

# SAUV II HIGH LEVEL SOFTWARE ARCHITECTURE

**Steven G. Chappell** [chappell@ausi.org](mailto:chappell@ausi.org)  
**Sai S. Mupparapu** [msaisharan@hotmail.com](mailto:msaisharan@hotmail.com)  
**Rick J. Komerska** [komerska@ausi.org](mailto:komerska@ausi.org)  
**D. Richard Blidberg** [blidberg@ausi.org](mailto:blidberg@ausi.org)

**Autonomous Undersea Systems Institute (AUSI)**  
**86 Old Concord Turnpike**  
**Lee, NH 03824**  
[www.ausi.org](http://www.ausi.org)

## Abstract

In January of 2003, the Autonomous Undersea Systems Institute (AUSI) joined with Falmouth Scientific, Inc. (FSI) and Technology Systems Inc. (TSI) to develop the second generation Solar powered Autonomous Underwater Vehicle (SAUV) II. The first generation SAUV was developed by AUSI along with the Institute of Marine Technology Problems (IMTP) [Ageev et. al., 2001] and served as a proof of concept for solar re-powering of an AUV.

The SAUV II development effort has been a team based program, where FSI focused on the vehicle hardware and its associated low level subsystem software, TSI focused on the user interface, and AUSI focused on program management and the development of the onboard high level mission/system software. In this manner, a diverse set of talents was brought together to realize the SAUV II design.

This paper will describe what we have named the SAUV "production" architecture, which has evolved during the past two and a half years. It will describe the various software modules and identify the functional components of the system that are controlled by the high level vehicle software. The paper will also describe the insights that resulted from in water testing and the impact that those insights have had on the overall system architecture.

## 1. Introduction

AUSI commenced work on the SAUV II (hereafter SAUV) system in January 2003 with a startup meeting between the three main players in the effort. Those players and the portion of the overall effort they were responsible for worked out to be the following:

**Technology Systems Inc.** (TSI) would design and implement the top side user interface to the SAUV system. This would be a standard Windows laptop running a special application called the Mission Planner. As its name indicates, it would be used to assist an operator in off-line

planning of SAUV missions. It would also run software for communicating over a variety of links (Ethernet, RF modem, acoustic modem, and satellite modem) with one or more SAUVs.

The **Autonomous Underwater Systems Institute** (AUSI) would implement the high level onboard software and manage the overall project. Previous success with layered control architectures [Blidberg and Chappell, 1986] prompted us to break the onboard SAUV control particulars into two layers. AUSI would build a new high level vehicle software suite based on lessons learned with the Experimental Autonomous Vehicles (EAVE).

**Falmouth Scientific, Inc.** (FSI) would implement the SAUV physical plant: all subsystems and the hull. This includes the low level microprocessors (with software), electronics, sensors, effectors, and the vehicle body. They would use information and findings that resulted from AUSI's work with IMTP during the initial SAUV I development effort.

System development continued through the spring of 2003 and by summer the first integration tests were carried out. Further development and testing continued through the fall and improvements were affected during the winter. A 22 hour mission was run in April of 2004 to test longevity issues [Mupparapu and Chappell, 2004]. In June of that year, a two week testing exercise to gain remote and long term operational experience with the SAUV system was completed at the Rensselaer Polytechnic Institute (RPI) Darrin Fresh Water Institute (DFWI) in Bolton Landing on Lake George, NY, [Komerska et. al., 2004]. In September, a circa 12 hour scientific mission to collect dissolved oxygen data in Greenwich Bay (RI) was run for the Environmental Protection Agency [Jalbert, September 2004][Crimmins et. al., June 2005]. In October 2004, we returned to DFWI for another week of operational testing [Jalbert, November 2004], this time stressing underwater networking ideas. The most recent operations were conducted in June at AUVFest 2005 in Keyport, WA [Jalbert, 2005] to further test

underwater networking, multiple vehicle cooperation, and Benthos/WHOI acoustic modem interoperability. To date, FSI has built five SAUVs; four are owned by AUSI, and one is owned by the Naval Undersea Warfare Center (NUWC) at Newport, RI.

This paper focuses on AUSI's component in the SAUV; the high level vehicle software. It will build on the overall description found in Jalbert et. al. [2003]. Section 2 describes this level in some detail. Section 3 discusses the connectivity issues associated with the SAUV. Section 4 outlines some of the more recent SAUV development efforts. Section 5 offers conclusions resulting from the past two and a half years of work. Section 6 relates future work.

## 2. Solar AUV High Level System

### 2.1 Hardware

The SAUV high level system runs on a three board PC-104 stack. The CPU board is a Tri-M Systems ([www.tri-m.com](http://www.tri-m.com)) MZ104+, which provides a 100MHz ZF86 CPU, 32Mbyte RAM, a 192 Mbyte M-Systems Disk on Chip (DoC), two Ethernet ports, plus the standard interfaces (keyboard, mouse, 2 serial ports, 2 USB ports, etc). Through Tri-M, we have added a Connect Tech Xtreme/104 8 port RS-232 board for communication with the various SAUV subsystems. At the top of the stack is a Tri-M Utility board, which provides for standard interface connectors, as well as the battery for the real time clock. This system was utilized as it was already in the FSI inventory (the result of a previous low power PC-104 acquisition effort).

Connection to the PC-104 can be via a directly connected keyboard/mouse/monitor when the SAUV pressure tube is open. This is obviously not the usual situation, nor is it the most flexible because it limits the operator's onboard presence to a single login job. When the pressure tube is sealed, SAUV connectivity for development purposes is achieved over a special Ethernet cable leading to the MZ104+'s first Ethernet port (eth0). Installing the SAUV on a LAN in this manner (full Internet protocol (IP) capability) provides for multiple ssh logins, [s]ftp sessions, SAUV website access, and Network Neighborhood access for Windows machines on the same LAN. The idea is to use other machines' keyboards and screens to work remotely on the SAUV platform.

While the SAUV is underway on a deployment and a connection to it is needed, that is achieved through one or more of its three modems: a FreeWave ([www.freewave.com](http://www.freewave.com)) radio frequency (RF) modem, a Benthos ([www.benthos.com](http://www.benthos.com)) acoustic modem, or an Iridium ([www.iridium.com](http://www.iridium.com)) satellite modem.

## 2.2 OS and Runtime Environment

The SAUV PC-104 OS is called "LinuxMZ", which is the Tri-M supplied flavor of Linux. It is based on the Slackware 7.0 distribution with the kernel updated to revision number 2.4.18 (ca. Slackware 8.1).

As delivered by Tri-M, the LinuxMZ runtime environment provides a necessarily scaled down subset (about 120) of the much larger "complete" Linux utility set. Notable examples from the subset include: `cd`, `date`, `cat`, `cp`, `df`, `find`, `gzip`, `ifconfig`, `killall`, `ls`, `more`, `passwd`, `ping`, `ps`, `rm`, `sleep`, `shutdown`, `sync`, `vi`, etc. In every day use, as well as while developing SAUV system scripts, we found that we "missed" a number of standard Linux runtime utilities in the LinuxMZ installation. We have, therefore, added the following Slackware 7.x era system utilities "back" to our SAUV target system environments: `basename`, `head`, `install`, `loadkeys`, `pwd`, `scp`, `tail`, `top`, `touch`, `unzip`, and `zip`.

The LinuxMZ package as delivered by Tri-M provides the following servers: FTP (file transfer), HTTP (website), NFS (network file system), Samba (Network Neighborhood), SFTP (secure file transfer), ssh (secure shell), and telnet (simple remote login). As a result of our ongoing work, we have identified a set of important but missing services in the LinuxMZ image. We have, therefore, added the following Slackware 7.x executables:

- **dhcpcd** - DHCP client daemon<sup>1</sup>. This allows the SAUV to participate in a DHCP oriented LAN.
- **pppd** - Point to Point Protocol daemon. When not hooked up by Ethernet, connectivity to the SAUV is via PPP over its radio modem. This daemon provides for that service.
- **ntpd** - Network Time Protocol Daemon. Synchronize system clock with other hosts' clocks.

In laboratory bound development and testing situations, the SAUV's PC-104 is connected to the laboratory LAN. The Samba package allows the SAUV's working directory to be mounted by Linux and Windows development systems. The SAUV software is built and initially tested on a Linux development system. Once ready, a "tarball" of the components is made and then copied over the network to the SAUV's PC-104. On the SAUV, the package is unbundled with a custom set of "install" scripts and the components exercised via a number of remote login windows. In the field, with no hard Ethernet link, the same procedure is

---

<sup>1</sup> A daemon is a process that can run unattended in the background, unattached from any terminal.

accomplished over a PPP link through the FreeWave modems, although at a reduced bandwidth.

### 2.3 Applications and Support Utilities

As described to this point, the PC-104 is just another Linux system. Turning it into a SAUV high level control system requires additional software. We have implemented a family of cooperating processes running on top of Linux, which supplies the system with its “SAUV personality”. This family is brought up at system boot up and taken down gracefully at system shutdown. An operator can also boot and gracefully shutdown the family at will (while leaving the OS running) with a special utility. The primary players in this family are a Mission Manager, a Data Manager, a Communications Manager, and sometimes a User Interface Task. They are, in turn, supported by a small set of port handling processes. The overall configuration of these processes is shown in Figure 1.

In this Figure, round shapes denote executing pieces of software. Cornered shapes denote physical boundaries (such as “Topside Laptop” or “SAUV”), major subsystem boundaries (such as “PC-104” or “Navigator”) or files maintained within a computational environment (such as the “Mission Files”). Arrows within a system boundary indicate inter-process communication (UDP in this case) and its directionality; or file reading/writing. Labeled arrows between hosts and/or subsystems indicate the kind of hardware connection between those two systems. Colored arrows denote the type of communication being moved. Dotted lines indicate items that have been added at one time or another to the PC-104 level on an experimental basis, but have not yet become part of the so called “production” grade software suite. The User Task and all communications to and from it are denoted in dashes to indicate that it is a formal member of the family, but it is not always present in the system.

The TSI component of the SAUV system is found at the top of Figure 1: a laptop running the Mission Planner and equipped with a Benthos acoustic modem, a FreeWave radio modem, and an Ethernet interface. The rather busy rectangle consuming the rest of the Figure represents the entire SAUV hull and attempts to show the important details of its innards. Three possible communication links are shown between these two components: acoustic, radio, and Ethernet. The Ethernet is used only when possible, i.e. during development, or when the vehicle happens to be close enough for a physical connection. The RF connection is also only used when necessary. This intermittent usage is denoted in the diagram by the dashed lines.

The FSI component of the SAUV is represented in the Figure in the lower left of the SAUV rectangle by the three subsystems: Energy Manger, Propulsion/Motion, and the

Navigator. While the other subsystems in the SAUV body are for the most part already completed “black boxes”, these three subsystems were designed and built by FSI from the ground up.

Focusing on the PC-104 rectangle in the Figure, we see a number of processes (ellipses) and files (yellow rectangles). This is the SAUV process family. The light orange ellipses represent the aforementioned subsystem handling processes. They listen to specific ports, receive data from the specific subsystem on the other end of that port’s connection, package that input up into an internal message format, and send it on to the central data repository for further processing. The orange processes are the various SAUV Managers; each having its own specific area of expertise and appropriate name. SAUV data acquisition and recording is shown with green arrows. Self initiated command execution is shown with blue arrows. The red arrows depict operator directed commands and responses. The purple arrows show the path status reports take on their way up to the Mission Planner. Unspecified communications are shown with black arrows.

The System Event Log files shown on the right represent the event logs kept by the Linux OS as a normal consequence of running. After a number of experiments keeping completely separate event logs, it was found to be simpler to piggyback SAUV process family event logging along with the system event logs. This was accomplished using the Linux `syslog()` facility. These Event Logs are written to by all processes in the family. They are of use mostly to engineers wanting (needing) to closely dissect a mission.

In greater detail, the SAUV family process members are:

**Energy Manager Daemon:** This process collects serial frames arriving from the Energy Manager subsystem, tests for errors, packages them up into internal Energy Manager data packets and sends them on to the Data Manager.

**Navigator Daemon:** This process collects serial frames arriving from the Navigator subsystem, tests for errors, packages them up into internal Navigator data packets and sends them on to the Data Manager.

**Data Manager:** This process receives data packets from the various data collection daemons in the system. Data Manager integrates them all into a single data record. Such records contain a snapshot of the vehicle state at any single time stamp. Currently, a new record is generated each second. The Data Manager keeps the 60 most recent records in memory for quick access. It also writes each new record to the current Data Log file. These log files are tailored for efficient storage until they can be off-loaded by operator command for post mission analysis.

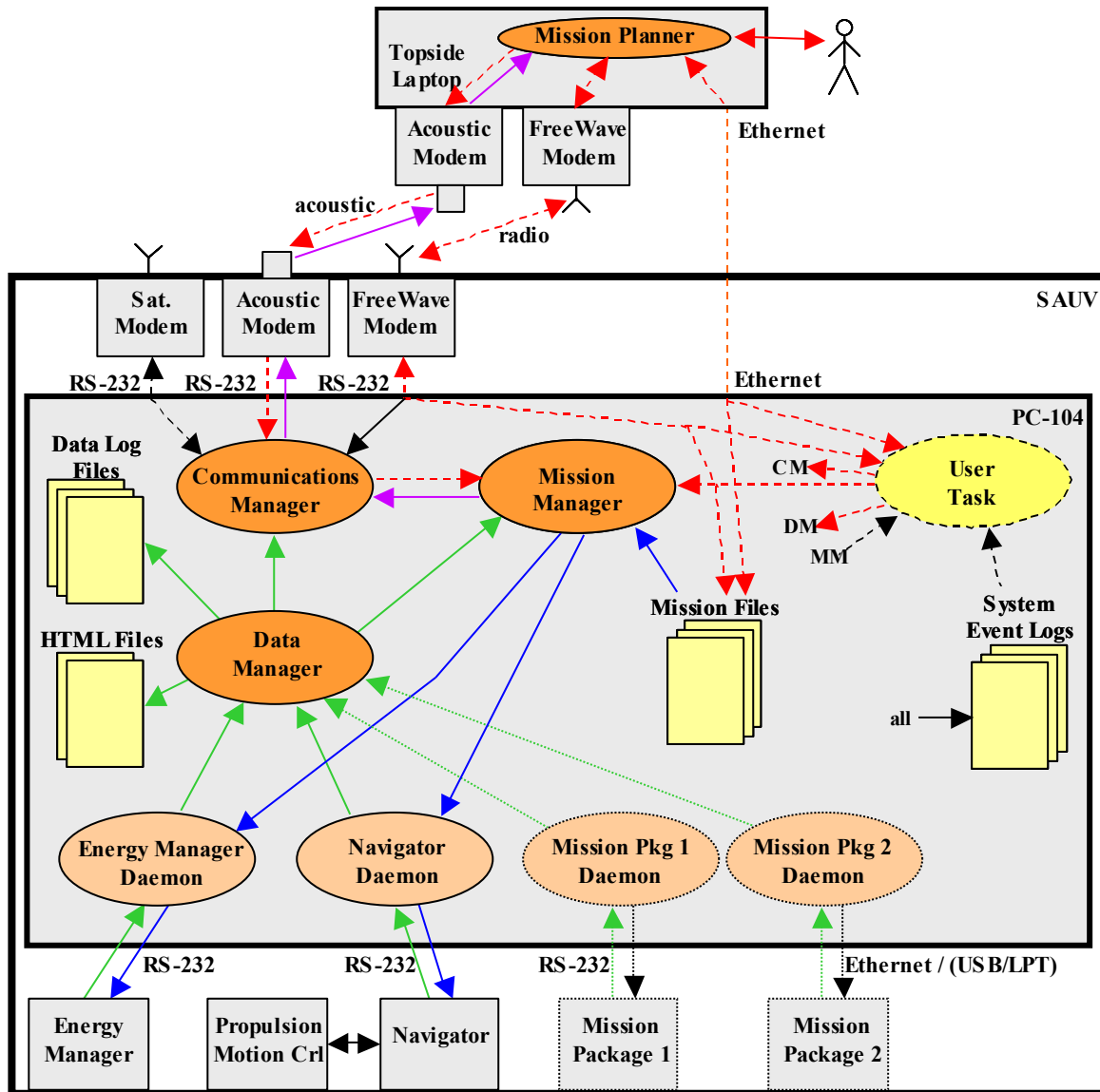


Figure 1. SAUV PC-104 Software Architecture.

The Data Log files are the real products of a SAUV deployment. During the time span from process family boot up all the way through to shutdown of same, the Data Manager collects and records data in a series of compressed files. Rather than write the data to a single file over the entire time interval Data Manager is running, we have found it advantageous to break the data stream into more easily manipulated file "fragments". Each fragment file contains a block of vehicle data in comma separated value (csv) format. Each line in each file is a time stamped comma separated ASCII record of the state of about 70 values collected or computed by the vehicle at that time. This format should be considered a "raw" format, in that it is suitable for additional

post mission processing (i.e. directly with spreadsheet programs) and reorganization into more specific and/or refined data products. In order to conserve vehicle file system space, each file is compressed when closed by the Data Manager.

The Data Manager also sends a copy of the current data record to the other managers in the system so that they have the most recent data set in their possession at all times. Finally, Data Manager writes copies of select data fields from the current record into a pair of HTML files. These are laced into the SAUV's website to provide quick remote access to that select data using only a web browser.

**Mission Manager:** This process implements the vehicle commander. While underway, its principle job is to transpose the high level statements found in a Mission File into lower level commands that can be understood by the SAUV's Navigator subsystem. Missions in such files are expressed as sequences of Common Control Language (CCL) [Komerska, 2001] statements. Those files are generated by the operator using the Mission Planner resident on the Topside Laptop. Once such a file has been generated, the operator FTP's it to the SAUV (see red dashed arrows leading from Mission Planner to Mission Files). Any number of such Mission Files can reside in the SAUV file system ready for use. When commanded by the operator to run a particular mission, the Mission Manager first reads the named Mission File. It then compiles those CCL statements into a set of lower level commands (LLCs) specific to the SAUV's various subsystems. With that completed, it then runs an interpreter to step through the LLC set issuing them to the Navigator and Energy Manager subsystems one after another in order to execute the mission. The blue arrows in the Figure show this progression.

**Communications Manager:** This process is the communications hub for the SAUV. It handles all the communication packets that go in and out of the system except for the FTP and remote login packets. (That sort of communication is handled by the OS.) Essentially the Communications Manager is the gateway between the Mission Manager and the serial ports. A packet that comes in through a serial interface (acoustic, for example) is first picked up by a dedicated handler processed to check packet integrity (checksum). Then it is passed to the Communications Manager to be further processed. Depending on its operation mode, Communications Manager makes specific library calls in order to affect various kinds of communications strategies. Communications Manager can be operated in any of the following four modes:

- None or raw
- Frames: CR, CRLF, short frames (< 256 bytes), and long frames (< 65 Kbytes)
- AUSNet (Autonomous Undersea Systems Network) [Benton et. al., 2003]
- COFSNet (Controlled Flooding for Small Networks) [Mupparapu et. al., 2005]

The first mode is where there is no protocol or data wrapping. Data packets are simply transmitted onto and received from the physical media. This mode might be used, for example, when the SAUV must directly control its modem. The second mode adds a frame wrapper to data packets so that receivers on the other end of the link can at least test for packet integrity. We have implemented four varieties of framing library calls, which range from simple carriage return terminated frames to more formal arrangements using special leading bytes, length specifiers,

special trailing bytes, and checksums. The third and the fourth modes use the routing protocols AUSNet and COFSNet respectively. These routing protocols add fields to the data packets so as to provide an ad-hoc networking feature on the communications media being used.

When in COFSNet mode, the Communications Manager can also provide an additional communication feature called the "gateway functionality". In this mode, Communications Manager routes packets between the FreeWave link and the acoustic link based on the Group ID of the destination node. A SUAV performing this function could then serve as a gateway node between acoustic communications beneath the surface and radio communications above it. This feature is currently under research and development.

**User Task:** This process is *not* routinely launched with the other processes. Under normal circumstances, the above process family is sufficient to run the vehicle. All operator control is affected from the Mission Planner running in the topside computer over one or both of the acoustic and the radio links. If neither of those links is functioning, the SAUV is completely autonomous.

Because the SAUV a relatively new vehicle, however, there are times when an operator's presence onboard is needed. There are two stages or levels to such a remote onboard presence: a remote login to accomplish system level operations (i.e. file system maintenance, file transfer, Data or Event Log viewing, etc.) and a "higher" more SAUV specific level of onboard presence. The User task was implemented in the beginning of development to provide the SAUV level presence. With it, the operator can issue various testing commands to the Navigator and Energy Manager subsystems, test the integrity of Mission Files by test compiling them, perform a crude form of mission simulation based on those Mission Files, change runtime parameter values in the three Manager processes (truncated outgoing arrows from the User Task in the Figure), and start and stop missions. As the Mission Planner gained more and more functionality during its development, it gradually subsumed most of the User Task's functionality. Thus, as development has progressed, the User Task has moved from being an operator interface to more of a technician's interface to the SAUV.

The User Task is accessed by first remotely logging into the SAUV and then launching the application. The login is accomplished via the Ethernet connection, or with a PPP link running over the FreeWave modems. This is shown in the Figure as dashed red arrows. As yet, there is no access to the User Task via the acoustic link. Once User is running, it presents a self documenting suite of commands to the operator. When the operator is done, a "quit" command drops the User Task out of existence. The other processes in

the family do not require the presence of the User Task in order for them to do their jobs.

**Utilities:** The above described process family handles the business of actually running the SAUV. During the design, implementation, and evolution of that family, a number of other programs were written to run onboard the vehicle and support that family. We call these pieces of the SAUV software suite the “utilities”. They help make setting up and running the SAUV a bit easier.

- **details** - record revision details of a SAUV's software installation
- **installLibs** - install SAUV third party libraries
- **installHost** - install SAUV host configuration files and system scripts
- **installDocs** - install SAUV on-line documentation
- **installApps** - install SAUV applications and files
- **procLogs** - process system log files into tarball
- **purgeMessages** - safely purge /var/log/messages
- **sauv** – bring SAUV family up or take it down
- **switchIP** - switch IP address on eth0 to allow for DHCP network connections
- **tunnel** - set up communications tunnel between two serial ports
- **tunnelFreeWave** - manage a data tunnel on FreeWave port.

**Website:** The SAUV also supports a small website. This can be accessed when the SAUV is able to enjoy IP connectivity either through a hard Ethernet or a PPP mediated RF link. Besides the “home” page, there are currently three other pages offered: a web version of the current status report, a special status report which features any extra mission sensors that may be on board (i.e. dissolved oxygen), and a set of SAUV on-line documentation.

Why have a website on something like an AUV at all? Initially, it was implemented in a rudimentary manner (single identifying home page) simply because LinuxMZ came with the server and it was easy to do. Gradually, however, specific uses for it began to show themselves. During our October 2004 Lake George testing, for example, the SAUV website became a useful mechanism by which the vehicle could post a short history of its dissolved oxygen readings. Although only accessible when the SAUV was on the surface, it turned out to be an advantageous feature in the SAUV's connectivity bag of tricks. Further, if the proper routing software could be activated in the topside laptop during such an access, then it would be possible for the SAUV itself to have a presence on the Internet. The on-line documentation is the most recent addition to the vehicle

website. In preparation for AUVFest 2005, many new people were being introduced to the vehicle. We found it advantageous to have operating procedures and other information installed on the vehicle itself for easy access. All SAUV website files have been kept small in view of the reduced bandwidth that may be in force when they are viewed.

To this point, we have described what has come to be called the “production” software. When installed and running, it presents a well known capability set. It is the baseline on which additional software can be installed for development testing and experimentation.

**Mission Pkg 1 and 2 (MP1, MP2) Daemons:** These two processes represent additions to the SAUV process family, which we made during a pair of experiments (see section 4). As additional subsystems are added to the SAUV, they are attached to the PC-104 via one or another of the PC-104's extra ports. MP1 shows the addition of a serial port handling daemon. MP2 shows the addition of a handler listening to a subsystem attached to the one of the PC-104's Ethernet ports. Other systems might eventually be added to the USB interface or the printer parallel port (LPT). This demonstrates how mission packages will be added to the SAUV. This scheme should work for low to medium bandwidth packages. Those that generate enormous amounts of data may require special treatment when interfacing.

### 3. Connectivity

The SAUV supports a number of different communications media (links) between it and an operator. This leads to a fair amount of confusion concerning what level of communication can be had with the vehicle at any one time. This section describes the various kinds of connections that can be made with the SAUV. They are arranged in order of decreasing bandwidth:

**Ethernet** (Cat5 cable): In the lab, or when very close at hand in the field, a special Cat5 cable can be attached to the nose of the SAUV. This is auto-sensed and set at 10 or 100 Mbits per second when the cable is attached. This medium provides full IP connectivity at very good bandwidth, but the SAUV must obviously be hard connected. This is the preferred connection for debugging and exchanging files with the vehicle. Full IP connectivity allows for simultaneous communications using many protocols:

- Mission Planner parameter changing commands to various SAUV applications
- FTP for exchanging files with the vehicle (software installation, Mission File on loading, Data Log file offloading)
- HTTP for browser access to the vehicle's website

- Multiple remote logins to the vehicle for close and varied monitoring and control
- Various ICMP commands such as `ping` and `traceroute`

**Radio Modem (FreeWave):** This medium is good for line of sight above the water at about 1 Kbyte per second. The following connection types over RF are supported:

- PPP. Via the Mission Planner, the operator can initiate the establishment of a PPP link with the SAUV over the FreeWave modems. This provides the same full IP connectivity that is enjoyed over the Cat5 cable (vehicle website, FTP file transfers, logins, etc), but at a much lower bandwidth. Thus, the same multiple simultaneous communications are possible, but they will be forced to compete for the limited bandwidth. Other annoyances accompany PPP use. There is additional bandwidth consuming overhead associated with a PPP connection. Also, being point to point, it operates at the exclusion of any other nodes that might want to communicate with the target vehicle over the same medium. The RF link breaks when the SAUV's RF antenna submerges and that drops the PPP connection and that abruptly drops all communication activity on that link. Since both sides of the connection cannot directly sense this, they must run timers on PPP connection activity. When no transmissions have been sensed on its end for a time interval, the SAUV side takes steps to formally clear the connection away and make ready for a Mission Planner side initiation of the next one. This timer is currently set to two minutes in order to minimize the operator wait time after an unexpected PPP connection drop (i.e., a wave laps over the RF antenna). This means, however, that the operator must be sure to use the link (type something) at least once every 1 minute 59 seconds, or the link will *automatically* be dropped.
- Raw serial login. If the overhead associated with PPP is undesired, or only a single login is needed, then Hyperterminal can be used to establish an isolated login. That single login will have all the RF bandwidth to itself. This use of the RF link is also exclusive; other nodes attempting to communicate with the same SAUV will only generate disruption of the connection.
- Gateway. There is a special case involved when the SAUV is actively running a mission. In that state, the Communications Manager "connects" the SAUV's RF and acoustic modems together to provide a so-called gateway functionality. Traffic arriving at either modem is routed to the other modem and transmitted. While this special mode is

in operation, the FreeWaves may not be used for anything else (no HTTP, FTP, remote login, etc).

**Acoustic Modem (Benthos):** Good for up to 2 Km (sometimes more) through the water at about 800 bytes per second. No IP connectivity is currently available via this link. There is, however, a small set of commands that can be sent to the SAUV via the Mission Planner over the Acoustic Modem. These are initiated by clicking one of five display buttons in the Mission Planner display. The commands are:

- Send status report. A status report containing current values is built and sent back to the operator. This implements status polling of the SAUV.
- Stop. The current mission is stopped. Applications level software continues to collect data and run in an "idle" mode. The vehicle is awaiting further commands to run missions.
- Run mission. A name is collected from the operator and sent to the vehicle, which then attempts to run that mission file. That file must already be onboard the vehicle.
- Abort. This means shut the applications level software down and drift. Linux OS remains active to accept remote logins.
- Send dissolved oxygen oriented status report.

The Acoustic link can be operated in a number of modes. All of the above commands can be sent in any mode. For successful communication, both sides of the connection must operate in the same mode:

- Raw serial. In this mode, only the command and response data is transmitted. No packet framing is used at all, thus, no error checking is performed.
- Framed serial. In this mode, command and response packets are enclosed in one of up to four kinds of frames. Both sides of the link must be using the same framing scheme. This allows for beginning and end of packet and checksum error detection.
- COFSNet. In this mode, command and response packets are framed and then "header-ed" with enough information to implement the protocol. It specifically provides for multi-hop communication via packet sequence numbering and time-to-live hop counting.
- AUSNet. In this mode, command and response packets are framed and then "header-ed" with enough information to implement the protocol. It specifically provides for Dynamic Source Routing (DSR) in order to enable submerged ad-hoc networking.

#### 4. Recent Development Efforts

Using multiple SAUVs at the recent AUVFest in Keyport WA, two experiments that exercised our early decisions concerning the PC-104 architecture were undertaken. The simpler of the two was to demonstrate the beginnings of interoperability between a WHOI Micro-Modem and a modified Benthos ATM 885 Modem. A much more involved experiment involved installing a special runtime environment (Distributed Control Environment, or DICE [Duarte et. al., 2005]) on the SAUV and implementing a number of new processes according to specifications required by that new environment.

**Modem Interoperability:** The objective of the Benthos/WHOI modem experiment was to specifically demonstrate a SAUV reacting properly to a status request issued from a WHOI Micro-Modem external to the SAUV. A proper response involved the vehicle modem acoustically picking up the request, communicating that request to the vehicle PC-104, the vehicle software building the proper Compact Control Language [Stokey, 2005] response (a REMUS MDAT\_STATE structure), sending same to the vehicle modem, and, finally, that modem acoustically transmitting that structure to the originating WHOI modem. Transactions of this nature were to be demonstrated in two ways: through a newly installed vehicle side WHOI Micro-Modem and through a Benthos ATM 885 modem modified to understand a subset of the WHOI Micro-Modem command suite.

All Micro-Modem applications must use NMEA 0183 messaging [WHOI, 2004]. Therefore, in order to affect a proper data request response, the SAUV software was augmented to be able to handle four of the 44 NMEA strings that make up the Micro-Modem command suite. In order to minimize development time, the payload of the outgoing message contained an already designed structure: the REMUS class vehicle basic status information.

SAUV number 2 was selected for this experiment. Benthos generated and downloaded new firmware to the vehicle's standard Benthos modem and that modem's transducer was replaced with one suitable for the WHOI Micro-Modem acoustic frequency. WHOI provided and assisted in the mounting of a WHOI Micro-Modem on the vehicle. This new modem was connected to a previously unused SAUV PC-104 serial port.

In order to "speak" with these new modems, AUSI produced a set of "replacement" software, which would be swapped into use on top of the SAUV 2's production software for the modem experiments. That new set consisted of:

- **mmHandler:** new handler process to read the `<cr><lf>` frames arriving at the new serial port
- **whoiman:** new manager process to parse and generate the WHOI Micro-Modem command and response strings
- **mmDataMan:** new Data Manager to send whoiman process vehicle data records
- **mmSauv:** special sauv up/down script for this experiment
- **installWhoi:** special installation script for this experiment

Using the `installApps` and `installWhoi` utilities, swapping between the production software and this modem suite took only seconds.

The experiment was a success: a WHOI modem was successfully installed on SAUV 2 and shown to communicate with another WHOI modem which was sending it status requests. The onboard Benthos modem's new firmware was also shown to mimic the WHOI command set to the degree that it could successfully participate in the same sort of status requests sent to the SAUV from the external WHOI modem. From the standpoint of manipulating the WHOI Micro-Modem command strings, the SAUV software was able to use the modified Benthos and the WHOI Micro-Modem interchangeably. Further details may be found in Chappell [2005] and Jalbert [2005].

**Distributed Control Environment:** A much more intensive addition and modification to the SAUV's PC-104 architecture was started in January, 2005. The objective of this research effort was to move the DICE runtime system out of the laboratory and into a fielded AUV; in this case, the SAUV. The idea was, that by employing a behavior-based control architecture, we would be able to add powerful new capabilities to the vehicle: ability to adapt to changing situations such as unanticipated sensor failures, support real-time onboard planning, and creation and execution of new vehicle behaviors, particularly those related to multiple vehicle cooperation. The specifics of the DICE design and integration effort are more fully documented in Duarte et. al. [2005]. Here, we describe only the effects this endeavor had on the existing SAUV production architecture.

The DICE environment offers an automated mechanism for transforming a mission expressed in terms of the second version of the CCL [Komerska, 2005] through a series of steps into running processes. This requires a C/C++ compiler. Whereas the LinuxMZ image is stripped down to fighting weight for operation in the traditional embedded system environment (small scale system resources) it had no compiler to offer. For this DICE experiment, the compiler related files were found on a full blown Slackware 7.x

system, archived into a tarball, and installed in the SAUV file system.

Several pieces of the DICE runtime machinery are implemented in C++. The runtime support libraries for C++ programs are not “standard LinuxMZ equipment”. They, too, had to be determined, archived, and installed in the same manner.

In order that we be able to switch back to the production code set when desired, the following scripts were written added to the SAUV utility set:

- **installComp** - install SAUV C/C++ compiler files
- **installDICE** - install SAUV DICE support software
- **installBeh** - install SAUV DICE behavior files

As with the modem interoperability experiment, these utilities now allow us to switch between the two software suites in a matter of seconds.

Once the DICE environment was running in the SAUV environment, the next step was to implement a subset of the basic behaviors presumed to be needed by an AUV as determined by previous work [Komerska et. al, 1999]. These seven behaviors (Maneuver, Navigate, Communicate, Monitor, Sense, Configure, and Execute Convention) were implemented in the DICE “behavior cell” file format (.bc) and put under the control of the DICE environment. Using the DICE formalism, second generation CCL mission files get compiled/transformed into one or more very high level processes which automatically interface to these seven in order to direct the vehicle to do things. This entire DICE mediated environment was then interfaced to the pre-existing SAUV architecture. The resulting hybrid architecture is diagrammed in Duarte et. al. [2005].

Aside from the impact of an additional 10 or so new processes running in the PC-104 environment, the greatest modification to the existing architecture was the dismantling of the old Mission Manager process and distributing its functionality into the various new behavior processes. Another issue was dealing the difference in inter-process communications used by the two architectures: shared memory by DICE and UDP datagrams by the production architecture. This was handled by implementing a bridge process, which served as the contact gateway between the two architectures.

Three DICE equipped SAUVs were operated together and performed well at AUVFest. This effort has laid a solid foundation within the SAUV to support further development

of multiple AUV cooperative behavior. Further details the tests may be found in Jalbert [2005].

## 5. Conclusions

Although it is quite a bit leaner than the installation found on a “standard” Linux box, the SAUV’s Tri-M LinuxMZ image has proven flexible and powerful. The utilities and service daemons included in the factory distribution are a very good mix. In those few situations where it was found to be lacking in a particular area (missing software or files), they were easy enough to find on a full system and copy over.

The SAUV’s high level of connectivity has proven extremely valuable in the development process. The full network connection in the lab has certainly speeded development by accelerating the edit/download/test cycle. In the field, full IP connectivity over the PPP link has proven immensely valuable. Many times, newly encountered problems were debugged, new software prepared in the control trailer, and downloaded over the link to the vehicle sitting in the water, without having to bring it back to shore.

The original design has worked over the last two and a half years of SAUV development. It has shown itself to be amenable to a number of experimental additions, one of them very intense. It is, however, beginning to show some weaknesses. The process count is escalating quite a bit, which, in turn, is leading to a rather confused inter-process communication arrangement.

While successful, the DICE installation is turning out to really tax certain aspects of the current PC-104’s runtime resources. In particular, installing the compiler, runtime support libraries, and the various DICE utilities has just about doubled the runtime foot print from 25% to nearly 50% of the current 192 Mbyte Disk on Chip. This reduces the amount of file space available for Data Log files; the SAUV’s main product. Additionally, the relatively slow processor (roughly equivalent to a 100 MHz Pentium 1) coupled with the smallish RAM field (32 Mbytes) is turning out to be somewhat slow when compiling a “typical” behavior cell file. That operation can consume minutes of time. Quick solutions to this issue would be to increase the Disk on Chip and the RAM field appropriately. Longer term solutions include looking into faster (but still low power) PC-104 alternatives and work to streamline the DICE packaging scheme.

## 6. Future Work

The Data Manager imposes no structure on vehicle data other than the record format itself and a Data Log being a simple flat sequence of those records. While this arrangement made for quick implementation and a workable

solution for preserving data for post mission analysis, it is not very usable if the SAUV is to answer queries something like “what are the dissolved oxygen values between 15:55:00 and 16:00:00”. We see such queries in the SAUV’s future. Roberts and Konz [2002] describe WANDER, a *non-mobile* system very like the SAUV system (solar recharged, energy aware, data collecting) that makes efficient use of the Berkeley DB database library ([www.sleepycat.com](http://www.sleepycat.com)). The complexity of the data dictionaries maintainable with DB sits somewhere between the very simple flat files we are currently using and that of a full relational database, which would be overkill. Such a database (sorted collections of key/values) does not require a separate server process, only a library of special function calls; a benefit in the confines of an embedded system. WANDER also makes good use of a web based user interface ([www.webmin.com](http://www.webmin.com)), which looks interesting and powerful.

The concept of separate listener/handler processes for each serial port has worked well so far, but as the handler count goes up with each added subsystem, it could become unwieldy. It may be more efficient to implement a single serial port listener that is able to monitor *all* PC-104 serial ports. While the internal complexity of such a process would be more than that of any single handler, the overall resource hit on the somewhat meager PC-104 processor may be a good deal less.

The SAUVs have Iridium satellite modems installed; we need to finish out the software configuration work to make them usable. This will be much like the set up for the FreeWave modems; just directed at a different serial port.

Our operational experiences have shown the benefit of simple trailer based deployment. In a couple of instances, however, we have found the area in the immediate vicinity of the launch ramp to be constrained enough to make pure GPS based maneuvering a touch problematic. We feel it would be a nice touch to implement a small piece of supervisory control for the SAUVs in the form of a joystick like interface in the Mission Planner. The purpose of this would be to provide for operator directed “drive-from” and “drive-to” launch site capability.

The SAUV personality and functionality has been implemented in a “traditional” manner: custom cooperating programs are arranged in a family using custom communications links. This family very definitely runs *on top* of the OS. The Linux environment, however, provides a rich set of runtime support features, which are not being directly used and might be better choices for future work. For example, Linux (and other \*NIX environments) have long had a time-of-the-day service called “cron”. This service allows for the automatic scheduling of arbitrary events in a very flexible manner. Rather than have control of

the vehicle flasher be embedded in the Mission Manager (as it is now), it may be more efficient to write a small independent program that simply turns the flasher on or off and have the very competent `cron` facility activate it at the proper times. This would move flasher control into the domain of the OS, not solely in the SAUV applications family and, thus, providing for flasher control whenever the OS is up, not only when the application family is up. Another improvement would be to move some of the various SAUV family processes to under the control of the Linux `init` process. This is how many OS level daemons are activated and controlled. The `init` process makes use of a configuration file, which supports a powerful syntax for controlling other processes.

In the effort to keep energy consumption down, the original SAUV design excluded the use of 802.11 technology. Now that we’ve seen situations where several SAUVs in close proximity to each other (and the topside laptop) could use some sort of RF borne networking, perhaps the energy budget could be reworked to allow the installation of such a wireless connection.

### Acknowledgements

The authors gratefully acknowledge the support from ONR grants N00014-04-1-0264 and N0014-04-C-0094, as well as NSF grant DMI-032084.

### References

- M. D. Ageev, V. E. Gornak, D. B. Khmelkov, A. Ph. Scherbatyuk, Ju. V. Vaulin, and D. Richard Blidberg (2001). Mission Control System for Solar AUV and Results of the Vehicle Long Time Operation Trials. In *Proceedings of the Twelfth International Symposium on Unmanned Untethered Submersible Technology*, August 2001.
- Charles Benton, James Kenney, Robert Nitzel, D. Richard Blidberg, and Steven G. Chappell (2003). Autonomous Undersea Systems Network (AUSNET) - Protocols to Support Ad Hoc AUV Communications. In *Proceedings of the Thirteenth International Symposium on Unmanned Untethered Submersible Technology*, August, 2003.
- D. Richard Blidberg and Steven G. Chappell (1986). Guidance and Control Architecture for the EAVE Vehicle. *IEEE Journal of Ocean Engineering*, volume 13, number 2, April, 1986, pp 449-461.
- Steven G. Chappell (2005). SAUV II AUVFest 2005 Benthos/WHOI Acoustic Modem Interoperability Report, July 2005. AUSI Technical Report 0507-03. AUSI, 86 Old Concord Turnpike, Lee, NH, 03824, [www.ausi.org](http://www.ausi.org).

D. Crimmins, E. Hinchley, M. Chintala, G. Cicchetti, C. Deacutis, D. Blidberg (2005). Use of a Long Endurance Solar Powered Autonomous Underwater Vehicle (SAUV II) to Measure Dissolved Oxygen Concentrations in Greenwich Bay, Rhode Island, USA. In *Proceedings of OCEANS 2005*, June 2005.

Christiane N. Duarte, Gerald R. Martel, Christine Buzzell, Denise Crimmins, Rick Komerska, Sai Mupparapu, Steve Chappell, D. Richard Blidberg, and Robert Nitzel (2005). A Common Control Language to Support Multiple Cooperating AUVs. In *Proceedings of the Fourteenth International Symposium on Unmanned Untethered Submersible Technology*, August 2005.

James Jalbert, John Baker, John Duchesney, Paul Pietryka, William Dalton, D.R. Blidberg, Steven G. Chappell, Robert Nitzel, and Ken Holappa (2003). Solar-Powered Autonomous Underwater Vehicle Development. In *Proceedings of the Thirteenth International Symposium on Unmanned Untethered Submersible Technology*, August, 2003.

James C. Jalbert (2004). SAUV II Greenwich Bay Test Report. AUSI Technical Report 0409-01, September, 2004. AUSI, 86 Old Concord Turnpike, Lee, NH, 03824, [www.ausi.org](http://www.ausi.org).

James C. Jalbert (2004). Multiple AUV-Communications Test Report - Lake George, NY; October 17 - 22, 2004. AUSI Technical Report 0411-01, November, 2004. AUSI, 86 Old Concord Turnpike, Lee, NH, 03824, [www.ausi.org](http://www.ausi.org).

James C. Jalbert (2005). SAUV II AUVFest 2005 Report; June 6 - 17, 2005. AUSI Technical Report 0507-02. AUSI, 86 Old Concord Turnpike, Lee, NH, 03824, [www.ausi.org](http://www.ausi.org).

Rick J. Komerska, Steven G. Chappell, Liang Peng, D. Richard Blidberg (1999). Generic Behaviors as an Interface for Standard AUV Command and Monitoring Language. AUSI Technical Report 9904-01, April, 1999. AUSI, 86 Old Concord Turnpike, Lee, NH, 03824, [www.ausi.org](http://www.ausi.org).

Rick J. Komerska (2001). AUV Common Control Language (CCL) - A Proposed Standard Language for AUV Monitoring and Control. AUSI Technical Report 0410-01. November, 2001. AUSI, 86 Old Concord Turnpike, Lee, NH, 03824, [www.ausi.org](http://www.ausi.org).

Rick J. Komerska, James C. Jalbert, Sai Mupparapu, and Steven G. Chappell (2004). SAUV II Lake George Test Report; June 1 - 11, 2004. AUSI Technical Report 0406-01, June, 2004. AUSI, 86 Old Concord Turnpike, Lee, NH, 03824, [www.ausi.org](http://www.ausi.org).

Rick J. Komerska (2005). AUV Common control Language (CCL) - A Proposed Standard Language for AUV Monitoring & Control, version 2.6. AUSI Technical Report 0507-01, July, 2005. . AUSI, 86 Old Concord Turnpike, Lee, NH, 03824, [www.ausi.org](http://www.ausi.org).

Sai Mupparapu and Steven G. Chappell (2004). SAUV Long Endurance Test Report; April 18 - 19, 2004. AUSI Technical Report 0404-02. AUSI, 86 Old Concord Turnpike, Lee, NH, 03824, [www.ausi.org](http://www.ausi.org).

Sai Mupparapu, Radim Bartos, and Matt Haag (2005). Performance Evaluation of Ad-Hoc Protocols for Underwater Networks. In *Proceedings of the Fourteenth International Symposium on Unmanned Untethered Submersible Technology*, August 2005.

Steven K. Roberts and Ned Konz (2002). WANDER: A Portable Linux Data-Collection System. *Embedded Linux Journal*, May/June, 2002. pp 14-18. Also: [www.linuxjournal.com/article/5716](http://www.linuxjournal.com/article/5716).

Roger P. Stokey (2005). A Compact Control Language for Autonomous Underwater Vehicles. Woods Hole Oceanographic Institution, Woods Hole, MA 02543. April 11, 2005.

WHOI (2004). Micro-Modem Software Interface Guide, version 2.7. Acoustic Communications Group, Woods Hole Oceanographic Institution, Woods Hole, MA 02543. June 15, 2004.