));

// Setup the serial port (9600,8N1, which is the
//default setting)
lLastError = serial.Setup
CSerial::EBaud19200,CSerial::EData8,CSerial::EParNone,CSerial::ESt
op1);

// Setup handshaking (default is no handshaking)
lLastError = serial.SetupHandshaking
(CSerial::EHandshakeHardware);
```

Figure 4.4 Sample code to open and configure a serial port using the Klein's serial library [Klein, 2001].

4.2.2 Modem initialization

Modem initialization is the first step for either Sender or Receiver to communicate with an external satellite transceiver (modem) through the serial port. Modem initialization methods for Iridium and Orbcomm are different due to different modem command compatibility. A separate method called init_modem is built for Iridium modem initialization. The method takes three parameters – satellite phone number, number of AT command bytes and a character ('s' for modem initialization and 'o' for modem termination) indicating that the method is either initializing or terminating the Iridium satellite modem. To initialize the Iridium modem, a string containing a satellite phone number for Receiver is written to the serial port after sending AT commands to configure the Iridium satellite modem. During satellite modem termination, AT commands are sent to switch the satellite modem from data mode to command mode, terminate the data link and disconnect the communication.

With each AT command sent through a serial port to an Iridium satellite transceiver (modem), either Sender or Receiver waits several seconds (2 to 20 seconds depending on the AT command) before it reads from the serial port for modem response. The serial port

reading is repeated until the correct AT response is found in the string read from the serial port. The satellite modem is then ready for the next AT command. The number of waiting seconds varies for different AT commands from 2 seconds for configuration AT commands to 20 seconds for dialing a satellite phone number. Dialing the satellite phone number containing 14 digits and 1 carriage return takes much longer than writing "AT" to the satellite modem to get its attention. Immediate reading from the serial port without waiting can cause unnecessary program execution and add extra burden to the CPU. Figure 4.5 shows the method for modem initialization. The Windows 2000 and QNX 4.25 versions of this method are essentially the same.

Modem initialization and configuration for the Orbcomm transceiver (Quake 1500 modem) is included in the packet of transferred data. As shown in Table 4.2, a SC-Originated message contains bytes such as data type, ACK level, Orbcomm gateway ID, priority level, recipient email address and message data type. No separate modem initialization method is required for the Orbcomm transceiver.

```
int init_modem(char command[], int bytes, char s){

 int t=0; char buffer[255]; DWORD dwBytesRead=0;
 bool modemFlag=true; string signal; int times=0;
 char first; char second;
 //determine if it is modem initialization of terminatation
 if(s=='s') times=3; if(s=='o') times=2;
 while(t<times){
    first='O'; second='K'; modemFlag=true;
    while(modemFlag){
       //it is modem initialization
       if(s=='s'){
          //it is "AT" command
          if(t==0){
             serial.Write("AT\r", 3); signal.assign("Ok!"); Sleep(2000);
          }
          //it is bearer set
          else if(t==1){
             serial.Write("AT+CBST=6,0,1\r",14);
             signal.assign("Data link set!"); Sleep(2000);
          }
          //it is dialing sat number
          else if(t==2){
             cout<<"Dialing..."<<endl; serial.Write(command, bytes);
             first='C'; second='O'; signal.assign("Connected!");
             Sleep(30000);
          }
       }
       //it is modem termination
       if(s=='o'){
          //it is escape sequence
          if(t==0){
             serial.Write("+++", 3); Sleep(2000); signal.assign("Data link
             off!");
          }
          //it is link disconnection
          else if(t==1){
             serial.Write("ATH\r", 4); Sleep(2000);
             signal.assign("Disconnected!");
          }
       }
       //keep reading from serial port until the right modem //response
       is detected
       dwBytesRead=0; serial.Read(buffer,sizeof(buffer)-1, &dwBytesRead);

       int response=0; modemFlag=true;
       cout <<"AT response: "<<buffer<<endl;
       while(modemFlag&&response<254){
          if(buffer[response]==first&&buffer[response+1]==second){
             modemFlag=false; buffer[response]=' ';
       buffer[response+1]=' '; cout<<signal<<endl;
          }
          response++;
       }}
    t++;}
 return 0;
}
```

Figure 4.5 The modem initialization method for the Iridium Motorola 9500 handset (`init_modem()` method on Windows 2000).

4.2.3 Fletcher Checksum

Detecting and, if possible, correcting data errors is crucial in network data communications, especially when using a wireless data link. With a one-bit loss or error a

runnable program can be corrupted so that it cannot run [Peterson, 2000]. An error detection and correction algorithm is very important at the link level of a communication network. Existing algorithms include Cyclic Redundancy Check (CRC), two-dimensional parity and Internet checksum. The Internet checksum algorithm simply sums (modulo 2^{16}) each byte of transmitted data and sends the result of the sum with the data to the receiving side. The receiving side does the same calculation to check if there is any error or loss in received data. This algorithm is very simple to implement using 1's complement, but is less protective than the CRC algorithm. If a pair of single-bit errors occurs, one increment and one decrement, the checksum algorithm is not able to detect the error since the sum of all bits remains the same [Seaman, 1995].

Fletcher's checksum is another implementation of a checksum algorithm which is a variant of the 1's complement algorithm. In Fletcher's checksum, the checksum is divided into two parts, each of which is in 16-bit hexadecimal format. To calculate the Fletcher's checksum for N bytes of data, the algorithm uses the following equations to calculate checksum bytes and insert them into the end of a data byte stream:

$$C_0 = \sum_{i=1}^{N} B_i \tag{4.1}$$

$$C_1 = \sum_{i=1}^{N} (N - i) B_i \tag{4.2}$$

where C_0 = first checksum byte, which is the sum of all data bytes, C_1 = second checksum byte, which is the sum of each data byte multiplied by its position index from the end of the data packet, and B_i = the i^{th} byte of data.

Fletcher's checksum algorithm is slower than the 1's complement algorithm and involves a pair of simultaneous equations. Although it adds some complexity to data error

checking, Fletcher's Checksum is an interesting possibility for binary applications [Seaman, 1995]. The code of Fletcher's checksum algorithm implementation for satellite system data communication testing is shown in Figure 4.6.

```
void MSC_vdFletcherEncode(unsigned char *buffer, int
count){
 int i;
 unsigned char c0=0, c1=0;
 //set both checksum bytes to zero
 *(buffer + count - 1) = *(buffer + count - 2) = 0;
 //sum up checksum bytes
 for( i = 0; i < count; i++ )
   {c0 += *(buffer+i);
    c1 += c0;
    *(buffer + count - 2) = c0;
    *(buffer + count - 1) = c1;}
}
```

Figure 4.6 Fletcher's checksum method (from [Fletcher, 1982] and [KBMW, 1999]).

4.2.4 Sender

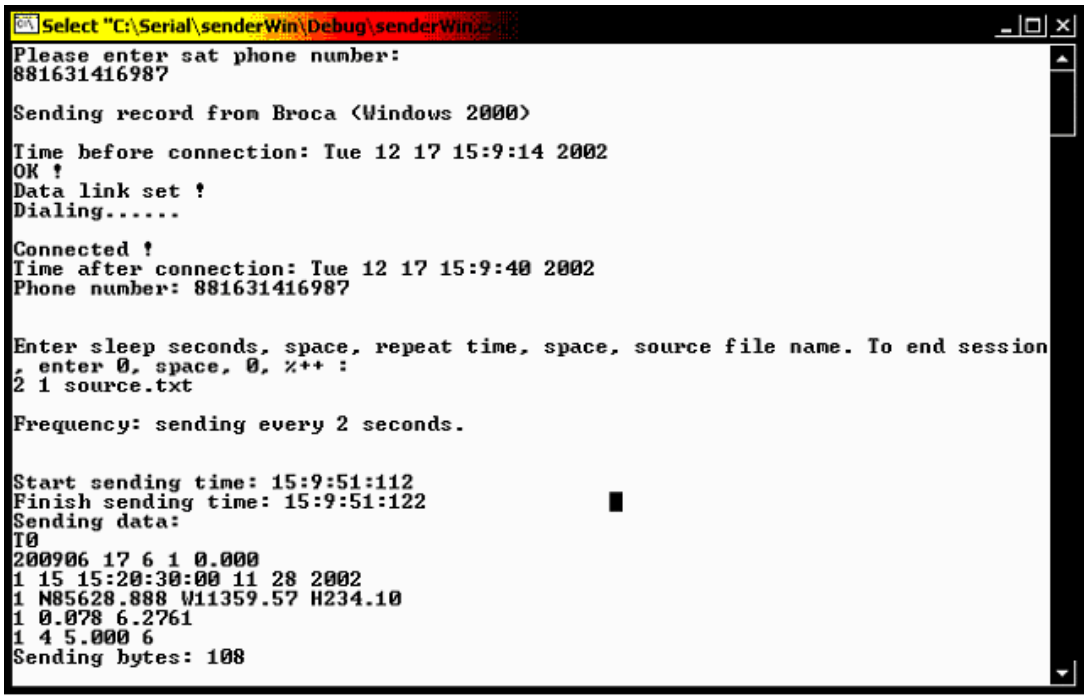
Sender is the class in satellite system data communication testing that is responsible for sending data to the satellite data link through a serial port for system latency testing. The data transferred into the satellite link is simulated GPS position data stored in a source file. For our research, we chose transmitted data with a size of 108 bytes, which approximately matches the length of a message in the Kodiak seismic survey operations. Figure 4.7 shows a sample message we used to test the satellite communication channels. A '^' character is used to indicate the end of each message.

```
T1
200906 17 6 1 0.000
1 15 15:25:30:00 12 11 2002
1 N85635.456 W11343.57 H222.70
1 0.078 6.2761
1 4 5.000 6^
```

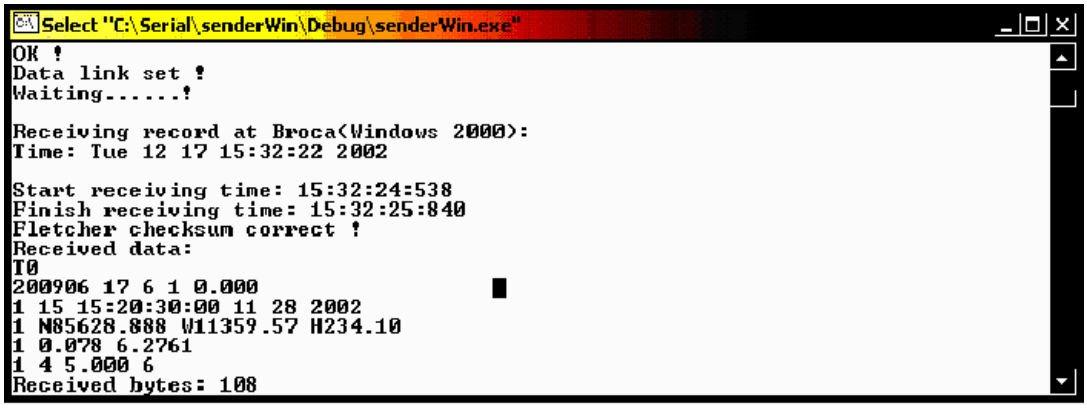
Figure 4.7 Sample data message (108 bytes long) for satellite communication channels testing.

After modem initialization and configuration, Sender dials the satellite phone number coded in the program and sets the satellite data link. This part of the operation only occurs in Iridium system testing. Data arrives at the Receiver side via an e-mail message for the Orbcomm system, and the e-mail address is included in the Orbcomm packet data message.

Once the satellite link is set, the Sender prompts the user to enter number of times M to repeat sending messages, time T (in seconds) to wait between sending messages and source data file name. The user is in control of entering various values of M and T to test satellite system latency. Figure 4.8 shows sample Iridium modem testing sessions (both sending and receiving) in operation. The times shown in (a) and (b) of Figure 4.8 do not correspond to the same transmission session.



(a) Sender



(b) Receiver

Figure 4.8 Sample Iridium modem testing sessions in operation.

Sending frequency is an important element in our satellite system latency testing. Sending data at different frequencies was used to the test data communication capability of the satellite link. The Iridium satellite system has a data transmission capability of

2400 bps and Orbomm's data transmission capability is 4800 bps. In our research the actual data transmission rate (maximum 1080 bps (assuming 10 bits are required to send one byte) when sending frequency is 1 Hz) is expected to be within both Iridium and Orbcomm system data transmission ability. Effective data rate is not necessarily equal to system design data rate, and our testing was designed to see if the Iridium and Orbcomm systems could keep up with our maximum sustained data rate.

In each session of data transmission, Sender opens the source data file, reads one set of simulated GPS data (N bytes long) and sends this data to the satellite data link. Once Sender reaches the end of the source data file, the data transmission session ends. Sender repeats the data transmission session M times, where M is entered by the user. Figure 4.9 shows the pseudo-code algorithm for reading a file of simulated GPS data.

```
Initialize Buffer for sending a string
Open the source data file
While (true) do
     While (true) do
          Read a line of data L
          Buffer ← Buffer + L
          If (first byte of L = "end of file") exit outer
              while loop
          If (last byte of L = "end of data set ('^')")
              exit inner while loop
     End of inner While loop
     Write data set message to serial port
End of outer While loop
Close source data file
```

Figure 4.9 Pseudo-code for reading a file of simulated GPS data.

When one sending session is finished, Sender prompts the user for the next session. Sender can either start another data transmission session with new sending parameters or enter special commands to end data communication and disconnect the satellite data link.

A record file named "sRecord.txt" is generated at the beginning of the Sender program to record data transmission information including transferred data, error

checking results and system time for each transmitted data message. For each N bytes of transmission message, Sender records the system time when data sending starts. The recorded file information is used to compare with a Receiver side file of the same name (sRecord.txt) to compute satellite system data transmission latency and determine if data errors occurred during transmission. Figure 4.10 is an example of part of a file recorded at the Sender side.

```
Sending record from Broca (Windows 2000)
Time before connection: Wed 12 11 16:20:50 2002
Time after connection: Wed 12 11 16:21:24 2002
Phone number: 881631416988
Frequency: sending every 1 seconds.

Start sending time: 16:21:30:664
Finish sending time: 16:21:30:714
Sending data:
T0
200906 17 6 1 0.000
1 15 15:20:30:00 12 11 2002
1 N85628.888 W11359.57 H234.10
1 0.078 6.2761
1 4 5.000 6
Sending bytes: 108

Start sending time: 16:21:31:755
Finish sending time: 16:21:31:815
Sending data:
T1
200906 17 6 1 0.000
1 15 15:25:30:00 12 11 2002
1 N85635.456 W11343.57 H222.70
1 0.078 6.2761
1 4 5.000 6
Sending bytes: 108
```

Figure 4.10 Example portion of a data transmission summary file recorded by Sender.

The Sender programs for Windows 2000 and QNX 4.25 are essentially identical. The only difference arises from the different file I/O processes of the two operating systems.

Sender programs for the Iridium and Orbcomm systems are different. For Iridium system testing, Sender is dealing with data and associating with the receiving side at the link level. Each data string D transmitted is inserted with an additional three bytes of

overhead. The last byte character '&' indicates end of this N bytes of simulated GPS data. The second and third last bytes are Fletcher's checksum bytes calculated by Sender. The three terminating bytes are used at the Receiver side to separate and handle each specific data transmission.

For Orbcomm system testing, the receiving side receives data via e-mail through SMTP. An Orbcomm data string is a packet of hexdecimal data as described in Table 4.2 in section 4.1.4. An Orbcomm data string contains several bytes of overhead including recipient e-mail address, type of data format, Fletcher's checksum, total data bytes, ACK level, priority level and gateway ID. This packet data has commands for Orbcomm modem initialization and configuration as well as complete data transmission instructions for the Orbcomm data network's TCP/IP layers. The overhead is 17 bytes if the recipient's e-mail address has been predefined in user account when the Orbcomm transceiver is activated and byte No. 13 in Table 4.2 is used to indicate this. When the value of byte No. 13 is '01', the Orbcomm system will look for the actual e-mail address in the user account instead of the data string.

Figure 4.11 shows the Iridium Sender algorithm for Windows 2000 and Figure 4.12 shows the unique part of the Orbcomm Sender algorithm on QNX 4.25.

```
CSerial serial
serial.Open(_T("COM2"))
serial.setup(Baud R,Data bit B, Parity P, Stop bit S)
init_modem(PN, PN.length(), 's') // PN is sat phone No.
Open record file r("sRecord.txt")
While (true) do
   Prompt user for user input sending command S
      including sleep seconds T, repeat times M and
      data file name

   If (S = "%++") exit outer while loop
   Else
       For(M) // M is no. of times to repeat sending
           Open the source data file
           While(true) do
               Fetch one set of data D from source data
                   file (as inner while loop in Figure
                   4.9)
               N ← total number of data bytes in D
               SYSTEMTIME tm
               MSC_vdFletcherEncode(D, N+2),
               D[N+2] ← '&'
               t1 ← GetSystemTime(&tm)
               serial.Write(D, N+3) /* send data string
                   to serial port */
               t2 ← GetSystemTime(&tm)
               Write t1, t2 and D into r /* record data
                   and time */
               Sleep(T) /* wait T seconds to send next
                   data */
           End of inner While loop
           Close source data file
       End of For loop
   End If
End of outer While loop
Close r //close record file
init_modem("off", 3, 'o') // terminate data link
serial.close() // close serial port
```

Figure 4.11 Sender pseudo-code for Iridium on Windows 2000.

```
sdf ← open("dev/ser2") //open serial port #2
Configure serial port and set input option as raw
Construct packet string D with basic Orbcomm bytes

int w /* w is the actual no. of bytes successfully written to the
   serial port during one serial port writing. This value is returned
   by write (serial port file, data string, no. of bytes) method */

While (true) do
   Prompt user for user input sending command S including sleep
      seconds T, repeat times M and data file name

   If (T = "9999") //indicate stop sending session
       exit outer While loop
   Else
       For(M times) /* M is no. of times to repeat sending  */
           Open the data file
           While(true) do
              Fetch one set of data ds /* as inner while loop in Fig.
                 4.9 */
              D ← D + ds, N ← length of D
              timeb tm /* system time variable for QNX system */
              MSC_vdFletcherEncode(D, N) /*insert checksum bytes into
                 data string */
              t1 ← ftime(&tm) //get system time
              w = write(sfd, D, N) /* w is no. of bytes successfully
                 written to serial port */
              If (w = N)  /* when all bytes of the data string are
                 successfully written to the serial port */
                 t2 ← ftime(&tm), Write t1,t2 and D to r
              End of If
              Sleep(T)
           End of inner While loop
           close the data file
       End of For loop
   End of If Else
End of outer While loop
Close r, close(sfd)//close record file and serial port
```

Figure 4.12 The different part of the Sender pseudo-code for Orbcomm on QNX 4.25.

4.2.5 Receiver

A Receiver was built only for Iridium system testing since the Orbcomm system sends data in e-mail format to the Receiver side. Figure 4.13 shows the steps of serial port reading at the Receiver side and Figure 4.14 is the pseudo-code of the Receiver program on Windows 2000.

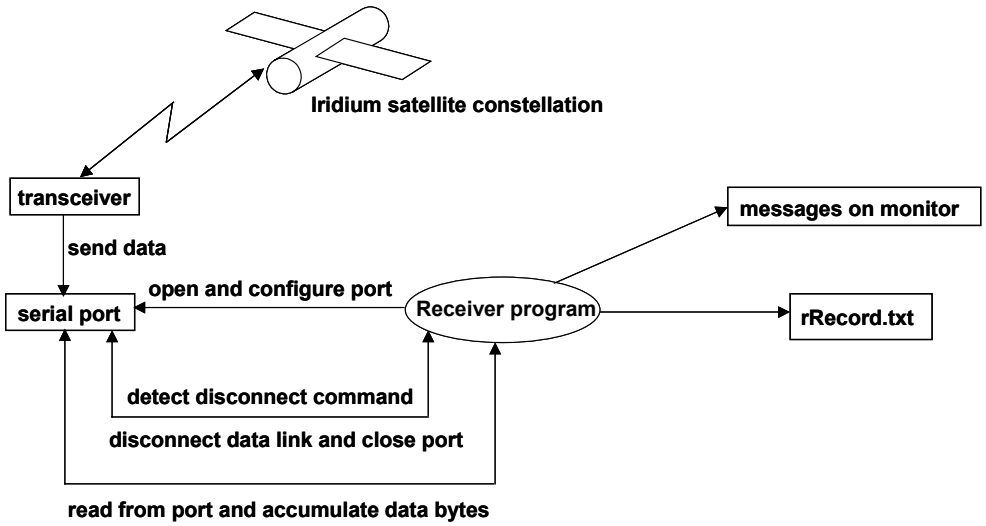


Figure 4.13 Steps of serial port reading at the Receiver side for Iridium communication.

As for Sender, Receiver starts with opening a serial port, setting the serial port, satellite modem initialization and configuration (see lines 1 to 3 in Figure 4.14). Receiver then listens to the serial port and waits for any incoming data signal from the attached satellite transceiver (modem) as described in lines 7 to 9. Reading data from the serial port and handling data involves data byte accumulation and data error checking. Since the testing is focused on satellite data transmission latency, the system time t_3 (epoch that the first data byte arrives) and t_4 (epoch that the last data byte arrives) of each data arrival is recorded. Lines 20, 21, 30 and 33 of the pseudo-code in Figure 4.14 show how the system time is recorded. The data arrival time is meaningful for satellite system latency calculations only if the Receiver detects a data byte at the serial port as soon as the data byte arrives. Serial port reading needs to be prompt and continuous.

Each serial port has an input buffer as initialized in line 4 of Figure 4.14. Serial port reading reads data bytes from the input buffer. A user has control of the size of the input buffer, which is the maximum number of data bytes for each serial port reading. The

actual number of data bytes read (rb in lines 12 and 27) each time depends on how many bytes of data are in the input buffer at the moment. A user has no control on the actual number of data bytes read each time. A serial read resulting in zero bytes occurs when there is no data in the input buffer. Zero bytes read can happen at any time during data reading and it does not indicate the end of data transmission. The Receiver program accumulates data bytes and waits for arrival of the "end-of-data" byte ('&') to determine when the end of current data transmission is received (line 32). Once the end indicator byte of the current data transmission is detected, Receiver handles the accumulated data, prints data to the monitor, writes data to the output file and calculates Fletcher's checksum on the received data to compare with the Sender's checksum result. This procedure is handled by a function of `HandleData(fs, N)` (lines 25 and 34) written separately in the Receiver program. Receiver records and shows an error message if its checksum disagrees with the Sender's checksum.

```
1. CSerial serial, serial.Open(_T("COM2"))
2. serial.setup(Baud R,Data bit B, Parity P, Stop bit S)
3. init_modem('s'), Open record file r("rRecord.txt")
4. initialize Buffer, N ← total number of read bytes
5. fs ← final data string
6. SYSTEMTIME tm  //system time variable
7. While (true) do
8.    e ← serial port event /* check serial port event (data receive,  break,
      error) */
9.    If(e = receive event)
10.      While (true) do
11.         Buffer ← data bytes read from serial port
12.         rb ← number of bytes read currently
13.         If (Buffer[0] = ending session byte '%')
14.           exit outer while loop
15.         Else
16.          If(Buffer[0] !=data start byte 'T'&& rb > 0)/* current read data
               bytes are the middle part of a data set */

              /* check special case when current read data bytes containing the
              start bytes of next data set */
17.           int k ← 1 // look for the start byte of next data set
18.           While(true)
19.             If (Buffer[k]='T')
20.               t5 ← GetSystemTime(&tm) /* start receiving time for next
                        data transmission */
21.               t4 ← GetSystemTime(&tm)/* finish receiving time for current
                        data transmission */
22.               C1 ← data bytes before k in Buffer (current transmission)
23.               C2 ← data bytes after k in Buffer (next transmission)
24.               fs ← fs + C1 , Buffer ← C2
25.               N ← N + rb, HandleData(fs, N) /* call method to handle data
                   set and calculate checksum */
26.               write t3, t4 and fs to r, N ← 0, t3 ← t5 /* assign t5 to t3
27.               rb ← rb – (k-1), , Exit inner while loop
                Else
28.               k ← k + 1
                End if
              End of inner while loop (line 18)
29.         Else If (Buffer[0]=data start byte 'T') /* case incoming data is the
                start of a data set and special case did not happen */
30.           t3 ← GetSystemTime(&tm) // get start time
            End if
            If(rb!=0)  //reading bytes is not zero
31.           fs ← fs + Buffer, N ← N + rb
            End If
32.         If(fs[last]=data set ending byte '&') // it is the end of data
33.           t4 ← GetSystemTime(&tm) // finish time
            End If
          End If
        End of second inner While loop (line 10)
34.     HandleData(fs, N), write t3, t4 and fs to r
     End If
   End of outer While loop (line 7)
35.Close r, init_modem('o'), serial.close()
```

Figure 4.14 Pseudo-code for the Receiver program on Windows 2000.

On occasion, the data read on the Receiver side happens to include the last several bytes of the current data transmission and several start bytes of the next data transmission. This happens more when sending frequency increases and the next data transmission is closer to the previous one. Lines 17 to 28 in Figure 4.14 show how the Receiver program handles this occurrence. In this case, Receiver separates the data

stream into two parts (C_1 in line 22 and C_2 in line 23) and handles each part separately. The same system time is recorded as finishing receiving time t_4 for the current data transmission (line 21 in Figure 4.14) and start receiving time t_5 for the next data transmission (line 20 in Figure 4.14). The value of t_5 is assigned to the variable t_3 after the record of current data transmission has been written into the output file (lines 25 and 26 in Figure 4.14). This avoids over writing t_3 of current data transmission

When a user decides to stop data transmission and enters the "stop sending" command on the Sender side, Sender ends the session and sends an AT command to terminate the satellite modem and disconnect the data link. Upon receiving the disconnect command as described in line 13 of Figure 4.14, the satellite transceiver at the Sender side changes from data mode to command mode and transmits a disconnection signal to the satellite transceiver at the Receiver side. The Receiver program then ends its data reading session, switches the satellite transceiver from data mode to command mode, disconnects the satellite data link and closes the serial port. Receiver coding on Windows 2000 is similar to Receiver coding on QNX 4.25 except for file I/O.

The output file "rRecord.txt" is generated to record data receiving information. The Receiver program records the system time when the first incoming data byte of the current transmission is detected and also when the ending data byte is received. With N = 108 bytes of data, the start receiving time and finish receiving time are significantly different. The satellite system latency calculation for each data transmission (see section 4.2.6) uses the finish receiving time on the receiver side (t_4 in Figure 4.14 above) and the

start sending time on the sender side (t_1 in Figure 4.11). A portion of the rRecord.txt file for Receiver is displayed in Figure 4.15.

```
Receiving record at Broca (Windows 2000)
Time: Wed 12 11 15:41:37 2002

Start receiving time: 15:41:54:245
Finish receiving time: 15:41:54:786
Fletcher checksum correct!
Received data:
T0
200906 17 6 1 0.000
1 15 15:20:30:00 12 11 2002
1 N85628.888 W11359.57 H234.10
1 0.078 6.2761
1 4 5.000 6
Received bytes: 108

Start receiving time: 15:41:55:227
Finish receiving time: 15:41:55:768
Fletcher checksum correct!
Received data:
T1
200906 17 6 1 0.000
1 15 15:25:30:00 12 11 2002
1 N85635.456 W11343.57 H222.70
1 0.078 6.2761
1 4 5.000 6
Received bytes: 108
```

Figure 4.15 Example portion of a rRecord.txt file recorded by the Receiver.

4.2.6 Computing satellite system latency

Calculation of satellite system latency is based on system time recorded in output files from Sender and Receiver (Iridium) (the receiving side time is the e-mail receiving time for Orbcomm). Equation (4.3) shows how the satellite system latency was calculated. Satellite system latency L_i is calculated as:

$$L_i = t_i^r - t_i^s \tag{4.3}$$

where L_i = system latency for data message i in seconds, t_i^r = time start of message i was received by the Receiver program and t_i^s = time start of message i was sent by the Sender program.

Both t_i^r and t_i^s are system time and were obtained by using system time methods on Windows 2000 and QNX 4.25. On Windows 2000, a system call of `GetSystemTime(&time)` (`time` is a `SYSTEMTIME` type variable) was used to get system time. On QNX 4.25 a system call of `ftime(&t)` (`t` is a `timeb` type variable defined in the QNX `sys/timeb.h` file) was used to get system time.

To make both t_i^r and t_i^s comparable and the time difference meaningful as system latency, system time on both Sender and Receiver was synchronized with the same time server. Time synchronizing technologies on Windows 2000 and QNX 4.25 are different. The Dimension 4 time synchronizer for Windows [Mills, 1998] was downloaded to Windows operating systems including Broca and the laptop computer used for dynamic Iridium testing. Xntp [Mills, 1996], version 3.5 of Network Time Protocol (NTP) was used for QNX1. Dimension 4 and Xntp were chosen out of many available time synchronizers because they both use the SNTP time protocol and allow the user to choose from a list of active NTP time servers. In this research, both time synchronizers poll ntp.unb.ca at the University of New Brunswick to synchronize system time for Broca, the laptop used for the Iridium dynamic testing and QNX 1. The time adjustment accuracy using Dimension 4 and Xntp is approximately 50 ms [Mills, 1998] [Mills, 2001].

Chapter 5. Testing and results analysis

The computers for satellite data transmissions were set up in Gillin Hall Room E112, the Hydrography Lab of the Geodesy and Geomatics Department at the University of New Brunswick (Fredericton, NB, Canada). To achieve the best satellite communication signal, the satellite antennae were installed on the roof of Gillin Hall, where the satellite antennae have the best view of the sky without being blocked by any structures.

5.1 Test environment

Satellite data transmission testing was between Broca (Windows 2000, the middle computer in Figure 5.1) and QNX1 (QNX 4.25, the right most computer in Figure 5.1). Two 60 foot (18 m) long RS232 DB9 cables were used to connect the satellite transceivers (one for sending and one for receiving) on the roof (as shown in Figure 5.2) to the computer serial ports in the lab at Gillin Hall.

Figure 5.1 Computer setup in Room GE112 (Hydrography Lab) of Gillin Hall.

(a) Orbcomm (right) and Iridium (left) antennae.

(b) Iridium Sensor Systems antenna.

Figure 5.2 Antennae and satellite transceiver setup on the roof of Gillin Hall.

Data signal loss on the antenna cable between the satellite antenna and satellite transceiver is an important element affecting the satellite data transmission. A low-loss cable is critical for the Iridium system which operates at 1610 – 1626.6 MHz in both transmitting and receiving. For the Orbcomm system, the antenna cable data loss is less important since Orbcomm operates at 148.0 – 150.05 MHz (uplink) and 137.0-138.0

MHz (downlink). To meet the requirement of total 3dB maximum antenna cable data loss for the Iridium system, a 10 foot (3 m) LMR400 antenna cable assembled with two male TNC connectors obtained from Cabco Inc., Halifax, N.S. was used to connect the Iridium satellite antenna and satellite transceiver (Motorola 9500 phone). Figure 5.3 shows the testing antenna layout on the roof of Gillin Hall.

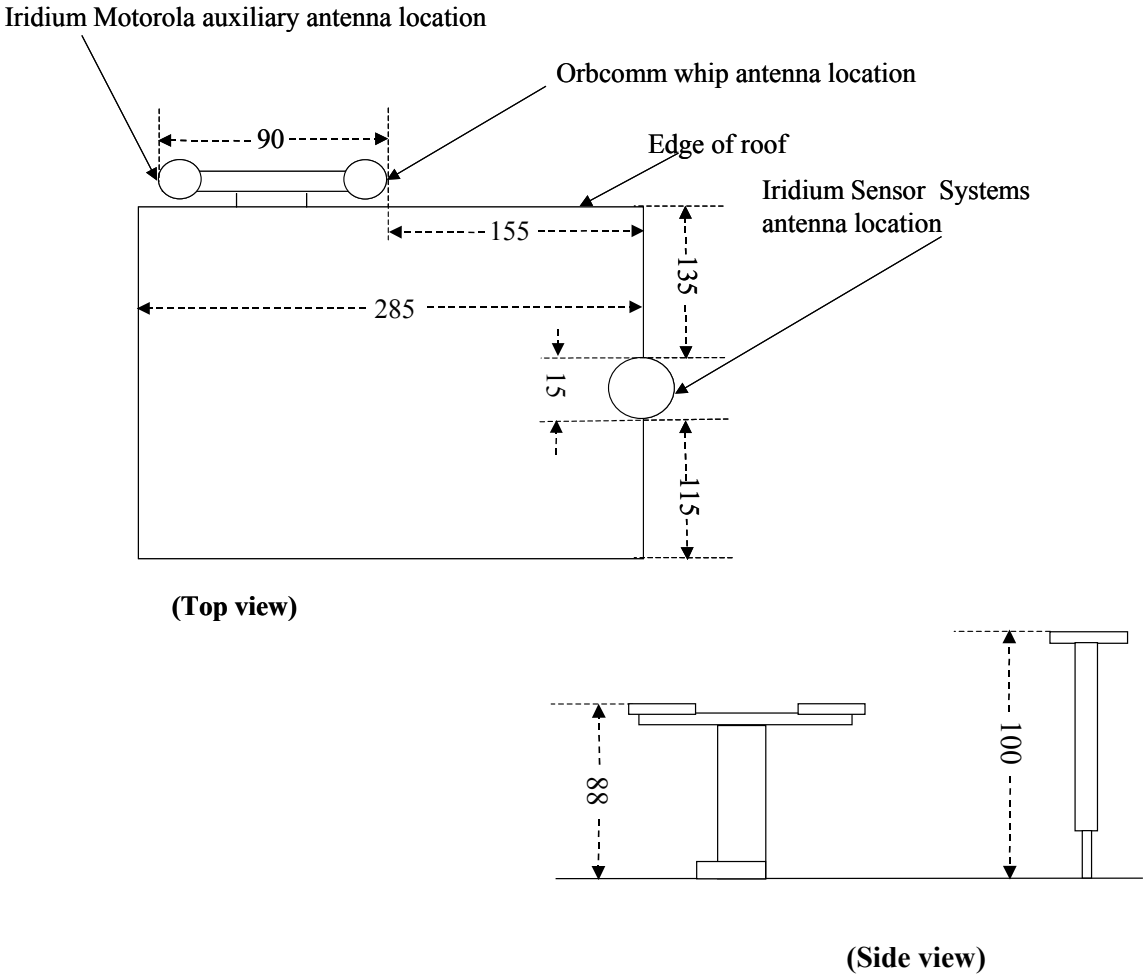


Figure 5.3 Satellite antenna layout on the roof (all units in cm).

5.2 Iridium experiments

Among three data services provided by the Iridium system, we chose the Mobile Terminated Data Service for Iridium data transmission latency testing. As explained in

section 3.1.1.2, a Motorola 9500 satellite phone was set up at each end of the satellite link for the experiments.

5.2.1 Static testing

In seismic survey operations satellite equipment can be used in a static environment such as a base station and in a dynamic environment such as a moving helicopter. The transmission environment can affect the satellite system latency. We measured the Iridium satellite system latency in both static and dynamic environments.

Static testing was performed when both Sender and Receiver were stationary. To probe overall availability of data channels of the Iridium satellite system, static testing was done at three different time periods - morning, afternoon and night on a regular working day. The sending frequency was also varied to test the Iridium system data transmission capability. Sending frequencies used were 1, 0.5 and 0.25 Hz, corresponding to sending data every 1, 2 and 4 seconds. A simulated GPS position message (see section 4.2.4) of 108 bytes was sent for each transmission. Each frequency test was conducted 40 times (20 times sending from Broca to QNX1 and 20 times from QNX1 to Broca). The average Iridium system latency and standard deviation are recorded in Tables 5.1 to 5.3 for three time slots, morning (8:00 am – 12:00 noon), afternoon (2:00 pm – 5:00 pm) and evening (7:00 pm – 10:00 pm) Atlantic time on November 12, 2002. The data link loss rate is the percentage of times that the Iridium satellite data link was lost during data transmission sessions. On the morning of November 12, 2002, one of the 40 transmission tests (2.5%) had a dropped satellite link. Two of 40 transmissions (5%) were lost during the afternoon of November 12, 2002. Testing results are analyzed in section 5.2.3.

Table 5.1 Iridium static testing results during the morning of November 12, 2002.

frequency (Hz)	*N*	average latency (ms)	standard deviation (ms)	data link loss rate (%)
1	40	1460	560	0
0.5	40	1781	581	2.5
0.25	40	1516	460	0
average	40	1585.6	533.6	8.3

Table 5.2 Iridium static testing results during the afternoon of November 12, 2002.

frequency (Hz)	*N*	average latency (ms)	standard deviation (ms)	data link loss rate (%)
1	40	1560	336	0
0.5	40	1186	815	0
0.25	40	1197	676	5.0
average	40	1314.3	609	1.7

Table 5.3 Iridium static testing results during the evening of November 12, 2002.

frequency (Hz)	*N*	average latency (ms)	standard deviation (ms)	data link loss rate (%)
1	40	1420	398	0
0.5	40	1126	535	0
0.25	40	1078	629	0
average	40	1208	520.7	0

5.2.2 Dynamic testing

Dynamic testing with the Iridium system tested system latency with one satellite transceiver moving. The dynamic test system was set up with one Windows 2000 laptop in a moving car to simulate helicopter movement. An additional element taken into consideration in dynamic testing is the speed of the car. The focus of dynamic testing is to compare data transmission latency, data error and data loss rate with static testing to check if dynamic movement affects the Iridium system's data transmission function. Figure 5.4 shows the equipment layout of the moving Iridium transceiver.

Figure 5.4 Dynamic Iridium transceiver layout.

Each dynamic test was performed 40 times (20 times from the laptop to QNX1 and 20 times from QNX1 to the laptop). Tables 5.4 and 5.5 show results for dynamic testing with different testing parameters. Tests were done at 4:45 pm – 6:00 pm on November 28, 2002 and 2:30 pm – 3:30 pm on December 11, 2002 Atlantic time. Dynamic testing results are analyzed with static testing results in section 5.2.3.

Table 5.4 Iridium dynamic testing results during the afternoon of November 28, 2002.

speed (km/h)	*N*	frequency (Hz)	average latency (ms)	standard deviation (ms)	data link loss rate (%)
50	40	0.5	1544	95	0
	40	0.25	1454	51	0
110	40	0.5	1356	225	0

Table 5.5 Iridium dynamic testing results during the afternoon of December 11, 2002.

speed (km/h)	*N*	frequency (Hz)	average latency (ms)	standard deviation (ms)	data link loss rate (%)
50	40	1	1510	105	0
110	40	1	1558	89	0

5.2.3 Analysis of testing results

We used statistical analysis on both static and dynamic data transmission latency testing results of the Iridium satellite system. The two-sample *t*-test for independent samples with equal variances (two variances are equal or not significantly different) or unequal variances (two variances are significantly different) [Rosner, 1990] was employed.

5.2.3.1 Statistical testing

Two-sample *t*-test statistical testing was used to compare the average system latency of two different data transmission scenarios. Our null hypothesis H_0 is $\overline{L}_1 = \overline{L}_2$ where $\overline{L}_1$ = the average system latency in the first data transmission test scenario, and $\overline{L}_2$ = the average system latency in the second transmission test scenario. We used the following equations to calculate the test statistic value *t* for the testing cases with equal variances:

$$s = \sqrt{\frac{\left((n_1 - 1)S_1^2 + (n_2 - 1)S_2^2\right)}{(n_1 + n_2 - 2)}} \tag{5.1}$$

$$t = \frac{\bar{L}_1 - \bar{L}_2}{\left(s\sqrt{\frac{1}{n_1} + \frac{1}{n_2}}\right)} \tag{5.2}$$

where n_1 = sample size of the first data transmission test scenario, n_2 = sample size of the second transmission test scenario, s_1 = standard deviation of the first data transmission test scenario, s_2 = standard deviation of the second data transmission test scenario,

$\bar{L}_1$ = the average system latency of the first scenario and $\bar{L}_2$ = average system latency of the second case [Rosner1990].

With the calculation result of the *t* value, we compare the *t* value with the value of $t_{n1+n2-2,\ 1-\alpha/2}$ found in the *t* distribution table. If the *t* value lies between $-t_{n1+n2-2,\ 1-\alpha/2}$ and $t_{n1+n2-2,\ 1-\alpha/2}$, H_0 is accepted at a significance level of $1-\alpha$; otherwise H_0 is rejected [Rosner, 1990].

For the cases with unequal variances, the following equations are used to calculate the *t* value:

$$t = \frac{\bar{L}_1 - \bar{L}_2}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}} \tag{5.3}$$

$$d' = \frac{\left(\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}\right)^2}{\left(\frac{\left(\frac{s_1^2}{n_1}\right)^2}{(n_1 - 1)} + \frac{\left(\frac{s_2^2}{n_2}\right)^2}{(n_2 - 1)}\right)} \tag{5.4}$$

where d' = the approximate degrees of freedom.

We rounded up d' to the nearest integer *d* and compared the *t* value with the value of $t_{d,\ 1-\alpha/2}$ from the *T* distribution table. If the *t* value is between $-t_{d,\ 1-\alpha/2}$ and $t_{d,\ 1-\alpha/2}$, H_0 is accepted at a significance level of $1-\alpha$; otherwise H_0 is rejected [Rosner, 1990].

5.2.3.2 Testing results analysis

Table 5.6 shows average system latencies and standard deviations based on the static testing results in the morning, afternoon and evening.

Table 5.6 Average system latencies and standard deviation at different frequencies and different time of the day.

Frequency	sample size (n)	average latency ($\bar{L}$)	standard deviation (s)
1	40	1480	431
0.5	40	1364	644
0.25	40	1264	588
Time of day			
morning	40	1585.6	533.6
afternoon	40	1314.3	609
evening	40	1208	520.7

The standard deviations at different frequencies and different time of the day are not significantly different. Thus we used equations (5.1) and (5.2) to calculate the *t* values. We have two testing result comparisons at various frequencies, 1 Hz vs 0.5 Hz, 0.5 Hz vs 0.25 Hz, 1 Hz vs 0.25 Hz, morning vs afternoon, afternoon vs evening and morning vs evening. Table 5.7 shows the calculation results of the *t* values with $\alpha = 0.01$.

Table 5.7 Results of the *t* values for Iridium static testing (equal variance assumption).

Comparison	$\bar{L}_1 - \bar{L}_2$ (ms)	$s\sqrt{\frac{1}{n_1}+\frac{1}{n_2}}$	t	$t_{78,\ 0.995}$
1 and 0.5	116	122.5	0.947	2.640
0.5 and 0.25	100	137.9	0.725	2.640
1 and 0.25	216	115.3	1.873	2.640
morning vs afternoon	271.3	127.9	2.119	2.640
afternoon vs evening	106.3	126.6	0.840	2.640
morning vs evening	377.6	117.7	**3.208**	2.640

From the Table 5.7 we can see that the *t* value in frequency comparison cases are within the range of the $-t_{n1+n2-2,\ 1-\alpha/2}$ and $t_{n1+n2-2,\ 1-\alpha/2}$. We can conclude that the system latencies at different frequencies in static testing are not significantly different at the 99% confidence level.

The available data rate provided by the Iridium system is 2400 bps, or approximately 240 bytes every second. Our maximum testing data transmission rate of 111 bytes (108 bytes data with 3 additional bytes) every second is less than half of the available data rate provided by Iridium.

Static testing also shows that during different times of the day, data transmission latency is slightly different due to the traffic on the Iridium satellite system's data channels. Figure 5.5 is plots the average Iridium data transmission latency for the three time slots. Data transmission in the evening has the lowest average system latency. Results of hypothesis testing for different time of a day from Table 5.7 indicate that only the average system latency in the morning is significantly different from the latency in the evening at the 99% confidence level.

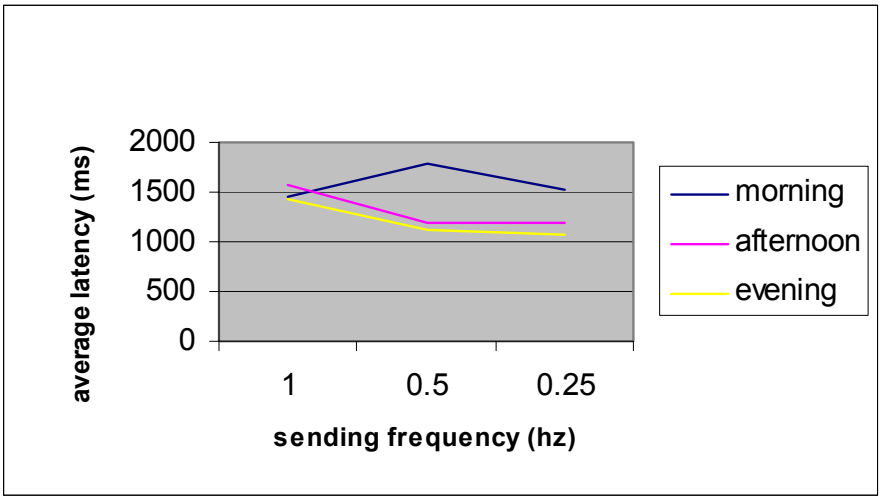


Figure 5.5 The Iridium system average data transmission latency for static testing.

Table 5.8 shows the average system latencies, standard deviations and *t* values of the static and dynamic testing results for our observed frequencies (1 Hz and 0.5 Hz) and time slots with the α value of 1%. Since the variances of static and dynamic testing are

significantly different, equations (5.3) and (5.4) are used to calculate t values. The sample size for each testing case is 40.

Table 5.8 Calculation results of t values for Iridium static and dynamic testing.

Comparison		average latency ($\overline{L}$)	standard deviation (s)	t	$t_{d,\ 0.995}$
1 Hz	dynamic	1534	97	0.773	2.695 (d = 43)
	Static	1480	431		
0.5 Hz	dynamic	1450	160	0.820	2.692 (d = 44)
	Static	1364	644		

We can see that in both scenarios the t values are between $-t_{d,\ 1-\alpha/2}$ and $t_{d,\ 1-\alpha/2}$, which indicates that the system latencies of static and dynamic data transmissions are not significantly different at the 99% confidence level. Testing result indicate that movement does not affect Iridium system latency, at least for the vehicle speed (up to 110 km per hour) we tested. The average system latency also does not increase when the moving speed is higher. The Iridium system works well in a dynamic environment. The testing results indicate that the Iridium system would perform well in both static and dynamic seismic survey scenarios.

Modem initialization for the Iridium system is the first step of satellite communication. Testing shows that modem initialization and data link setup in the dynamic situation takes longer than in the static situation. Average modem initialization time in static testing is 46 seconds and 83 seconds in dynamic testing. The maximum modem initialization time in the static case is 61 seconds compared with 5 minutes during our dynamic testing.

The data error rate is defined as percentage of times where a received data bit is different from the data bit sent by the sending side during the satellite data transmission.

For the Iridium system in both static and dynamic scenarios, testing results show that the data error rate is 0% in all testing cases. Data link loss only occurred three times in more than 1000 data transmissions. Testing results show that the Iridium system has good reliability.

5.3 Orbcomm experiment

Orbcomm system data transmission latency was tested only for the static case. Orbcomm testing uses one satellite transceiver as Sender that sends data through the Orbcomm network. The receiving side of the Orbcomm testing system receives data in email format.

5.3.1 Static testing

Static testing of the Orbcomm system was designed to test Orbcomm data transmission latency at different time slots of the day with various frequencies. As for the Iridium system testing, each Orbcomm test was performed 40 times (20 times from Broca to QNX1 and 20 times from QNX1 to Broca) to achieve reliable statistical information of Orbcomm system latency. The test data is the same 111 bytes simulated position data as was used for Iridium testing. Tables 5.9 to 5.11 shows static testing results for Orbcomm system data transmission. Testing was conducted during the morning, afternoon and evening (as defined for the Iridium testing) on November 12 and November 14, 2002.

Table 5.9 Orbcomm static testing results during the morning of November 12&14, 2002.

frequency (Hz)	average latency (min sec)	standard deviation (min sec)	data link loss rate (%)
0.5	14 min 39 sec	9 min 59 sec	0
0.25	15 min 0 sec	7 min 3 sec	0
0.125	18 min 12 sec	11 min 23 sec	0
0.0625	21 min 30 sec	7 min 54 sec	0
average	17 min 20 sec	9 min 4 sec	0

Table 5.10 Orbcomm static testing results during the afternoon November 12&14, 2002.

frequency (Hz)	average latency (min sec)	standard deviation (min sec)	data link loss rate (%)
0.5	22 min 12 sec	8 min 58 sec	0
0.25	6 min 18 sec	3 min 57 sec	0
0.125	9 min 35 sec	3 min 40 sec	0
0.0625	5 min 0 sec	2 min 53 sec	0
average	10 min 46 sec	4 min 52 sec	0

Table 5.11 Orbcomm static testing results during the evening of November 12&14, 2002.

frequency (Hz)	average latency (min sec)	standard deviation (min sec)	data link loss rate (%)
0.5	8 min 12 sec	7 min 2 sec	0
0.25	9 min 30 sec	3 min 12 sec	0
0.125	7 min 6 sec	3 min 30 sec	0
0.0625	15 min 35 sec	5 min 2 sec	0
average	10 min 9 sec	4 min 41 sec	0

5.3.2 Analysis of results

The same statistical testing used for the Iridium testing results was used to analyze the Orbcomm system latency testing results. Table 5.12 shows average system latencies and standard deviations based on the static testing results of the morning, afternoon and evening.

Table 5.12 Average system latencies and variances at different frequencies.

frequency (Hz)	sample size (N)	average latency ($\bar{L}$)	standard deviation (s)
0.5 (2 sec)	40	15 min 01 sec	8 min 40 sec
0.25 (4 sec)	40	10 min 16 sec	4 min 43 sec
0.125 (8 sec)	40	11 min 38 sec	6 min 11 sec
0.0625 (16 sec)	40	14 min 02 sec	5 min 16 sec

We compare the Orbcomm testing results in nine cases based on frequency and time of day. In each case, the standard deviations for different frequencies are not significantly different. The t values are calculated by equations (5.1) and (5.2). Table 5.13 shows the calculation results for t values with $\alpha = 0.01$.

Table 5.13 Results of the *t* values for Orbcomm testing (equal variance assumption).

case	Comparison	$\bar{L}_1 - \bar{L}_2$	$s\sqrt{\frac{1}{n_1}+\frac{1}{n_2}}$	t	$t_{78,\ 0.995}$
#1	0.5 vs 0.25	4 min 55 sec	1 min 36 sec	**3.073**	2.640
#2	0.5 vs 0.125	3 min 23 sec	1 min 41 sec	2.010	2.640
#3	0.5 vs 0.0625	0 min 59 sec	1 min 36 sec	0.615	2.640
#4	0.25 vs 0.125	1 min 22 sec	1 min 14 sec	1.108	2.640
#5	0.25 vs 0.0625	3 min 46 sec	1 min 7 sec	**3.373**	2.640
#6	0.125 vs 0.0625	2 min 24 sec	1 min 17	1.091	2.640
#7	morning vs afternoon	6 min 20 sec	1 min 38 sec	**2.653**	2.640
#8	afternoon vs evening	0 min 37 sec	1 min 4 sec	0.578	2.640
#9	morning vs evening	7 min 11 sec	1 min 37 sec	**4.463**	2.640

Calculation results in Table 5.13 show that the *t* values in case #2, #3, #4, #6 and #8 are within the range of the $-t_{n1+n2-2,\ 1-\alpha/2}$ value and the $t_{n1+n2-2,\ 1-\alpha/2}$ value. In cases #1, #3, #7 and #9, *t* values are out of the $-t_{n1+n2-2,\ 1-\alpha/2}$ and the $t_{n1+n2-2,\ 1-\alpha/2}$ value range. The null hypothesis in two out of six cases is rejected in the frequency comparisons, and for two out of three cases in the comparisons of different transmission time of day. With a confidence level of 99%, the conclusion is that the Orbcomm system data transmission latencies are not affected by frequency of transmission in most cases. For different transmission time of day, the Orbcomm system latencies are significantly different (at 99% confidence level) in the morning compared to the afternoon and evening transmissions. The lowest average system latency occurs in the afternoon session. Figure 5.6 plots Orbcomm system latency for different frequencies and time of day.

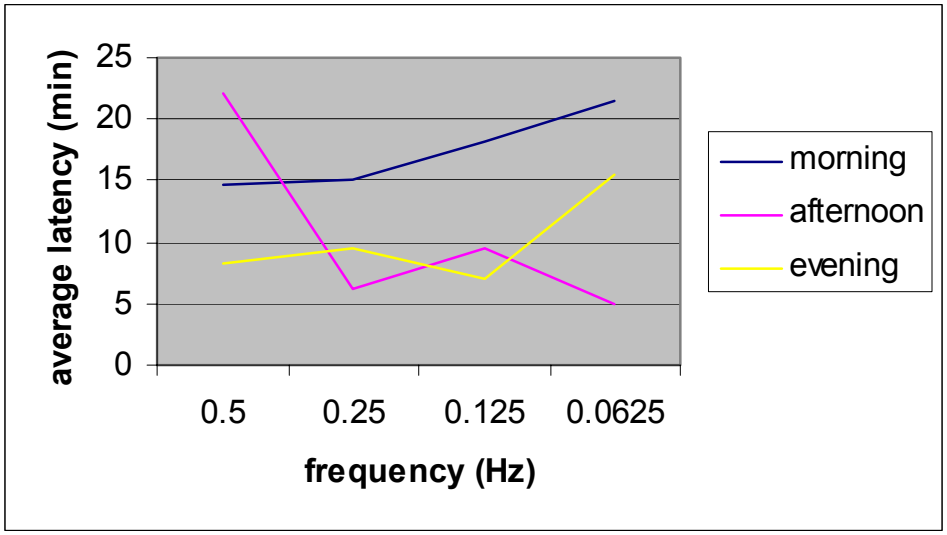


Figure 5.6 Orbcomm average latency.

Comparison of Orbcomm and Iridium testing shows a large difference between system average latency with the same size of data transmission, frequency and time of day. The Iridium system, with an overall average of 1465 ms data transmission latency is, on average, 522 times faster than the Orbcomm system average of 12 minutes 45 seconds. The technology used by the Orbcomm system to transfer data is very different from the Iridium system. When the Orbcomm satellite is in view of both the Gateway Earth Station (GES) and the satellite transceiver, a data message is transferred in real-time. Otherwise, data is stored on board the satellite and transferred when the satellite comes into view of the GES. This causes a major delay in Orbcomm data transmission. In the Fredericton, NB area, where testing was performed, the percentage of time that the Orbcomm satellite in view of the transceiver is as high as 99%. The nearest Orbcomm GES used for data relay is located in New York, N.Y., USA. The percentage of time that both the satellite transceiver and the GES are simultaneously in view of an Orbcomm satellite is 70%. This percentage is spread over the whole day following the satellite orbit

dynamics. The Orbcomm system latency is affected by time of day, requiring significantly more time in the morning compared to the afternoon and evening. Less significant “time-of-day” affect was noticed for the Iridium system.

Chapter 6. Conclusions and future work

6.1 Conclusions

In this research we have investigated satellite data communication systems in several different aspects including system cost, data link capacity and data transmission latency. Our research has shown that the Iridium satellite system has potential for use in real-time seismic survey operations. Our testing indicates that Orbcomm is better suited for applications such as e-mail where delays of greater than 12 minutes (on average) are acceptable. Our testing showed that the Iridium system was very reliable with consistently low latencies around 1.46 seconds for both static and dynamic operations.

We have developed a detailed cost model for seismic survey and forest fire operations using satellite data communications along with an XML-based computer implementation of the cost model. Orbcomm satellite communication system cost is about 10 times less than the Iridium system costs. Availability of Orbcomm satellite transceiver OEM boards is an advantage of the Orbomm system that can provide the user an opportunity for customized application integration.

In the satellite survey operation cost models (not counting the helicopter (s) flying costs), satellite communication cost is 19% of the total cost in a high accuracy seismic survey using both Iridium and Orbcomm systems. When the Iridium transceiver is replaced by a radio modem, the communication cost is only 1% of the total survey cost.

These percentages ignore the helicopter (s) flying costs which are the same in both cases. The increase in communication costs is partly offset by the cost of installing repeater stations for the radio modems. It is conceivable that the cost with satellite

communications would be less if the cost of installing repeater stations increased substantially.

The research also explored the possible role of SOAP in communicating satellite system operational costs. A SOAP server can provide convenient access to a client to retrieve satellite geospatial system costs in XML format using a standard Web browser.

6.2 Future work

Our research is a start for deploying satellite communications in real-time seismic survey system operations. There are interesting and challenging future investigations required before satellite data communications can be routinely used in helicopter operations. They include:

1) Integration of Kodiak Office and RGU500 software with satellite data transmission software. The existing Kodiak Office and RGU500 software need to be integrated with satellite data transmission software to provide a platform for field testing satellite communication links.

2) Investigation of automatic hand-off from a satellite system link to a cellular telephone link when cell phone signal strength of sufficient quality is detected. This hand-off can reduce seismic survey costs without losing the advantage of global coverage of satellite links. Automatic hand-off from a satellite link to cell phone and vice versa would be most convenient for the user.

3) Testing of Orbcomm satellite data transmission using a transfer protocol other than SMTP. Our testing shows that the Orbcomm system has a high system latency. Orbcomm also has advantages, however, such as low cost and availability of OEM boards. When real-time transmission is not critical, the Orbcomm system is a good choice as data

carrier. Normally, Orbcomm users receive data through e-mail. There is a possibility that other file transmission protocols such as FTP can be used in Orbcomm data transmission and that the recipient can receive data using an Orbcomm satellite transceiver directly. Motorola is currently exploring a project using FTP via Orbcomm. The project is called "Spatial Utility Portfolio Management". The project provides near real-time solutions of spatial data for information of gas, water and electricity [Motorola, 2000]. An investigation of how Orbcomm handles data transmission using other protocols such as FTP in terms of latency and data error rate would be valuable in satellite system development for other near real-time geospatial operations.

References

[Altova, 2002] Altova, "XMLSpy 5 Product Information", 1 page, 2002, available at http://www.xmlspy.com/products_ide.html.

[Analysis, 2002] Analysis Ltd., "Iridium", 10 pages, Feb 08, 2002, available at http://www.analysys.com/satellite/profiles/iridium.htm.

[Apache, 2001] Apache Software Foundation, "Xerces C++ Parser", 3 pages, 2001, available at http://xml.apache.org/xerces-c/.

[Barr, 1999] Barr, Michael, *Programming Embedded Systems in C and C++,* O'Reilly & Associates, Inc., 1999.

[BEKL, 2000] Box, Don, Ehnebuske, David, Kakivaya, Gopal, Layman, Andrew, Mendelsohn, Noah, Nielsen, Henrik Frystyk, Thatte, Satish and Winer, Dave, "Simple Object Access Protocol (SOAP) 1.1", W3C, May 08, 2000 available at http://www.w3.org/TR/SOAP/.

[Chatenay, 2000] Chatenay, Allen, "Seismic Surveys, Getting Geophysical with GPS", *GPS World*, May 2000, pp 22-30.

[Compass, 1999] CompassRose International Publications, "Introduction to Global Satellite Systems", 6 pages, 1999, available at http://www.compassroseintl.com/pubs/Intro_to_sats.html.

[Eagle, 1998] Eagle Navigation System, Inc., "Kodiak: A Vehicle Guidance and Management System for Seismic Operations Design Document", version 1.2, internal report, Calgary, Alberta, Canada, July 16, 1998, 52 pages.

[Eagle, 2001] Eagle Navigation System, Inc., "Kodiak NS500 RGU Operational Manual", version 2.1.1, internal report, Calgary, Albert, Aug. 2001, 50 pages.

[Fletcher, 1982] Fletcher, J., "An Arithmetic Checksum for Serial Transmissions", IEEE Transactions on Communication, Vol. COM-30, No. 1, pp. 247-252, January, 1982.

[Goldstein, 2000] Tally Goldstein, "Defence set to revive Iridium", FT.com Financial News, available at http://flashcommerce.com/articles/00/12/05/194357.html (click on "source"), December 6, 2000, 1 page.

[Ha, 2001] Ha, Rick, "Preliminary Survey on Satellite Wireless Standards", Faculty of Electrical & Computer Engineering, University of Waterloo, Waterloo, Ontario, Canada, February 9th, 2001, 11 pages.

[Hopkins, 2001] Hopkins, Joe, "Iridium Ownership Revealed", SPACEandTECH Digest, April 09, 2001, 1 page, available at http://www.spaceandtech.com/digest/sd2001-14/sd2001-14-007.shtml.

[IBM, 2002] IBM, "XML Parser for Java", 1 page, 2002, available at http://www.alphaworks.ibm.com/tech/xml4j.

[Iridium, 2002a] Iridium Satellite LLC, "Direct Internet Data User's Guide", Rev. 4, February 1, 2002, 38 pages.

[Iridium, 2002b] Iridium Satellite LLC, "Dial-Up Data User's Guide", Rev. 4, February 1, 2002, 67 pages.

[Iridium, 2002c] Iridium Satellite LLC, "Mobile Terminated Data User's Guide", Rev. 2, February 1, 2002, 25 pages.

[Iridium, 2002d] Iridium, FQA, "What is Short Burst Messaging?", 1 page, 2002, available at http://www.iridium.com/customer/iri_customer-detail.asp?careid=92.

[Jung, 2002] Jung, Helen, "Teledesic shuts down, dimming a dream", 4 pages, October 14, 2002, available at http://www.startribune.com/stories/535/3357432.html.

[KBMW, 1999] Kirchner, Al, Brickerd, D., Mazur, S. and Williams, B., "Orbcomm Serial Interface Specification", E80050015 Revision F, Orbcomm Global L.P., Dulles, VA, USA, April 20, 1999.

[Klein, 2001] Klein, Ramon de, "Serial Library for C++", 20 pages, 2001, available at http://www.codeproject.com/system/serial.asp.

[Lloyd, 2002] Lloyd Wood, "Lloyd's satellite constellations", http://www.ee.surrey.ac.uk/Personal/L.Wood/constellations/teledesic.html, October 18, 2002.

[McLellan, 2001] McLellan, James F., Schleppe, John B., Huff, Dave and Srajar, Peter, "Mobile Asset Management For Land Exploration", internal working document of Eagle Navigation System Inc., Calgary, Alberta, Canada, June 20, 2001, 9 pages.

[Mills, 1996] Mills, David L., "The Network Time Protocol (NTP) Distribution", 1996, available at ftp://ftp.qnx.com/usr/free/qnx4/tcpip/utils/xntp3-5f.tgz.

[Mills, 1998] Mills, Dave, "Time Synchronization Software", University of Delaware, 1998, available at http://www.eecis.udel.edu/~ntp/software/win9x.html.

[Mills, 2001] Mills, Dave, "NTP Performance Analysis", University of Delaware, available at http://www.eecis.udel.edu/~mills/database/brief/new/new_files/frame.htm 23 pages, June 01, 2001.

[Motorola, 2000] Motorola, "Spatial™ Product Description", 3 pages, 2000, available at http://www.motorola.com/cgiss/utility.shtml.

[MPCS, 2000] Motorola Personal Communications Sector, Satellite Subscriber products Division, "Iridium AT Command Reference", SSP-ISU-CPSW-USER-0005, Version 1.3, February 23, 2000, 61 pages.

[MSAT, 2002] O'Rourke, Neil, personnel E-mail communication, April 01, 2002.

[Nickerson and Shan, 2001] Nickerson, Bradford G., Shan, Ying, McLellan, James, "Demo Presentation Real-Time Wireless Mobile Geospatial Information Access using the Web", GEOIDE Conference, 2001, June 21-22, 2001, 1 page.

[Nickerson and Wu, 2002] Nickerson, Bradford G., Wu, Alex L., "Cost Model of Satellite Systems for Real-Time Helicopter Operations", Technical Report TR02-157, Faculty of Computer Science, University of New Brunswick, August 7, 2002, 26 pages.

[Orbcomm, 2001] Orbcomm, FAQs, "How does the Orbcomm System work", 1 page, 2001, available at http://www.orbcomm.com/faqs.htm#3.

[Peterson, 2000] Peterson, Larry L. and Davie, Bruce S. *Computer Networks: A Systems Approach,* Morgan Kaufmann, 2000, ISBN 1-55860-514-2.

[QNX, 1996] QNX Software Systems Ltd., *QNX Operating System, System Architecture,* Kanata, ON, Canada, 1996, 139 pages.

[QNX, 2002] QNX Software Systems Ltd., "Products and Services", 1 page, 2002, available at http://www.qnx.com/products/index.html.

[Rappaport, 1996] Rappaport, Theodore S., *Wireless Communications Principles and Practice*, Prentice Hall PTR, Upper Saddle River, NJ, USA, 1996.

[Rosner, 1990] Rosner, Bernard, Harvard University, *Fundamentals of Biostatistics,* 3rd Edition, PWS-Kent Publishing Company, Boston, MA, USA, 1990.

[Regents, 1998] The Regents of University of Berkeley, "TCP/IP User Guide", 1998, available at http://alert.udfcd.org/help/tcpip/user_guide.

[Seaman, 1995] Seaman, R.L. and Pence W.D., "FITS Checksum Proposal", August 24, 1995, http://heasarc.gsfc.nasa.gov/docs/heasarc/fits/oldchecksum/checksum.html.

[Shan, 2001] Shan,Ying, "Web Access to Real-Time Wireless Mobile Geospatial Information", Technical Report TR02-140, Faculty of Computer Science, University of New Brunswick, Fredericton, NB, Canada, June 2001, 111 pages.

[Smith, 2001] Smith, Scott J., "Managing your mobile fleet through integrated wireless solutions", www.ceswireless.com/DOWNLO~1/SYSTEMS0.PDF, January 17, 2002, 32 pages.

[SQLData, 2002] SQLData Systems Inc., "SQLData SOAP Server v3.0", 3 pages, 2002, available at http://www.sqldata.com/soap.htm.

[Stevens, 2000] Stevens, W. Richard, *TCP/IP Illustrated Volume 1 The Protocol,* Addison-Wesley Longman Ltd., 2000, ISBN 0-201-63346-9.

[StPo, 2000] Stevens, Perdita and Pooley, Rob, *Using UML Software Engineering with Objects and Components,* Updated Edition (2nd), Addison-Wesley Longman Ltd. and Pearson Education Ltd, London, UK, 2000, ISBN 0-201-64860-1.

[Sweet, 1999] Sweet, Michael R., "Serial port communication programming guide for POSIX operating system", 5th Edition, 2nd Revision, 60 pages, 1999, available at http://www.easysw.com/~mike/serial/serial.html#4_2..

[W3C, 2002] W3Schools, "SOAP Tutorial", 2002, available at http://www.w3schools.com/soap/default.asp.

[Winer, 2002] Winer, Dave, "The leading directory for SOAP 1.1 developer", 2002, available at http://www.soapware.org.

Appendix I. CostParms.dtd

CostParms.dtd is the Document Type Definition file for satellite systems basic cost elements XML file.

```
<?xml encoding="ISO-8859-1"?>
<!-- @version:  -->

<!ELEMENT CostParms (scenario, taxrate, Orbcomm,
Iridium,Internet,Helicopters,Companysupport,Install)>

<!ELEMENT scenario (#PCDATA) >
<!ELEMENT taxrate (#PCDATA) >

<!ELEMENT Orbcomm
(service_rate_o,monthly_cost_o,minute_cost_o,free_minutes_o,cost_per_byt
e_o,messages_per_minute_o,bytes_per_message_o,free_bytes_o,equipment_rat
e_o,weekly_rental_cost_o,equipment_purchase_cost_o,capital_days_o) >
<!ELEMENT service_rate_o (#PCDATA) >
<!ELEMENT monthly_cost_o (#PCDATA) >
<!ELEMENT minute_cost_o (#PCDATA) >
<!ELEMENT free_minutes_o (#PCDATA) >
<!ELEMENT cost_per_byte_o (#PCDATA) >
<!ELEMENT messages_per_minute_o (#PCDATA) >
<!ELEMENT bytes_per_message_o (#PCDATA) >
<!ELEMENT free_bytes_o (#PCDATA) >
<!ELEMENT equipment_rate_o (#PCDATA) >
<!ELEMENT weekly_rental_cost_o (#PCDATA) >
<!ELEMENT equipment_purchase_cost_o (#PCDATA) >
<!ELEMENT capital_days_o (#PCDATA) >

<!ELEMENT Iridium
(service_rate_i,monthly_cost_i,minute_cost_i,free_minutes_i,cost_per_byt
e_i,messages_per_minute_i,bytes_per_message_i,free_bytes_i,equipment_rat
e_i,weekly_rental_cost_i,equipment_purchase_cost_i,capital_days_i) >
<!ELEMENT service_rate_i (#PCDATA) >
<!ELEMENT monthly_cost_i (#PCDATA) >
<!ELEMENT minute_cost_i (#PCDATA) >
<!ELEMENT free_minutes_i (#PCDATA) >
<!ELEMENT cost_per_byte_i (#PCDATA) >
<!ELEMENT messages_per_minute_i (#PCDATA) >
<!ELEMENT bytes_per_message_i (#PCDATA) >
<!ELEMENT free_bytes_i (#PCDATA) >
<!ELEMENT equipment_rate_i (#PCDATA) >
<!ELEMENT weekly_rental_cost_i (#PCDATA) >
<!ELEMENT equipment_purchase_cost_i (#PCDATA) >
<!ELEMENT capital_days_i (#PCDATA) >

<!ELEMENT Internet (internet_monthly_cost) >
<!ELEMENT internet_monthly_cost (#PCDATA) >

<!ELEMENT Helicopters
(hourly_pilot_cost,hourly_fuel_cost,hourly_rental_cost,daily_insurance_c
ost,daily_maintenance_cost,total_helis,flying_days+,flying_hours+) >
<!ELEMENT hourly_pilot_cost (#PCDATA) >
<!ELEMENT hourly_fuel_cost (#PCDATA) >
<!ELEMENT hourly_rental_cost (#PCDATA) >
<!ELEMENT daily_insurance_cost (#PCDATA) >
<!ELEMENT daily_maintenance_cost (#PCDATA) >
```

```
<!ELEMENT total_helis (#PCDATA) >
<!ELEMENT flying_days (#PCDATA) >
<!ELEMENT flying_hours (#PCDATA) >

<!ELEMENT Companysupport
(daily_equipment_cost,total_personals,personal_hourly_cost+,personal_wor
king_days+,personal_working_hours+) >
<!ELEMENT daily_equipment_cost (#PCDATA) >
<!ELEMENT total_personals (#PCDATA) >

<!ELEMENT personal_hourly_cost (#PCDATA) >
<!ATTLIST personal_hourly_cost number (1|2|3|4|5|6|7|8|9|10) "1" >

<!ELEMENT personal_working_days (#PCDATA) >
<!ATTLIST personal_working_days number (1|2|3|4|5|6|7|8|9|10) "1" >

<!ELEMENT personal_working_hours (#PCDATA) >
<!ATTLIST personal_working_hours number (1|2|3|4|5|6|7|8|9|10) "1" >


<!ELEMENT Install
(heli_install_cost,base_install_cost,head_install_cost,orb_activation_co
st,ird_activation_cost,repeater_install_cost,repeater_number,heli_sat_nu
mber,heli_sat_type,base_sat_number,base_sat_type) >
<!ELEMENT heli_install_cost (#PCDATA) >
<!ELEMENT base_install_cost (#PCDATA) >
<!ELEMENT head_install_cost (#PCDATA) >
<!ELEMENT orb_activation_cost (#PCDATA) >
<!ELEMENT ird_activation_cost (#PCDATA) >
<!ELEMENT repeater_install_cost (#PCDATA) >
<!ELEMENT repeater_number (#PCDATA) >
<!ELEMENT heli_sat_number (#PCDATA) >
<!ELEMENT base_sat_number (#PCDATA) >
<!ELEMENT heli_sat_type (#PCDATA) >
<!ELEMENT base_sat_type (#PCDATA) >
```

Appendix II. ProjectCost.dtd

ProjectCost.dtd is the Document Type Definition file for satellite project cost XML file generation.

```
<?xml encoding="ISO-8859-1"?>
<!-- @version:  -->

<!ELEMENT ProjectCost
(Sat_Comm_Cost,Heli_Cost,Com_Support_Cost,Install_Cost) >
<!ATTLIST ProjectCost cost_S CDATA #REQUIRED project_model CDATA
#REQUIRED >

<!ELEMENT Sat_Comm_Cost (Heli_Sat_Cost,Base_Sat_Cost,Head_Sat_Cost) >
<!ATTLIST Sat_Comm_Cost cost_A CDATA #REQUIRED percentage_A CDATA
#REQUIRED >

<!ELEMENT Heli_Sat_Cost (Heli_Sat_Details) >
<!ATTLIST Heli_Sat_Cost cost_A1 CDATA #REQUIRED >

<!ELEMENT Heli_Sat_Details
(service_rate_h?,monthly_cost_h?,minute_cost_h?,free_minutes_h?,cost_per
_byte_h?,messages_per_minute_h?,bytes_per_message_h?,free_bytes_h?,equip
ment_rate_h?,weekly_rental_cost_h?,equipment_purchase_cost_h?,capital_da
ys_h?) >
<!ATTLIST Heli_Sat_Details heli_sat_type (0|1|2|3|9999) "0"
heli_sat_number (0|1|2|10) "0" each_heli_sat_cost CDATA #REQUIRED >

<!ELEMENT service_rate_h (#PCDATA) >
<!ELEMENT monthly_cost_h (#PCDATA) >
<!ELEMENT minute_cost_h (#PCDATA) >
<!ELEMENT free_minutes_h (#PCDATA) >
<!ELEMENT cost_per_byte_h (#PCDATA) >
<!ELEMENT messages_per_minute_h (#PCDATA) >
<!ELEMENT bytes_per_message_h (#PCDATA) >
<!ELEMENT free_bytes_h (#PCDATA) >
<!ELEMENT equipment_rate_h (#PCDATA) >
<!ELEMENT weekly_rental_cost_h (#PCDATA) >
<!ELEMENT equipment_purchase_cost_h (#PCDATA) >
<!ELEMENT capital_days_h (#PCDATA) >

<!ELEMENT Base_Sat_Cost (Base_Sat_Details) >
<!ATTLIST Base_Sat_Cost cost_A2 CDATA #REQUIRED >

<!ELEMENT Base_Sat_Details
(service_rate_b+,monthly_cost_b+,minute_cost_b+,free_minutes_b+,cost_per
_byte_b+,messages_per_minute_b+,bytes_per_message_b+,free_bytes_b+,equip
ment_rate_b+,weekly_rental_cost_b+,equipment_purchase_cost_b+,capital_da
ys_b+) >
<!ATTLIST Base_Sat_Details base_sat_type (0|1|2|3|9999) "0"
base_sat_number (0|1|2) "0" each_base_sat_cost CDATA #REQUIRED >
<!ELEMENT service_rate_b (#PCDATA) >
<!ELEMENT monthly_cost_b (#PCDATA) >
<!ELEMENT minute_cost_b (#PCDATA) >
<!ELEMENT free_minutes_b (#PCDATA) >
<!ELEMENT cost_per_byte_b (#PCDATA) >
<!ELEMENT messages_per_minute_b (#PCDATA) >
<!ELEMENT bytes_per_message_b (#PCDATA) >
<!ELEMENT free_bytes_b (#PCDATA) >
<!ELEMENT equipment_rate_b (#PCDATA) >
<!ELEMENT weekly_rental_cost_b (#PCDATA) >
```

```
<!ELEMENT equipment_purchase_cost_b (#PCDATA) >
<!ELEMENT capital_days_b (#PCDATA) >

<!ELEMENT Head_Sat_Cost (Head_Sat_Details) >
<!ATTLIST Head_Sat_Cost cost_A3 CDATA #REQUIRED >

<!ELEMENT Head_Sat_Details (internet_monthly_cost) >
<!ATTLIST Head_Sat_Details head_sat_type CDATA #REQUIRED >

<!ELEMENT internet_monthly_cost (#PCDATA) >

<!ELEMENT Heli_Cost
(hourly_pilot_cost,hourly_fuel_cost,hourly_rental_cost,daily_insurance_c
ost,daily_maintenance_cost,helicopters) >
<!ATTLIST Heli_Cost cost_B CDATA #REQUIRED percentage_B CDATA #REQUIRED
>

<!ELEMENT hourly_pilot_cost (#PCDATA) >
<!ELEMENT hourly_fuel_cost (#PCDATA) >
<!ELEMENT hourly_rental_cost (#PCDATA) >
<!ELEMENT daily_insurance_cost (#PCDATA) >
<!ELEMENT daily_maintenance_cost (#PCDATA) >

<!ELEMENT helicopters (flying_days+,flying_hours+) >
<!ATTLIST helicopters total_helis (1|2|3|4|5|6|7|8|9|10) "1" >

<!ELEMENT flying_days (#PCDATA) >
<!ELEMENT flying_hours (#PCDATA) >

<!ELEMENT Com_Support_Cost (daily_equipment_cost,personals) >
<!ATTLIST Com_Support_Cost cost_C CDATA #REQUIRED percentage_C CDATA
#REQUIRED >

<!ELEMENT daily_equipment_cost (#PCDATA) >

<!ELEMENT personals
(personal_hourly_cost+,personal_working_days+,personal_working_hours+) >
<!ATTLIST personals total_personals (1|2|3|4|5|6|7|8|9|10) "1" >

<!ELEMENT personal_hourly_cost (#PCDATA) >
<!ELEMENT personal_working_days (#PCDATA) >
<!ELEMENT personal_working_hours (#PCDATA) >

<!ELEMENT Install_Cost
(Heli_Install_Cost,Base_Install_Cost,Head_Install_Cost,Repeater_Install_
Cost?) >
<!ATTLIST Install_Cost cost_D CDATA #REQUIRED percentage_D CDATA
#REQUIRED >

<!ELEMENT Heli_Install_Cost (Heli_Install_Details) >
<!ATTLIST Heli_Install_Cost cost_D1 CDATA #REQUIRED >

<!ELEMENT Heli_Install_Details
(heli_install_cost,heli_sat_activation_cost?) >
<!ATTLIST Heli_Install_Details heli_sat_type (0|1|2|3|9999) "0"
heli_sat_number (0|1|10) "0" >

<!ELEMENT heli_install_cost (#PCDATA) >
<!ELEMENT heli_sat_activation_cost (#PCDATA) >

<!ELEMENT Base_Install_Cost (Base_Install_Details) >
<!ATTLIST Base_Install_Cost cost_D2 CDATA #REQUIRED >
```

```
<!ELEMENT Base_Install_Details
(base_install_cost+,base_sat_activation_cost+) >
<!ATTLIST Base_Install_Details base_sat_type (0|1|2|3|9999) "0"
base_sat_number (0|1|2) "0" >

<!ELEMENT base_install_cost (#PCDATA) >
<!ELEMENT base_sat_activation_cost (#PCDATA) >

<!ELEMENT Head_Install_Cost (Head_Install_Details) >
<!ATTLIST Head_Install_Cost cost_D3 CDATA #REQUIRED >

<!ELEMENT Head_Install_Details (head_install_cost) >
<!ATTLIST Head_Install_Details head_sat_type CDATA #REQUIRED >

<!ELEMENT head_install_cost (#PCDATA) >

<!ELEMENT Repeater_Install_Cost (Repeater_Install_Details) >
<!ATTLIST Repeater_Install_Cost cost_D4 CDATA #REQUIRED >

<!ELEMENT Repeater_Install_Details (repeater_install_cost) >
<!ATTLIST Repeater_Install_Details repeater_number (0|1|2|3) "0" >

<!ELEMENT repeater_install_cost (#PCDATA) >
```

Appendix III. ProjectCost.xml output files

A III. 1 High accuracy seismic survey with satellite systems

This is the projectCost.xml file for high accuracy seismic survey with Iridium and Orbcomm satellite systems. The helicopter has one Iridium transceiver. The base station is installed with one Iridium transceiver and one Orbcomm transceiver.

```
<?xml version="1.0" encoding="iso-8859-1" ?>
  <!DOCTYPE ProjectCost (View Source for full doctype...)>
  <ProjectCost cost_S="230977" project_model="0">
    <Sat_Comm_Cost cost_A="9025.29" percentage_A="3.90744">
      <Heli_Sat_Cost cost_A1="4259.37">
        <Heli_Sat_Details heli_sat_type="1" heli_sat_number="1"
          each_heli_sat_cost="4259.37">
          <service_rate_h>1.6</service_rate_h>
          <monthly_cost_h>19.95</monthly_cost_h>
          <minute_cost_h>0.68</minute_cost_h>
          <free_minutes_h>0</free_minutes_h>
          <cost_per_byte_h>0</cost_per_byte_h>
          <messages_per_minute_h>0</messages_per_minute_h>
          <bytes_per_message_h>0</bytes_per_message_h>
          <free_bytes_h>0</free_bytes_h>
          <equipment_rate_h>1</equipment_rate_h>
          <weekly_rental_cost_h>0</weekly_rental_cost_h>
          <equipment_purchase_cost_h>1440</equipment_purchase_cost_h>
          <capital_days_h>450</capital_days_h>
        </Heli_Sat_Details>
      </Heli_Sat_Cost>
      <Base_Sat_Cost cost_A2="4733.82">
        <Base_Sat_Details base_sat_type="2" base_sat_number="2"
          each_base_sat_cost="474.448 4259.37">
          <service_rate_b>1</service_rate_b>
          <service_rate_b>1.6</service_rate_b>
          <monthly_cost_b>0</monthly_cost_b>
          <monthly_cost_b>19.95</monthly_cost_b>
          <minute_cost_b>0</minute_cost_b>
          <minute_cost_b>0.68</minute_cost_b>
          <free_minutes_b>0</free_minutes_b>
          <free_minutes_b>0</free_minutes_b>
          <cost_per_byte_b>0.003678</cost_per_byte_b>
          <cost_per_byte_b>0</cost_per_byte_b>
          <messages_per_minute_b>0.2</messages_per_minute_b>
          <messages_per_minute_b>0</messages_per_minute_b>
          <bytes_per_message_b>76</bytes_per_message_b>
          <bytes_per_message_b>0</bytes_per_message_b>
          <free_bytes_b>0</free_bytes_b>
          <free_bytes_b>0</free_bytes_b>
          <equipment_rate_b>1.6</equipment_rate_b>
          <equipment_rate_b>1</equipment_rate_b>
          <weekly_rental_cost_b>0</weekly_rental_cost_b>
          <weekly_rental_cost_b>0</weekly_rental_cost_b>
          <equipment_purchase_cost_b>1150</equipment_purchase_cost_b>
          <equipment_purchase_cost_b>1440</equipment_purchase_cost_b>
          <capital_days_b>450</capital_days_b>
          <capital_days_b>450</capital_days_b>
        </Base_Sat_Details>
      </Base_Sat_Cost>
```

```
    <Head_Sat_Cost cost_A3="32.1">
      <Head_Sat_Details head_sat_type="0">
        <internet_monthly_cost>30</internet_monthly_cost>
      </Head_Sat_Details>
    </Head_Sat_Cost>
  </Sat_Comm_Cost>
  <Heli_Cost cost_B="182970" percentage_B="79.2156">
    <hourly_pilot_cost>0</hourly_pilot_cost>
    <hourly_fuel_cost>350</hourly_fuel_cost>
    <hourly_rental_cost>2500</hourly_rental_cost>
    <daily_insurance_cost>0</daily_insurance_cost>
    <daily_maintenance_cost>0</daily_maintenance_cost>
    <helicopters total_helis="1">
      <flying_days>10</flying_days>
      <flying_hours>6</flying_hours>
    </helicopters>
  </Heli_Cost>
  <Com_Support_Cost cost_C="23362" percentage_C="10.1144">
    <daily_equipment_cost>600</daily_equipment_cost>
    <personals total_personals="2">
      <personal_hourly_cost>72.92</personal_hourly_cost>
      <personal_hourly_cost>125</personal_hourly_cost>
      <personal_working_days>10</personal_working_days>
      <personal_working_days>10</personal_working_days>
      <personal_working_hours>8</personal_working_hours>
      <personal_working_hours>8</personal_working_hours>
    </personals>
  </Com_Support_Cost>
  <Install_Cost cost_D="15619.9" percentage_D="6.76251">
    <Heli_Install_Cost cost_D1="2700.68">
      <Heli_Install_Details heli_sat_type="1" heli_sat_number="1">
        <heli_install_cost>2500</heli_install_cost>
        <heli_sat_activation_cost>24</heli_sat_activation_cost>
      </Heli_Install_Details>
    </Heli_Install_Cost>
    <Base_Install_Cost cost_D2="10779.2">
      <Base_Install_Details base_sat_type="2" base_sat_number="2">
        <base_install_cost>5000</base_install_cost>
        <base_sat_activation_cost>50</base_sat_activation_cost>
        <base_sat_activation_cost>24</base_sat_activation_cost>
      </Base_Install_Details>
    </Base_Install_Cost>
    <Head_Install_Cost cost_D3="2140">
      <Head_Install_Details head_sat_type="0">
        <head_install_cost>2000</head_install_cost>
      </Head_Install_Details>
    </Head_Install_Cost>
    <Repeater_Install_Cost cost_D4="0">
      <Repeater_Install_Details repeater_number="0">
        <repeater_install_cost>1600</repeater_install_cost>
      </Repeater_Install_Details>
    </Repeater_Install_Cost>
  </Install_Cost>
</ProjectCost>
```

A III. 2 High accuracy seismic survey with radio modems

This is the ProjectCost.xml file for high accuracy seismic survey with one radio modem in the helicopter, plus one radio modem and one Orbcomm transceiver at the base station.

```
<?xml version="1.0" encoding="iso-8859-1" ?>
  <!DOCTYPE ProjectCost (View Source for full doctype...)>
  <ProjectCost cost_S="225831" project_model="3">
    <Sat_Comm_Cost cost_A="506.548" percentage_A="0.224304">
      <Heli_Sat_Cost cost_A1="0">
        <Heli_Sat_Details heli_sat_type="3" heli_sat_number="0"
          each_heli_sat_cost="0">
          <service_rate_h>0</service_rate_h>
          <monthly_cost_h>0</monthly_cost_h>
          <minute_cost_h>0</minute_cost_h>
          <free_minutes_h>0</free_minutes_h>
          <cost_per_byte_h>0</cost_per_byte_h>
          <messages_per_minute_h>0</messages_per_minute_h>
          <bytes_per_message_h>0</bytes_per_message_h>
          <free_bytes_h>0</free_bytes_h>
          <equipment_rate_h>0</equipment_rate_h>
          <weekly_rental_cost_h>0</weekly_rental_cost_h>
          <equipment_purchase_cost_h>0</equipment_purchase_cost_h>
          <capital_days_h>0</capital_days_h>
        </Heli_Sat_Details>
      </Heli_Sat_Cost>
      <Base_Sat_Cost cost_A2="474.448">
        <Base_Sat_Details base_sat_type="0" base_sat_number="1"
          each_base_sat_cost="474.448">
          <service_rate_b>1</service_rate_b>
          <monthly_cost_b>0</monthly_cost_b>
          <minute_cost_b>0</minute_cost_b>
          <free_minutes_b>0</free_minutes_b>
          <cost_per_byte_b>0.003678</cost_per_byte_b>
          <messages_per_minute_b>0.2</messages_per_minute_b>
          <bytes_per_message_b>76</bytes_per_message_b>
          <free_bytes_b>0</free_bytes_b>
          <equipment_rate_b>1.6</equipment_rate_b>
          <weekly_rental_cost_b>0</weekly_rental_cost_b>
          <equipment_purchase_cost_b>1150</equipment_purchase_cost_b>
          <capital_days_b>450</capital_days_b>
        </Base_Sat_Details>
      </Base_Sat_Cost>
      <Head_Sat_Cost cost_A3="32.1">
        <Head_Sat_Details head_sat_type="0">
          <internet_monthly_cost>30</internet_monthly_cost>
        </Head_Sat_Details>
      </Head_Sat_Cost>
    </Sat_Comm_Cost>
    <Heli_Cost cost_B="182970" percentage_B="81.0208">
      <hourly_pilot_cost>0</hourly_pilot_cost>
      <hourly_fuel_cost>350</hourly_fuel_cost>
      <hourly_rental_cost>2500</hourly_rental_cost>
      <daily_insurance_cost>0</daily_insurance_cost>
      <daily_maintenance_cost>0</daily_maintenance_cost>
      <helicopters total_helis="1">
        <flying_days>10</flying_days>
        <flying_hours>6</flying_hours>
```

```
    </helicopters>
  </Heli_Cost>
  <Com_Support_Cost cost_C="23362" percentage_C="10.3449">
    <daily_equipment_cost>600</daily_equipment_cost>
    <personals total_personals="2">
      <personal_hourly_cost>72.92</personal_hourly_cost>
      <personal_hourly_cost>125</personal_hourly_cost>
      <personal_working_days>10</personal_working_days>
      <personal_working_days>10</personal_working_days>
      <personal_working_hours>8</personal_working_hours>
      <personal_working_hours>8</personal_working_hours>
    </personals>
  </Com_Support_Cost>
  <Install_Cost cost_D="18992.5" percentage_D="8.41005">
    <Heli_Install_Cost cost_D1="2675">
      <Heli_Install_Details heli_sat_type="3" heli_sat_number="0">
        <heli_install_cost>2500</heli_install_cost>
        <heli_sat_activation_cost>0</heli_sat_activation_cost>
      </Heli_Install_Details>
    </Heli_Install_Cost>
    <Base_Install_Cost cost_D2="10753.5">
      <Base_Install_Details base_sat_type="0" base_sat_number="1">
        <base_install_cost>5000</base_install_cost>
        <base_sat_activation_cost>50</base_sat_activation_cost>
      </Base_Install_Details>
    </Base_Install_Cost>
    <Head_Install_Cost cost_D3="2140">
      <Head_Install_Details head_sat_type="0">
        <head_install_cost>2000</head_install_cost>
      </Head_Install_Details>
    </Head_Install_Cost>
    <Repeater_Install_Cost cost_D4="3424">
      <Repeater_Install_Details repeater_number="2">
        <repeater_install_cost>1600</repeater_install_cost>
      </Repeater_Install_Details>
    </Repeater_Install_Cost>
  </Install_Cost>
</ProjectCost>
```

A III. 3 Forest fire operation #1

This is the ProjectCost.xml file for forest fore operation #1. The model has one Iridium system installed at base station and one Orbcomm transceiver for each of the 10 helicopters.

```
<?xml version="1.0" encoding="iso-8859-1" ?>
  <!DOCTYPE ProjectCost (View Source for full doctype...)>
  <ProjectCost cost_S="486046" project_model="1">
    <Sat_Comm_Cost cost_A="7707.49" percentage_A="1.58575">
      <Heli_Sat_Cost cost_A1="3416.02">
        <Heli_Sat_Details heli_sat_type="0" heli_sat_number="10"
          each_heli_sat_cost="474.448">
          <service_rate_h>1</service_rate_h>
          <monthly_cost_h>0</monthly_cost_h>
          <minute_cost_h>0</minute_cost_h>
          <free_minutes_h>0</free_minutes_h>
          <cost_per_byte_h>0.003678</cost_per_byte_h>
          <messages_per_minute_h>0.2</messages_per_minute_h>
          <bytes_per_message_h>76</bytes_per_message_h>
          <free_bytes_h>0</free_bytes_h>
          <equipment_rate_h>1.6</equipment_rate_h>
          <weekly_rental_cost_h>0</weekly_rental_cost_h>
          <equipment_purchase_cost_h>1150</equipment_purchase_cost_h>
          <capital_days_h>450</capital_days_h>
        </Heli_Sat_Details>
      </Heli_Sat_Cost>
      <Base_Sat_Cost cost_A2="4259.37">
        <Base_Sat_Details base_sat_type="1" base_sat_number="1"
          each_base_sat_cost="4259.37">
          <service_rate_b>1.6</service_rate_b>
          <monthly_cost_b>19.95</monthly_cost_b>
          <minute_cost_b>0.68</minute_cost_b>
          <free_minutes_b>0</free_minutes_b>
          <cost_per_byte_b>0</cost_per_byte_b>
          <messages_per_minute_b>0</messages_per_minute_b>
          <bytes_per_message_b>0</bytes_per_message_b>
          <free_bytes_b>0</free_bytes_b>
          <equipment_rate_b>1</equipment_rate_b>
          <weekly_rental_cost_b>0</weekly_rental_cost_b>
          <equipment_purchase_cost_b>1440</equipment_purchase_cost_b>
          <capital_days_b>450</capital_days_b>
        </Base_Sat_Details>
      </Base_Sat_Cost>
      <Head_Sat_Cost cost_A3="32.1">
        <Head_Sat_Details head_sat_type="0">
          <internet_monthly_cost>30</internet_monthly_cost>
        </Head_Sat_Details>
      </Head_Sat_Cost>
    </Sat_Comm_Cost>
    <Heli_Cost cost_B="420176" percentage_B="86.4478">
      <hourly_pilot_cost>0</hourly_pilot_cost>
      <hourly_fuel_cost>114</hourly_fuel_cost>
      <hourly_rental_cost>795</hourly_rental_cost>
      <daily_insurance_cost>0</daily_insurance_cost>
      <daily_maintenance_cost>0</daily_maintenance_cost>
      <helicopters total_helis="10">
        <flying_days>10</flying_days>
        <flying_days>10</flying_days>
```

```
        <flying_days>10</flying_days>
        <flying_days>10</flying_days>
        <flying_days>10</flying_days>
        <flying_days>6</flying_days>
        <flying_days>6</flying_days>
        <flying_days>6</flying_days>
        <flying_days>2</flying_days>
        <flying_days>2</flying_days>
        <flying_hours>6</flying_hours>
        <flying_hours>6</flying_hours>
        <flying_hours>6</flying_hours>
        <flying_hours>6</flying_hours>
        <flying_hours>6</flying_hours>
        <flying_hours>6</flying_hours>
        <flying_hours>6</flying_hours>
        <flying_hours>6</flying_hours>
        <flying_hours>6</flying_hours>
        <flying_hours>6</flying_hours>
      </helicopters>
    </Heli_Cost>
    <Com_Support_Cost cost_C="23362" percentage_C="4.80653">
      <daily_equipment_cost>600</daily_equipment_cost>
      <personals total_personals="2">
        <personal_hourly_cost>72.92</personal_hourly_cost>
        <personal_hourly_cost>125</personal_hourly_cost>
        <personal_working_days>10</personal_working_days>
        <personal_working_days>10</personal_working_days>
        <personal_working_hours>8</personal_working_hours>
        <personal_working_hours>8</personal_working_hours>
      </personals>
    </Com_Support_Cost>
    <Install_Cost cost_D="34800.7" percentage_D="7.15995">
      <Heli_Install_Cost cost_D1="27285">
        <Heli_Install_Details heli_sat_type="0" heli_sat_number="10">
          <heli_install_cost>2500</heli_install_cost>
          <heli_sat_activation_cost>50</heli_sat_activation_cost>
        </Heli_Install_Details>
      </Heli_Install_Cost>
      <Base_Install_Cost cost_D2="5375.68">
        <Base_Install_Details base_sat_type="1" base_sat_number="1">
          <base_install_cost>5000</base_install_cost>
          <base_sat_activation_cost>24</base_sat_activation_cost>
        </Base_Install_Details>
      </Base_Install_Cost>
      <Head_Install_Cost cost_D3="2140">
        <Head_Install_Details head_sat_type="0">
          <head_install_cost>2000</head_install_cost>
        </Head_Install_Details>
      </Head_Install_Cost>
      <Repeater_Install_Cost cost_D4="0">
        <Repeater_Install_Details repeater_number="0">
          <repeater_install_cost>1600</repeater_install_cost>
        </Repeater_Install_Details>
      </Repeater_Install_Cost>
    </Install_Cost>
  </ProjectCost>
```

A III. 4 Forest fire operation #2

The ProjectCost.xml file for forest fire operation #2 shows one Orbcomm transceiver used at the base station and one Orbcomm transceiver installed at each of the 10 helicopters.

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<!DOCTYPE ProjectCost (View Source for full doctype...)>
<ProjectCost cost_S="482289" project_model="2">
  <Sat_Comm_Cost cost_A="3922.57" percentage_A="0.813324">
    <Heli_Sat_Cost cost_A1="3416.02">
      <Heli_Sat_Details heli_sat_type="0" heli_sat_number="10"
        each_heli_sat_cost="474.448">
        <service_rate_h>1</service_rate_h>
        <monthly_cost_h>0</monthly_cost_h>
        <minute_cost_h>0</minute_cost_h>
        <free_minutes_h>0</free_minutes_h>
        <cost_per_byte_h>0.003678</cost_per_byte_h>
        <messages_per_minute_h>0.2</messages_per_minute_h>
        <bytes_per_message_h>76</bytes_per_message_h>
        <free_bytes_h>0</free_bytes_h>
        <equipment_rate_h>1.6</equipment_rate_h>
        <weekly_rental_cost_h>0</weekly_rental_cost_h>
        <equipment_purchase_cost_h>1150</equipment_purchase_cost_h>
        <capital_days_h>450</capital_days_h>
      </Heli_Sat_Details>
    </Heli_Sat_Cost>
    <Base_Sat_Cost cost_A2="474.448">
      <Base_Sat_Details base_sat_type="0" base_sat_number="1"
        each_base_sat_cost="474.448">
        <service_rate_b>1</service_rate_b>
        <monthly_cost_b>0</monthly_cost_b>
        <minute_cost_b>0</minute_cost_b>
        <free_minutes_b>0</free_minutes_b>
        <cost_per_byte_b>0.003678</cost_per_byte_b>
        <messages_per_minute_b>0.2</messages_per_minute_b>
        <bytes_per_message_b>76</bytes_per_message_b>
        <free_bytes_b>0</free_bytes_b>
        <equipment_rate_b>1.6</equipment_rate_b>
        <weekly_rental_cost_b>0</weekly_rental_cost_b>
        <equipment_purchase_cost_b>1150</equipment_purchase_cost_b>
        <capital_days_b>450</capital_days_b>
      </Base_Sat_Details>
    </Base_Sat_Cost>
    <Head_Sat_Cost cost_A3="32.1">
      <Head_Sat_Details head_sat_type="0">
        <internet_monthly_cost>30</internet_monthly_cost>
      </Head_Sat_Details>
    </Head_Sat_Cost>
  </Sat_Comm_Cost>
  <Heli_Cost cost_B="420176" percentage_B="87.1212">
    <hourly_pilot_cost>0</hourly_pilot_cost>
    <hourly_fuel_cost>114</hourly_fuel_cost>
    <hourly_rental_cost>795</hourly_rental_cost>
    <daily_insurance_cost>0</daily_insurance_cost>
    <daily_maintenance_cost>0</daily_maintenance_cost>
    <helicopters total_helis="10">
      <flying_days>10</flying_days>
      <flying_days>10</flying_days>
```

```
        <flying_days>10</flying_days>
        <flying_days>10</flying_days>
        <flying_days>10</flying_days>
        <flying_days>6</flying_days>
        <flying_days>6</flying_days>
        <flying_days>6</flying_days>
        <flying_days>2</flying_days>
        <flying_days>2</flying_days>
        <flying_hours>6</flying_hours>
        <flying_hours>6</flying_hours>
        <flying_hours>6</flying_hours>
        <flying_hours>6</flying_hours>
        <flying_hours>6</flying_hours>
        <flying_hours>6</flying_hours>
        <flying_hours>6</flying_hours>
        <flying_hours>6</flying_hours>
        <flying_hours>6</flying_hours>
        <flying_hours>6</flying_hours>
      </helicopters>
    </Heli_Cost>
    <Com_Support_Cost cost_C="23362" percentage_C="4.84397">
      <daily_equipment_cost>600</daily_equipment_cost>
      <personals total_personals="2">
        <personal_hourly_cost>72.92</personal_hourly_cost>
        <personal_hourly_cost>125</personal_hourly_cost>
        <personal_working_days>10</personal_working_days>
        <personal_working_days>10</personal_working_days>
        <personal_working_hours>8</personal_working_hours>
        <personal_working_hours>8</personal_working_hours>
      </personals>
    </Com_Support_Cost>
    <Install_Cost cost_D="34828.5" percentage_D="7.2215">
      <Heli_Install_Cost cost_D1="27285">
        <Heli_Install_Details heli_sat_type="0" heli_sat_number="10">
          <heli_install_cost>2500</heli_install_cost>
          <heli_sat_activation_cost>50</heli_sat_activation_cost>
        </Heli_Install_Details>
      </Heli_Install_Cost>
      <Base_Install_Cost cost_D2="5403.5">
        <Base_Install_Details base_sat_type="0" base_sat_number="1">
          <base_install_cost>5000</base_install_cost>
          <base_sat_activation_cost>50</base_sat_activation_cost>
        </Base_Install_Details>
      </Base_Install_Cost>
      <Head_Install_Cost cost_D3="2140">
        <Head_Install_Details head_sat_type="0">
          <head_install_cost>2000</head_install_cost>
        </Head_Install_Details>
      </Head_Install_Cost>
      <Repeater_Install_Cost cost_D4="0">
        <Repeater_Install_Details repeater_number="0">
          <repeater_install_cost>1600</repeater_install_cost>
        </Repeater_Install_Details>
      </Repeater_Install_Cost>
    </Install_Cost>
  </ProjectCost>
```

VITA

Candidate's full name: Alex Lemin Wu

Place of birth: Hunan, P.R.China

University attended: Bachelor of Engineering (Metal Material), 1984 – 1988
Central South University of Technology,
Hunan, P.R.China

Selected publications:

Nickerson, Bradford G., Wu, Alex L., "Cost Model of Satellite Systems for Real-Time Helicopter Operations", Technical Report TR02-157, Faculty of Computer Science, University of New Brunswick, August 7, 2002, 26 pages.