Australian Government
Department of Defence
Defence Science and
Technology Organisation

An Evaluation of Potential Operating Systems for Autonomous Underwater Vehicles

C. Madden

Maritime Platforms Division

Defence Science and Technology Organisation

DSTO–TN–1194

ABSTRACT

This document explores the Operating System (OS) features that are required for the automation of Unmanned Undersea Vehicles (UUVs), from the perspective of a new system development. Recent improvements in system components and sensors allow for increased vehicle capability; however reliable automated software is required to perform extended missions due to the communications limitations with submerged vehicles. The first step of software development for such systems is selecting an appropriate Operating Systems which allows the system to support the software required to perform real-time sensor analysis, and control its behaviour to achieve the mission within the local environmental conditions. This document defines key features for the evaluation of potential Operating Systems for the UUVs with autonomous or semi-autonomous capabilities. Using these features, it explains why some UUVs use a real-time Operating System with custom designed software, but others use the open source Ubuntu Operating System combined with the Mission Oriented Operating System (MOOS) robotics software package, or metaOS. It also emphasises that reuse of existing software and skills are important to advancing any AUV project more rapidly.

APPROVED FOR PUBLIC RELEASE

Published by

DSTO Defence Science and Technology Organisation
506 Lorimer St,
Fishermans Bend, Victoria 3207, Australia

Telephone: 1300 Defence
Facsimile: (03) 9626 7999

AR No. AR-015-676
February 2013

APPROVED FOR PUBLIC RELEASE

An Evaluation of Potential Operating Systems for Autonomous Underwater Vehicles

Executive Summary

With oceans covering over 70% of the earth's surface, there is an interest in unmanned underwater vehicles (UUVs) that can exploit this environment. Due to environmental limitations, remote control of such vehicles requires the use of a tether, limiting the vehicle's range; however operating underwater vehicles autonomously requires advanced sensors and robust software to achieve its mission with limited communications.

This document identifies key features that can be used to compare a range of Operating Systems (OSs) used to develop Autonomous Underwater Vehicles (AUVs). This study found that many of the critical features are implemented throughout all of the potential OSs considered, with differences occurring in real-time capability, previous AUV usage and its availability in an open source form. Of the OSs evaluated, few were designed to meet hard real-time requirements, as this increases development time, and Ubuntu was the only fully open source OS. This open source nature and the limited number of operational conditions that require real-time reactions has led to Ubuntu being widely used, especially where many academic groups develop their systems, as freely available open source projects. A separate computer or control board can be used to provide hard real-time actuator control with an appropriate OS.

Most AUV specific functionality, such as sensor drivers or sensor analysis tools, are provided using software packages, called here metaOSs. These can often be combined using common data structures and messages to operate together to provide additional combined functionality. Most of the metaOSs evaluated operated across a variety of non-real-time OSs. Of these, the Mission Oriented Operating System (MOOS) is the most focused upon AUV development, whilst many of the others are aimed towards automated ground or aerial vehicles. This focus means MOOS provides a wider range of AUV targeted components, including a simulation mode with hardware in the loop, though it can be combined with other metaOSs. MOOS has a loop based structure which cannot provide hard real-time processing, though this structure is common to most metaOSs.

The selection of an OS and metaOS for use in an AUV is often based upon the skills, experiences and existing software available within the AUV development project. This strategy allows the project to reduce its risks by reusing as many skills and components as possible, and reducing the dedicated training required, especially for verifying real-time vehicle control behaviours. Many AUV research groups therefore use an open source OS like Ubuntu, though QNX is used where the designers deemed that they required real-time motor control. As many AUVs are operated at relatively slow speeds in open waters, such control is often not deemed necessary. As MetaOSs are mostly open source, developers often combine components of them with other software libraries into their own existing software operating on the OS they are most familiar with.

THIS PAGE IS INTENTIONALLY BLANK

Author

Christopher Madden
Maritime Platforms Division

Christopher Madden graduated from the Australian National University in 2003 with a BEng and a BIT. Christopher conducted a Ph.D. at the University of Technology Sydney in video surveillance focusing upon tracking individual people throughout a campus environment. This research in object analysis and tracking was then applied as a part of Kingston University's entry into the 2008 MOD Grand Challenge, a UK based robotics competition aimed at using automated systems to identify potential threats in a combat arena. From 2008 to 2010, Christopher worked at the University of Adelaide investigating projects ranging from large scale video surveillance to the MAGIC 2010 robotics competition. This competition promoted coordinated groups of robots that can explore an area and identify key threats autonomously. He now works in DSTO's Maritime Platforms Division conducting research into increasing the automation and sensor processing capabilities of unmanned vehicles.

THIS PAGE IS INTENTIONALLY BLANK

Contents

Figures

Tables

THIS PAGE IS INTENTIONALLY BLANK

Glossary

API Application Programming Interface

AUV Autonomous Underwater Vehicle

AUVSI Association for Unmanned Vehicle Systems International

CPU Central Processing Unit

DSP Digital Signal Processor

HIL Hardware In the Loop

IDE Integrated Development Environment

metaOS Meta Operating System

MOOS Mission Oriented Operating System

OpenCV Open Computer Vision

OS Operating System

PHINS Photonic Inertial Navigation System

POSIX Portable Operating System Interface for Unix

ROS Robot Operating System

ROV Remotely Operated Vehicle

RTOS Real-Time Operating System

SAUC-E Student Autonomous Underwater Vehicle Challenge - Europe

TCP Transmission Control Protocol

URBI Universal Robot Body Interface

UUV Unmanned Underwater Vehicle

THIS PAGE IS INTENTIONALLY BLANK

1 Introduction

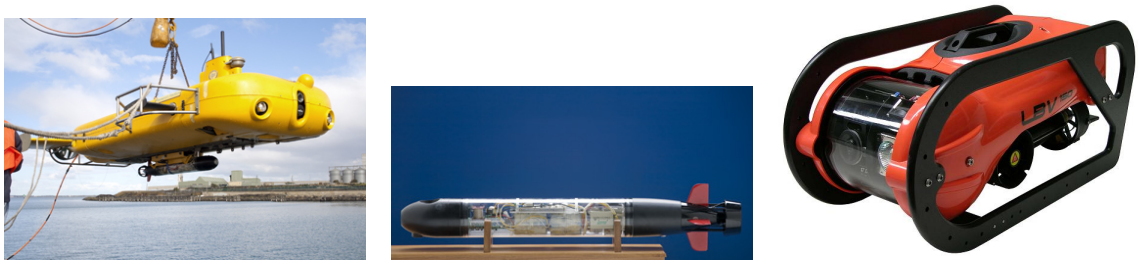
Recent improvements in many critical robot components are leading to significant investment in the development of advanced robotic systems. These improvements are occurring due to many factors including the increased resolution and accuracy of sensors, improved stored energy density and battery life, as well as increases in the computational power that can be embedded within the confined spaces of a robotic system. While these are improving the performance capabilities in many areas of robotics, it is especially important in the challenging underwater environment, where many platforms are still reliant upon an umbilical tether for power and high bandwidth communications. This tether allows for extended duration of platform operation and the high bandwidth data transfer to allow remote control operation; however it limits the range of the vehicle to the tether length, changes the vehicle dynamics and increases the possibility of the platform becoming snagged. Exploitation of this difficult environment is increasing from commercial, academic and military interests as they attempt to create Unmanned Undersea Vehicles (UUVs) that can be utilised to access more of the ocean environment.

The US Navy's UUV Master Plan [1] defines a UUV as a self-propelled submersible vehicle whose operation is either fully autonomous (pre-programmed or real-time adaptive mission control) or under minimal supervisory control and is untethered except, possibly, for data links such as a fiber optic umbilical. In practice some systems also use the umbilical tether to provide power, reducing the need to have on-board batteries; however making the UUV autonomous enough to remove their tether can improve the range and maneuverability of the platform, though not necessarily its endurance. Such an Autonomous Underwater Vehicle (AUV) requires sophisticated software and control systems to achieve its tasks with very limited operator intervention due to the low data rates of underwater communications.

The growing range of publications about UUV platforms and applications demonstrates the increasing interest in this challenging domain. The UUV Master Plan produced by the US Navy [1] demonstrates a significant level of interest from defence organisations in the advancing capabilities of the UUVs as well as investigating their potential uses. Yildiz *et al.* [2] has conducted a recent survey of the UUV literature and describes the growing research interest in the underwater domain as well as providing a good outline of the current challenges and developments in the areas of navigation, communications, control and basic software and components to reduce costs where possible. This has lead to many advances in AUVs as evidenced by the scope of AUVs being utilised and their range of operations. Many AUV systems are designed and operated as research platforms, using open source software components, and reusing existing software. Such research groups, include MIT [3], the SIRIUS project [4] and the Naval Postgraduate School [5, 6]. In addition AUV competitions are increasing in popularity, with examples including the Student Autonomous Underwater Vehicle Challenge - Europe (SAUC-E) [7] and the AUVSI robosub competition [8].

For more than a decade the Defence Science and Technology Organisation (DSTO) has gained significant experience investigating a variety of UUV platforms and behaviours, with Figure 1 showing some of the platforms that DSTO has developed and/or operated. Their sizes vary considerably along with the payloads they can carry. Whilst the 4 metre

long Wayamba platform, with its flatfish design [9], can carry a significant payload, the Mullaya platform is torpedo shaped and less than a metre long with a more limited payload. The lbv150 is a small commercial platform with very flexible motion but a very limited payload. These vehicles are currently all Remotely Operated Vehicles (ROVs) piloted by an operator at the dry end of the tether, though increasing levels of automation are being investigated and incorporated into the vehicle control systems [10].



(i) Wayamba (ii) Mullaya (iii) lbv150

Figure 1: *Examples of DSTO's underwater platforms that can become more automated*

Creating platforms which can operate autonomously in the ocean is difficult because it is a constantly changing environment, which varies depending upon the location and the local conditions. Electromagnetic characteristics, density, salinity, temperature and motion of the water create a number of operational challenges ranging from limited communications bandwidth to limited data for accurate localisation and the power required to adjust or even maintain position. Wireless or satellite communications are widely used in ground or air based unmanned platforms to provide significant communication bandwidth with a predictable latency and a low rate of signal attenuation. Such communications are possible with vessels on the ocean surface; however communications to submerged vehicles are significantly attenuated, even over short distances. The current limitations with underwater optical and acoustic signals, such as their low bandwidth, latency and range limits have led to many research programs being conducted into improving underwater communications [11]. The limitations in terms of available space and power also have a significant impact upon the payload as they necessitate trade-offs between the range of resources available including sensors, computers, propulsion, communications and a variety of other factors that can affect the platform capability and operational duration. In practice most reconfigurable platforms will tend to focus upon the sensor and software packages as many physical components, such as propulsion, are not readily changeable. The latter tend to be very important considerations in the platform design and construction phases and cannot be adjusted significantly during software development.

In order to minimise the operational difficulties of limited on-board AUV capability and external communications within a dynamic ocean environment, AUVs require an Operating System (OS) that will support its ability to sense and react appropriately to its local environment without requiring continual operator intervention. Many OSs have been developed both commercially and by open source communities, which have been applied in the AUV domain. In this report a distinction is made between an OS, which provides the ability for software to access the physical computing hardware and peripheral devices, and a meta Operating System (metaOS), which builds a software based interface

to applications that utilises the OS functionality. A metaOS is therefore a software package which provides a structure with components of functionality to coordinate the utilisation of important system components, such as sensors or communications devices, and processing the available data. They utilise OS features to gain the raw data from sensors, such as reading from serial or network ports, and organise or analyse the raw data into something more useful, such as video data. This is distinct from a software library which might provide functions to analyse data, but not a program structure. MetaOSs follow a similar structure to many of the earlier AUV software architectures explored in [12], though they are designed to be used by a community rather than just for a single platform. An example of the connections between the OS and metaOS on the vehicle and external software is shown in Figure 2. This shows how applications both onboard the UUV and offboard can use metaOS functionality and data structures to communicate and access data or the sensors through the OS. Use of a metaOS reduces the need to develop some aspects of AUV projects, such as communication structures and sensor analysis packages for commonly used hardware, but can also include modules that utilise the available data to perform other useful tasks, such as mission planning or obstacle avoidance behaviours. The software development of an AUV system can therefore be built using an OS with a metaOS executing on top of it so that the project can focus upon developing software for novel areas where appropriate solutions do not already exist.

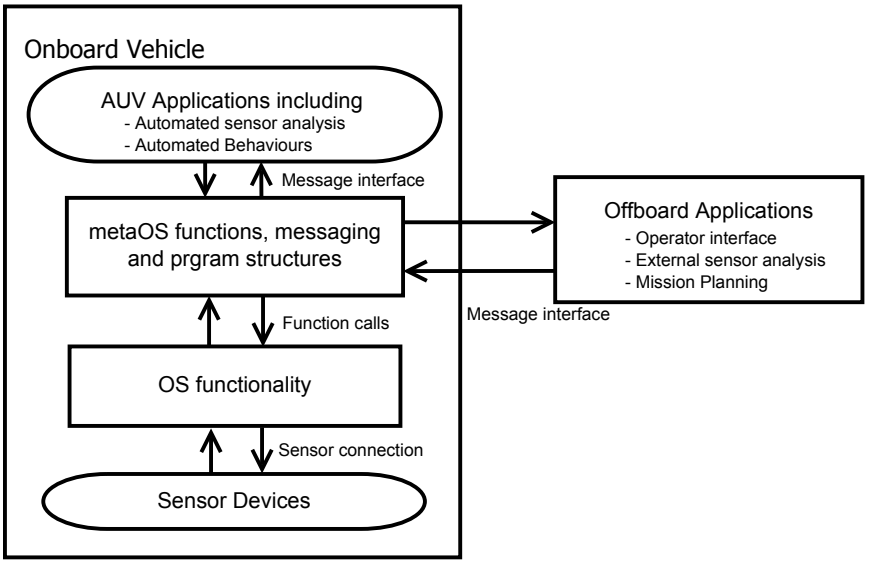


Figure 2: *Typical AUV configuration where OS and metaOS functionality is utilised*

This document is broken into a number of sections to fully explore the topic of evaluating potential OSs. Section 2 develops a set of features that are required for operations by autonomous underwater platforms. This examines the need for fast sensor access and processing capability, which is achieved through adherence to some broad computing standards such as POSIX standards and real-time evaluation. Section 3 describes the OSs that are most commonly used in the robotics area and compares them based upon the important features that are identified. Section 4 describes the metaOS packages that are being developed for robotics and examines how they can combine with the OS in order to best meet the features that are important to AUV operation. Section 5 discusses how the best

candidate OS and metaOS can be utilised for increasing the speed of AUV development, given that many projects begin with existing sets of skills and experience. A subsection is also included as Section 5.1 discussing how this applies to the case study of DSTO's Wayamba UUV. The conclusions about how to utilise the list of features to select an OS are then provided in Section 6.

2 Features to Compare Operating Systems

The challenging nature and increased unpredictability of the underwater operating environment puts a greater emphasis on the requirement for an appropriate OS than many other environments where autonomous platforms operate. The range of these difficulties depends upon vehicle size, as well as its configuration. This is important as system errors or equipment faults can lead to AUV behaviour where the entire platform may founder and not be recoverable. For example in 2005 the AUV "Autosub" was lost under an ice shelf [13]. The incident report states that the cause of the loss is not yet definitely known as the AUV could not be recovered. For an ROV its tether, which is required for fast data transfer and operator control, may become snagged in confined conditions. In addition to the operational impact of such situations, platform loss is expensive and can make identification and correction of the cause very difficult. Fast and predictable reaction times may assist UUVs to adapt to and overcome challenges in the environment, such as strong water currents or localised variations in water density, which may overpower the vehicle's ability to follow its mission plan. In addition communication limitations can make such adaptations difficult to transmit back to a base or support vessels and has already led to the loss of AUVs [14].

Given these challenges, an important consideration for the selection of an appropriate OS is its real-time nature. This can be classified as hard real-time, soft real-time, or non real-time based upon its ability to meet task deadlines, such as accessing system resources. A soft real-time system is one that usually meets the task processing deadline, where hard real-time systems can be deterministically guaranteed to meet the task deadline every time. A non real-time system may consistently overrun task deadlines, or may occur where no deadlines for task execution are determined or evaluated. An important criterion for selection of the real-time nature of the OS for a project is related to the impact upon the system should a task overrun its deadline, either occasionally or consistently. For an AUV based project the results vary depending upon the operational environment and conditions. For example delays in processing sensors or planning tasks when moving slowly will have little effect, though similar delays when travelling at high speed in a cluttered environment may be catastrophic. It should also be noted that the development and verification of performance for hard real-time systems is more time consuming and costly than an equivalent soft real-time system. Final consideration depends upon a variety of factors including required vehicle performance, the likelihood of vehicle loss, and the cost to replace the vehicle.

A number of other considerations are also important for determining an appropriate OS for AUVs. These relate to its support for processing multiple inputs using threading and the availability of code specifically for reuse in projects aimed at developing AUVs. Whilst a real-time focused OS may be important, having the ability to perform scheduling

across multiple cores allows platforms to effectively prioritise the processing of data inputs. This is becoming increasingly important and more commonly utilised as computers move towards multiple processing cores, gaining the ability to process multiple tasks concurrently. With appropriate scheduling and levels of prioritisation, this can ensure important tasks are executed within deadlines, such as calculating actuator behaviour for critical maneuvering. For this reason many OSs provide multi-threaded program execution. In order to best utilise these aspects of the operating system it is important that support exists for the OS to utilise common programming languages and practices. This facilitates programming tasks to be conducted and tested across a wider variety of hardware without the need for personnel to perform significant retraining. Using such languages can also allow for significant reuse of code packages, though this will be most useful where packages have been previously developed aimed specifically for AUVs. It is also useful where device driver code or analysis modules for specific sensors can be reused, as these are time consuming to develop accurately and AUV devices can differ considerably from other robotic domains due to the underwater environmental conditions.

Due to the costs inherent in deploying AUVs, especially in long duration open ocean missions, the ability to design and test the platform within an appropriate simulation environment is very important. Deployment costs can vary significantly depending upon the platform size and availability of nearby testing facilities, though even small scale trials can cost tens of thousands of dollars and can be improved through preliminary simulation to identify and reduce the effect of potentially unforeseen challenges. Developing a simulation environment from scratch is both time consuming and difficult to achieve with a useful level of fidelity; however once this investment has been made, the availability of a simulation environment or framework can allow for significant verification of platform performance to maximise the productivity of costly trails. This can allow for faster development of more accurate and reliable platforms as it can identify possible errors or problems with the system in a range of challenging scenarios before they become an operational problem. Such simulations can be especially useful where a multi-fidelity system allows for fast simulation and analysis of operational parameters, but can also provide slower, more detailed analysis with an environment and physics model to compare the most optimal parameters. Some simulations also incorporate the platform hardware in the simulation loop so that system verification can be improved before full deployment. This is important for complex missions where multiple platforms are in use, as it can allow for significant optimisation and risk reduction before the systems are deployed into the ocean environment. A survey of some existing robotics simulations aimed at providing visualisations of the simulated robots are provided in [15, 16], though few of these systems are targeted to simulate underwater environments.

Utilising an open source OS is a consideration for many research projects as it may provide a greater understanding of the OS and increased flexibility of the low level OS parameters, though perhaps more importantly they don't require the payment of licensing fees. This is even more important where the project is aimed at the development of AUVs for research purposes with very limited budgets. Such open source or open licensed OSs also tend to have dedicated communities that can assist with developer problems, although these are usually web based and take longer to get responses than on-call technical staff that may be available for some commercial OSs. This consideration is less important for organisations which have resource available to cover licensing fees and have existing

experience with development using one of the commercially available OS that can be reused for AUV development.

It is important to note that OSs are generally developed to provide broad support for computing tasks and interrupt driven device drivers for peripherals. Few if any are developed with specific functionality in the operating system kernel for robotic tasks, though some specialised open source OSs do target other areas, such as Scientific Linux [17]. Many groups are developing metaOSs that provide reusable software and architectures for typical applications and devices used in tasks for a given area, such as controlling motors and analysing sensors in robotics. These metaOSs often provide capability and support for specific tasks, allowing developers to reuse many typical components without developing them from scratch, although the metaOSs utilise the underlying OS functionality. Some of the metaOSs are reliant upon a single OS, whilst others are developed as cross-platform systems to enhance portability by enabling their code to be re-compiled for use on many underlying OSs. This distinction between an OS and a metaOS is important as many of the desired features of an AUV system are achieved through a combination of both.

The features considered in this report are not intended to be exhaustive, but as a guide to provide a more methodical comparison of potential OSs and metaOSs that can be utilised in AUV development. A comparative scoring of potential candidates should be conducted by providing relative weighting to each of the criteria for the AUV project in mind. This is important as the weighting can differ depending upon the project goals, the vehicle size, the organisation's familiarity with the potential OS candidates, and the ability to reuse legacy code for the AUV project.

For DSTO this weighting would focus upon the minimisation of resources through the use of existing systems, especially where there is a large user and development community that utilise existing common programming skills and software components or packages. Previous AUV use and existing simulation capability are also important as this aligns with the AUV design focus and increases the usability of existing software packages and device interfaces, such as device driver code. Real-time programming is a lower priority as Wayamba is intended for open ocean environments and the real-time actuator control is being handled by a dedicated control box separate to the sensor processing computer.

A summary of OS and metaOS features to be evaluated for potential AUV projects can be given as:

1. Support for appropriate real-time performance given the nature of the intended AUV deployment.
2. Ability to utilise multiple CPU cores for parallel processing and improved task scheduling.
3. The number of scheduling priority levels to differentiate task priorities.
4. Targets for the required CPU architecture (x86, ARM, etc) of the system hardware.
5. Ability to use common programming languages to reduce the need for programmer retraining.
6. Use common interface standards, such as POSIX threads, to reduce programmer retraining.

7. Existing source code packages able to be reused for typical tasks to reduce development time.

8. Open source or open license with an established user community, as this can reduce costs and allow for greater code reuse.

9. Support and drivers for typical AUV sensors to limit the need for low level driver development

10. Previous use on AUV projects to ensure it is suitably designed for the operational environment.

11. Ability to be used to support design and performance of system simulations, to reduce the development costs and risk of equipment loss.

12. Existing familiarity of research staff with OS and metaOS capabilities which can be reused on the AUV project.

3 Potential AUV Operating Systems

Many commercial organisations and development communities have developed OSs aimed at supporting or performing certain activities, including real-time data processing. These are often generic in nature as they provide the underlying computing capabilities and ability to develop device drivers to interact with peripheral devices, though some are developed for particular tasks. They may focus on real-time behaviour, target embedded processors, or focus on memory or task limited devices. These focussed OSs are often developed as a core OS kernel that provides the base functionality of the full OS of the same type, but add only the necessary extra functionality for the current AUV project. This can reduce the memory size of the OS used and eliminate any functionality that is not required within the AUV.

This section explores many of the common OSs that are used in the robotics area, with a focus upon those in widespread use or providing real-time capability. Whilst this list is not exhaustive, it tries to cover many of the common OSs used in robotics and AUV projects, including Windows XP, QNX Neutrino, Windows CE, Ubuntu, RTLinux, Mac OS X, OSE and VxWorks. Each OS is described first to provide some background of its typical usage and how well it provides the features identified as useful for AUV projects. The candidate OSs are then compared in Table 1, with a discussion detailing which are likely to be the most useful OS candidates.

3.1 Windows XP

Windows XP (WinXP) was released by Microsoft late in 2001 for personal computers, and is based upon the Windows NT kernel [18], though it has been succeeded by the more recent Windows CE and Windows 7 OSs. WinXP is not a Real-Time Operating System (RTOS) and suffers when evaluated as one because it does not include enough levels of threading priority, non-deterministic scheduling decisions, or the possibility of priority inversion,

especially when performing interrupt handling through deferred procedure calls [19]. The evaluation in [19] shows that these limitations can make the execution time difficult to predict under periods of high loading, even if the average execution time may be well within the project limits when average CPU loads are not high. They also suggest that WinXP, or any other non real-time OS, could be coupled with an RTOS that handles all the critical real-time execution, with other tasks being processed on a different machine running the non real-time OS. Such a combination has previously been used in DSTO's UUV Wayamba [9], which ran WinXP in combination with QNX, though combinations with other OSs are also used [20]. AUV simulation software has been developed that will run on many operating systems [21, 22] including WinXP and later windows platforms, though the recent reviews of the systems conclude that there is not an existing simulation that handles multiple environments, is free and easy to use [15, 16], though some of the advanced physics based engines can handle ocean environments.

3.2 Mac OS X

Apple's Mac OS X [23] is beginning to be used in some development communities. It is based on the Berkeley Software Distribution (BSD) Unix variant with a significant level of proprietary API's added. Although it is optimised for Apple specific hardware, it is being increasingly supported by a wide range of peripheral devices. Mac OS X provides preemptive thread scheduling for enhanced CPU sharing, and supports many programming languages as well as typical standards such as POSIX threads [23]. The application specific metaOSs consider Mac OS X to be used widely enough that many of them are set up to compile and execute on it, though no evidence was found of its use in AUV projects. This may be because Mac OS X is not as well supported by sensor and simulation developers.

3.3 Ubuntu

Ubuntu is based upon the Debian GNU/Linux OS distribution maintained by Canonical Ltd. and is free and open source. The base Ubuntu OS is not an RTOS, but can use one of the open source real-time Linux kernels to achieve hard real-time interrupt processing if required [24]. Ubuntu is a common Linux-based distribution with a wide array of community supported functionality obtained through open source reusable packages. It supports most programming languages and Integrated Development Environments (IDEs), which can be written to be POSIX compliant. It also has a large forum based support network of users and developers. They can provide support and advice on development issues, and have developed some reusable software packages. The community can also assist users to get information about known bugs and fixes. Many sensor interfaces are designed to be supported on Unix-style systems, with Ubuntu having been used on robotics and AUV projects, though this is usually facilitated by open source metaOSs, such as MOOS [25], ROS [26], and Player [27]. Its open source nature has also led it to be widely used on university student platforms such as Red Herring [28] for entry into AUV competitions such as SAUC-E [7], and AUVSI [8].

3.4 Windows CE

Windows CE (officially Windows Embedded Compact) has been developed by Microsoft as a commercial compact OS and is a new OS kernel, distinct from Windows XP Embedded, which is NT-based [29]. The kernel is optimised for devices with minimal storage, and is often configured as a closed system that does not allow for end-user extension, though parts are offered to developers in source code form where they may benefit from being adapted to specific hardware. WinCE is an RTOS with a deterministic interrupt latency, with recent versions featuring 256 priority levels that can be applied to individual threads [29]. Compiling for WinCE is supported by Microsoft Visual Studio 2008, though the Embedded Visual C++ tool is becoming the preferred compiler option. Windows CE is developed for embedded devices, and has been used in user interfaces for some sensors, such as the display on Applied Acoustics Engineering's 2650 Easytrak Portable Display for acoustic positioning [30].

3.5 Operating System Embedded

Operating System Embedded (OSE) is a commercial RTOS from Enea aimed at large scale distributed systems [31], but is also used widely in the smart phone sector. It uses a proprietary high level message passing API to allow for software to be developed as modular concurrent processes. It also provides some abstraction from the lower level hardware, making it easier to develop for systems that comprise of multiple Central Processing Units (CPUs) and Digital Signal Processors (DSPs) running the relevant OSE version. OSE has also developed a Soft Kernel Environment which allows OSE processes to run on a Windows, Unix, or Solaris host and can also work with a running real-time system. It allows run-time software upgrades and is dynamically reconfigurable to increase system availability, though no evidence was found of its use in AUV projects.

3.6 Real-Time Linux

Real-Time Linux (RTLinux) is a proprietary OS developed by V. Yodaiken and his colleagues at the New Mexico Institute of Technology [32] and is now owned by Wind River systems. It runs a thin Linux microkernel that separates and prioritises real-time applications and interrupts for fully preemptive processing [32]. Interrupts are handled by the real-time core for deterministic processing, whilst others are passed using pipes or shared memory to the Linux core which is being processed as a lower priority thread. Other commercial and open source real-time Linux options are available [33], though finding an exhaustive list is difficult. One of the main differences between the commercial and open source options is in the way they provide technical support if it is needed. As RTLinux is based upon a similar Unix OS base to Ubuntu, it also has many of the similar functional support features with device drivers and other software generally being supported and available. RTLinux has been previously used in some AUV projects [34].

3.7 VxWorks

VxWorks is a proprietary RTOS developed by Wind River Systems [35] aimed at embedded systems. It is designed to allow for its programming code to be compiled on one computer, but can then be transferred to run on a separate 'target' device. This allows development and simulation of systems from many OSs, with the proprietary kernel providing the optimisation for the targeted hardware without requiring code changes. It otherwise provides many of the same functionalities of other real-time OSs, though driver support for some devices and sensors may not be as wide spread as other OSs. VxWorks has been used on a variety of projects and is widely used in many industries including space robotics [36] and AUV development projects [37].

3.8 QNX Neutrino

QNX Neutrino is a commercial OS aimed at hard real-time systems. It has been developed since 1986 and is aimed to be a modular OS with a small microkernel of core functionality, with extra modules that can be added for additional functionality, including networking tools and graphical interfaces [38]. The microkernel handles largely POSIX compliant message passing between processes and allows for a range of priority based thread processing that allows for fine level of control over the CPU load sharing, which also scales well over multiple cores and devices. It can also be compiled and transferred to a target machine, increasing the portability to a variety of embedded devices. Whilst QNX was not developed specifically for AUV projects, it has been incorporated on some commercial and research AUV platforms [39], though the availability of AUV sensor drivers is not clear.

3.9 Integrity-178B

Green Hills Software has developed their Integrity-178B OS over more than 10 years as a secure POSIX-certified royalty-free hard real-time operating system. They claim it is the most secure and reliable real-time OS as it has been certified by the National Security Agency managed Evaluation Assurance Level (EAL) 6+, indicating high robustness [40], whilst Linux and Windows are EAL4+ certified, indicating a commercially viable level of security. Integrity uses multiple levels of security and proprietary tools for debugging and ensuring the removal of deadlock conditions and other multi-threading problems that might cause the OS to lock under specific circumstances. Their proprietary compilers are also developed to provide a high level of optimisation across a wide range of embedded CPU architectures. Green Hills claim their products are used widely in the avionics industry, including in defence platforms [41].

3.10 Comparison of OS Features

Table 1 compares the ability of prominent OS candidates to meet the key requirements of any AUV project as listed in Section 2, with the number in parentheses indicating its position in the feature list. It should be noted that Table 1 compares the features

that differ between the OS's, with all the OSs meeting features such as the support of multiple CPUs. The key differences occur in their open source availability and whether it is known that they have been previously used on AUVs. The three non real-time candidates evaluated are commonly used for a range of projects, with their main limitations being the lack of deterministic real-time behaviour and that WinXP and Mac OS X are not open source, though they are widely used for general computing tasks. The Xenomai framework can be used as a micro-kernel in conjunction with Ubuntu [24] (or other Linux distributions), to provide hard real-time support for hardware, though it also makes the system more complicated to use. The remaining OS candidates all could deterministically achieve the hard real-time deadlines. The lack of deterministic behaviour of tasks meeting their deadlines can be a serious problem within very dynamic environments, and where the consequences are significant. Such conditions might occur for AUV systems where multiple vehicles are working in close proximity, or where vehicles are operating within complex harbour environments. In many other situations such as open ocean deployment, the consequences of course, location, actuation and sensor errors or delays within the system are less severe. The implementation of hard real-time code that has been verified and optimised to ensure deadlines are reached can take significantly longer to develop and verify than implementations aimed at meeting soft or non real-time requirements where the average CPU use is not excessive. Additionally for many cost sensitive AUV projects, especially low budget research projects into small AUVs, the software licensing costs have led to the use of the freely available Ubuntu OS and other Linux variants, though many commercial AUV platforms utilise the proprietary OSs as their cost is a much smaller fraction of the overall system budget. Consideration of the CPU architecture is also important to ensure that the chosen OS will run on the available hardware for the project, though many of the OSs do support a wide range of CPU targets.

Table 1: *Operating System summary based upon features from Section 2*

OS Name	Real-time Type (1)	Supported Language(5)	POSIX Threads(6)	Open Source (8)	Previous AUV use (10)
WinXP	Non	Many	No	No	Yes [9]
Mac OS X	Non	Many	Yes	No	Unclear
Ubuntu	Non	Many	Yes	Yes	Yes [28]
Windows CE	Hard	Many	No	No	Unclear
OSE	Hard	C/C++	Yes	No	Unclear
RTLinux	Hard	Many	Yes	No	Yes [34]
VxWorks	Hard	Many	Yes	No	Yes [37]
QNX Neutrino	Hard	Many	Yes	No	Yes [39]
Integrity	Hard	Many	Yes	No	Unclear

Most of the OS candidates allow for broad support of multi-threaded applications to fully utilise multiple CPU cores and most of the typical programming languages. This allows the flexibility for software to optimise the CPU cores to perform multiple tasks at the same time, and allows for the significant reuse of existing software components. POSIX compliant threading commands are also widely used, except on the Windows OSs, though some of the OSs extend these to provide additional inter-thread or inter-process communications, such as within QNX. Where most of the OSs can be differentiated is in their open source status and their known use and support for previous AUV projects. The

availability of existing simulation environments for code testing and mission evaluation of AUV systems is also important, with some limited simulation options being available on the non-real-time systems. For organisations that have a legacy of experience and licensing for proprietary OSs, especially where they have been used on previous AUV projects, these factors would be a lesser consideration than the ability to reuse existing code.

OSs provide the capability to utilise the CPU hardware and other peripheral devices. They can be used with a metaOSs for rapid development of effective AUV platforms and simulations, as the metaOS can provide some of the application specific capability, such as communication frameworks and device drivers. Many groups are utilising Ubuntu as an open source OS option where they don't consider real-time to be essential, though QNX and VxWorks are commonly used for AUV projects to provide hard real-time capabilities. Whilst some of these OSs are freely available, the cost of commercial OSs may be less significant than the cost of the time required for the programmers to learn to effectively utilise a new OS, the cost of the physical hardware and sensors for all but the smallest AUVs, or the consideration of whether the OS can be supported on the architecture of the underlying computer hardware. For some projects the software development may be conducted upon available computer systems for multiple devices, so cross-platform support may also be important. Some of this abstraction of code away from a specific OS can also be handled by selecting a metaOS which provides application specific software components that may operate on multiple OSs. For some projects with larger AUVs it may be appropriate to use multiple machines running different OSs optimised for specific tasks, such as real-time motor control, or non real-time sensor processing, reducing the difficulties of obtaining a single OS to support the capability of the entire platform.

4 Potential Robotic Meta-Operating Systems

Many real-time OSs are widely available for generic tasks, but they can be used with application specific software packages to provide typically used capabilities, such as generating robotic motion and sensor processing. These are sometimes termed "metaOS" as they often provide levels of abstraction to access typical low level device drivers, such as sensors and actuators, and are often developed by interested communities as shared, open source resources. These differ from useful code libraries, such as the Open Computer Vision (OpenCV) library [42], which only provide a set of data structures and functions that can be used as needed, as libraries do not specify how the program is structured or functions. The metaOS allows platform developers to focus upon developing new or unique components in their system, which may be incorporated into future versions of the metaOS. Some of the common metaOSs aimed specifically at robotics are described in this section, including the Robot Operating System (ROS), the Mission Oriented Operating Suite (MOOS), Universal Robot Body Interface (URBI) and Microsoft Robotics Developer Studio (RDS), with Table 2 providing a comparison of their capabilities for AUV systems.

4.1 Robot Operating System

Robot Operating System (ROS) is an open source metaOS designed specifically to support code reuse in robotic applications [26]. It is currently supported by Willow Garage with

many partner organisations in the robotics field. ROS operates as a set of peer-to-peer processes, or nodes, that are connected using the ROS communication infrastructure. This specifies the communication format used to send information between the nodes. The nodes provide many of the low-level device control and hardware abstraction layer components, with the inter-process message-passing and package management performed in a peer-to-peer manner similar to many other robotic frameworks. It also provides tools and libraries for developing code across multiple processors or computers. ROS is aimed primarily for Ubuntu and Mac OS X, though many other Linux based platforms are supported by the community. Windows based platforms are yet to be well supported. Code has primarily been developed for Python and C++, though other experimental libraries are also being developed. ROS can also be integrated with some other metaOSs and frameworks, such as the Stage or Gazebo simulation environments.

4.2 Player

Player [43] is an open source metaOS that is widely used in the robotics community. It is the robotic server and device interface of the Player/Stage project [27], and provides a network style interface to connect a range of robot and sensor hardware. Player uses a client/server model, which allows for software development to occur in many programming languages and run on any computer that has a network connection to the message cue of the robot components, such as drivers [44]. Although Player was initially developed around the Pioneer 2 robotic platform, it now supports a wide range of typical robotic hardware. Player is also developed in conjunction with Stage, a 2D scalable robot simulation, and Gazebo, a 3D environment with rigid body physics models. These simulation environments allow for code to be tested without requiring the physical robot and are typically used for ground based robots and although broader use may be possible, physics models for water environments do not seem to be developed yet.

4.3 Mission Oriented Operating Suite

Mission Oriented Operating Suite (MOOS) was initially developed by Paul Newman as a robotic metaOS aimed specifically for AUV applications [25]. It has grown into a community supported system that provides a control loop based cross-platform structure. Its star based topology uses a central database, called MOOSDB, which connects to each application in order to provide the appropriate messages in a human readable string or double precision format. These applications are executed once for each loop cycle in a predefined order. Many platforms can connect to a single MOOSDB in order to share and access data within their own systems, or across platforms. A simple simulation flag allows for the same code to be operationally tested across multiple platforms in a simple Hardware In the Loop (HIL) simulation. In this mode the system generates data either by repeating its expected state, such as the AUV location, or it can use simple environmental models to add noise to the data. More recently the Helm control architecture has been added to create MOOS-IvP. This combination builds upon MOOS to provide behaviour based automated platform control [45]. MOOS-IvP has been used in a variety of projects, including recent operations with multiple platforms [46]. A variety of MOOS-IvP based

AUV platforms have been demonstrated by the Naval Undersea Warfare Center [47, 48]. One of the main limitations of MOOS is its inherent loop-based structure that does not allow for real-time guaranteed behaviour, or operate well with sensor data that is obtained at varying rates.

4.4 Universal Robot Body Interface

Universal Robot Body Interface (URBI), has been developed by Gostai and is now available as an open source cross-platform metaOS aimed at supporting the development of complex systems primarily in the robotics domain [49]. URBI is based on UObject, a distributed C++ component architecture, which utilises the urbiscript, a parallel and event-driven scripting language. This allows it to be developed as a cross-platform system and is embeddable to many CPU boards. URBI is interoperable with ROS and some other C++ based software, and is also usable in the proprietary Webots professional robot simulation. The RTCs (Robot Technology Components) are proprietary and involve many of the control aspects developed for existing robotic platforms, though these seem to be primarily ground based platforms including popular consumer items, such as Lego Mindstorms.

4.5 ERSP Robotics Development Platform

ERSP Robotics Development Platform was created by Evolution Robotics Inc. as a commercially available metaOS for robotic application development [50]. It operates on WinXP as well as some Linux variants, and provides a hardware abstraction layer, a behaviour abstraction layer with over 75 predefined behaviours, and a task execution layer. Applications are developed to interact with these layers as well as the proprietary vision, navigation and interaction modules to achieve the desired goals with the platform. These modules and layers aim to provide a significant portion of the tasks that form the basis of robotics platforms so that developers can focus upon those components that they are most interested in developing.

4.6 Robotics Developer Studio

Microsoft's Robotics Developer Studio (RDS) 2008 R3 is actually a free robotics library rather than a metaOS. As a library it only provides a visual programming interface for tools and a simulation interface, rather than providing the program structure. RDS makes available asynchronous, state driven components [51] that can be programmed through their visual interface. It is aimed at supporting a variety of robots to allow for code and skill transfer between robotics projects. RDS is designed to use the Microsoft Visual Studio .NET framework, which is not generally POSIX compliant, though in practice it is aimed directly for the Windows OS making the code less portable.

4.7 Robotics Simulation Software

Being able to utilise robotics simulation software, or having an inbuilt simulation mechanism, can reduce the field trials required to develop and verify a robotic capability. Stage and Gazebo are two simulation systems that were developed as simulations for Player, but their network interface can be connected with other metaOSs, so long as the data is transferred in an appropriate structure. Stage as a 2D simulation allows for many robots to be simulated and interact at the same time, though it is limited in its applicability as it can not model the inherently 3D environments in which AUVs operate. Gazebo does provide the 3D environment; however it is not as optimised in its processing as other possibilities, such as game engines, and can only simulate a small number of vehicles in a world that uses low quality graphics. They also focus upon ground based robots and may not be as applicable for AUVs. Other robotic simulation environments have also been built utilising interfaces to game engine technology [15, 16]. These simulations provide more realistic graphics and physics, though they can require some development to interface with the robotic code, implement appropriate physics and develop useful operational maps. Examples include USARSim, which was built upon the Unreal Tournament game engine, and has recently been extended to provide simulations of water environments [52, 53], and the Search And Rescue Game Environment (SARGE) [54], which was built using the Unity game engine. It may also be cost effective to use other high grade gaming environments where they provide easy to use artificial intelligence systems for actors within the simulation and where the productivity and quality of terrain generation is high enough to allow for both simulation and system demonstration to interested parties with less technical knowledge.

4.8 MetaOS comparisons

Table 2 compares the most widely used metaOS packages developed for robotics applications and environments against the relevant criteria for AUV development provided in Section 2. The aims of these metaOSs are often quite similar, with many components being developed in the open source community and are somewhat inter-operable, as well as being somewhat OS independent. URBI differs from the other metaOSs in that it does have some aspects that are proprietary in nature and has dedicated technical support behind it. It also utilises Urbiscript, whose event driven nature requires new developers to spend some time becoming familiar with its different structures. ERSP is also significantly different to the other metaOSs evaluated as it is a commercial product and is not open source, though it does provide modules to perform some tasks that can be important in robotics, such as image processing and localisation. RDS is also included here as it does provide many of the similar components to the other metaOSs; however it is implemented as a library, allowing for the easy use of only those components required.

The open source nature of the development of many of these metaOSs as well as the size of the robotics development community has led to collaboration between many of the communities, such that many of the metaOS are interoperable. This can be either via interfaces that support the use of components from some other metaOSs, or by allowing messages to be easily passed between components implemented under other metaOSs. The open source nature also allows for users to adjust and improve some of the components such

Table 2: *meta Operating System summary based upon features from Section 2*

metaOS Name	Supported OS	Supported Language(5)	POSIX Threads(6)	Open Source(8)	AUV use (10)
ROS	Many	Many	Yes	Unclear	Yes
Player	Many	Many	Yes	Yes	Yes
MOOS	C++	Many	Yes	Yes	Yes
URBI	C++	Many	Yes	Unclear	Mostly
ERSP	C++, Python	Win & Linux	No	Unclear	No
Microsoft RDS	C++, C#	Windows	No	No	No

as messages that are sent, or algorithms for sensor analysis or achieving certain robotic behaviours. These improvements can then be incorporated back into the main source code to improve the metaOS with the release of the next version. This interoperability is often also possible with the commercial metaOSs, who also offer technical support for the use of the API of their product.

Whilst many metaOSs are aimed at the robotics area, most of them are developed primarily for ground based platforms, with some incorporating aspects for aerial robotics. MOOS is the only metaOS developed specifically for underwater environments and AUV platforms. The sensors available and the dynamics required for useful simulation of AUVs can differ significantly from those required for ground-based or even airborne robotics. This specialisation for AUVs is inherent in MOOS, which has also developed an easy to use simulation system that can utilise the actual hardware in a HIL simulation. Such simulations can reduce the need for research groups to conduct expensive field trials to test the interfaces between hardware and software or to more quickly verify its accuracy. HIL simulation is also possible with other metaOSs, though some metaOSs are designed to easily interface with virtual simulations and visualisations. For example Player has the 2D Stage and 3D Gazebo simulations, whilst Microsoft RDS has its own simulation environment. If the simulations are intended to do more than verify code execution, then they need to incorporate a physical model to determine the robot's interaction with its external environment, though for AUVs the model will need to incorporate some degree of fluid dynamics. In general running simulations with a more complex environmental model takes longer, so having multiple levels of fidelity may also be useful. It is also possible to perform the simulation of the robot moving using other physics based game engines, such as the Open Dynamics Engine [55] in Gazebo, or the PhysX engine within MarineSim [53]. These simulations tend to be network based, allowing for a simple scalable interface where the simulation can run on separate dedicated hardware; however developing a physics model of the vehicle can take some time regardless of the simulation. It can also be easier to reuse skills in developing the simulation if the simulation operates on the same OS as the AUV. It is crucial that the simulation model behaves in a very similar manner to the real AUV and utilises the same messaging in order to verify that the software will operate effectively when deployed into the field.

When selecting the appropriate metaOS, a key consideration is the ability to reuse any existing code and skills for AUV development, including development experience for specific hardware and OSs. Where a project team has little experience with metaOSs and does not require real-time performance, then MOOS may be a good choice as it focusses

upon the AUV domain specifically. It can operate across the common non real-time OSs and its centralised model can be easily adapted to a vehicle centric approach, although other metaOS such as ROS or Player are more suitable if a decentralized approach is preferred. The centralised loop-based structure of MOOS will also not be able to meet real-time requirements if those are required for the AUV's operation. In practice the implementation of hard real-time AUV performance may be based upon components and even the structure of one of the other open source metaOSs; however the optimisation to meet the real-time deadlines is likely to require significant experience to rewrite the necessary components, reducing the benefits of using the metaOS to reduce development time.

5 Discussion

The previous sections have explored the features of candidate OS and metaOS for a system based primarily upon the operational requirements of AUV design; however unmanned systems research groups generally have existing software and skilled development personnel. These are an important consideration for the selection of the OS and metaOS because they may improve the progression of the project where they can be reused effectively. This is especially important when considering the real-time requirements of the AUV, as this can be a difficult area to effectively implement, optimise and verify without significant experience. Effective hard real-time software is also tied significantly to the hardware used because high data acquisition rates and complex system behaviours will require greater onboard computational power. For these reasons metaOSs do not try to provide real-time functionality, but rather provide more general methods that can give access to sensors and actuators, leaving the development and optimisation of real-time components to those groups who require it.

Where a research group already has significant development experience or software that can provide methods to communicate between devices, it is generally beneficial to re-use that experience. This choice is likely to drive the selection of the OS to be used as well as the metaOS or software architectures, though the open source nature of the metaOSs may allow them to provide software components, such as a driver for a different sensor type. MOOS is the only metaOS that is specifically targeted at providing a software resource for AUV platforms and it can be implemented on a range of OSs. The behaviour based control of the IvP Helm component [45] that operates with MOOS is another area that may provide useful ideas or components for a system, as it demonstrates the areas where other research groups have developed control strategies. Other metaOSs may also be useful as they offer a different range of components or interfaces to simulation environments. For example the interface that Player uses to interact with the Gazebo 3D simulation might be reused for physics based simulation of AUV control or sensor processing, though interfaces to other simulations like USARSim [52] may be better depending upon where the group has more experience and the fidelity of the physics required for the simulation.

Such considerations of where it is important to reuse the existing software and experience depend heavily upon the experience level within the development group as well as the system requirements. A brief Case Study of DSTO's Wayamba Project is provided

in the following section to outline how these considerations can impact upon an existing project.

5.1 Case Study: DSTO's Wayamba Project

DSTO has been developing its UUV research platform Wayamba since 1998, with an initial report provided in 2002 [9]. The Wayamba platform is a flatfish design capable of a maximum speed of approximately 4 knots which has two main thrusters, three vertical thrusters, and a single rear lateral thruster as shown in Figure 3. Large static control surfaces at the rear of the vehicle make it very stable in flight, but the mix of propulsors yield flexibility of maneuvering, including having the capability of underwater hovering. Wayamba has two on-board computers to manage the access to on-board sensors, actuators and other devices, as well as the communications within the vehicle and to a remote operations system that can be setup upon a dock or support vessel. The design of Wayamba can be adjusted to allow for the investigation and evaluation of new UUV equipment, such as novel sensor design, or as a concept demonstrator, such as simulating a system to test the launch of offboard submarine sensors [56].

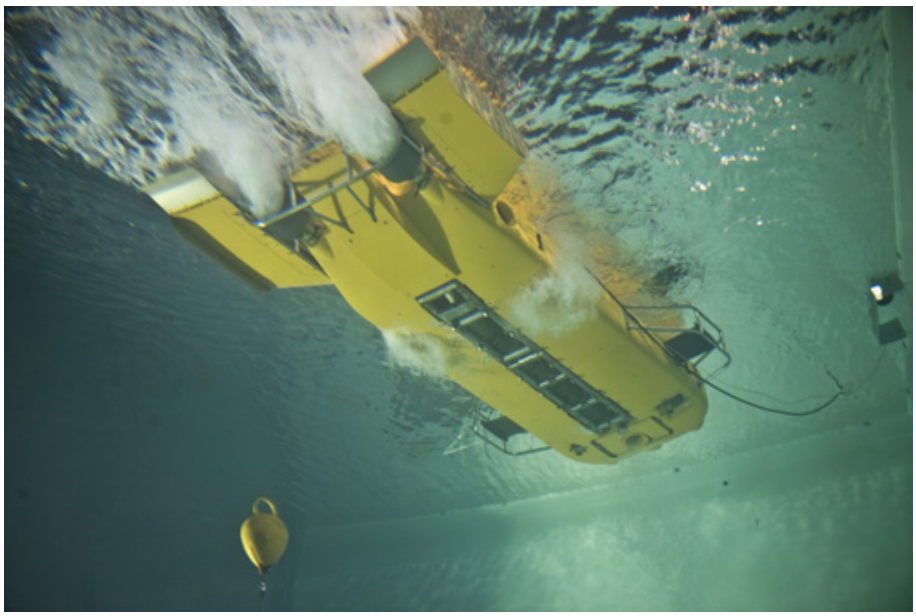


Figure 3: *DSTO's UUV Wayamba undergoing testing in a water tank.*

The two on-board computers are currently split between running the motor control functions and processing sensor inputs and communications with the ship or dockside control station, with both computers running the WinXP OS. A separate commercial autopilot unit has been incorporated for real-time adjustments to maintain the vehicle's attitude and course [10]. Wayamba's software system, WANE [57], utilises a custom communications system based upon a combination of TCP and UDP to connect the AUV components together in a decentralised fashion and connect it with the dockside operator console via a tether. The motor control computer is dedicated to obtaining the UUV's pose information from a high grade Photonic Inertial Navigation System (PHINS) and parsing

the motor control commands with the motor control unit. The other computer obtains data from the other sensors and provides the communications via the optical fibre tether to the operator console. Some components, such as the BlueView sonar, are implemented as network devices and can be access from any computer on the network. Currently the operator console is used to provide commands to control the vehicle motion as well as displaying and logging the system data for the operator.

Given the historical scope and skill sets within the Wayamba project, as well as the typical operation of the vehicle as a demonstration platform in relatively open seas, there is little requirement for the development of highly responsive hard real-time control code. This is more evident when considering that the vehicle usually operates a speeds significantly less than its maximum of 4 knots and, due to its size and mass, it has significant latency in reacting to commands requiring large speed changes. A conversion of the communications structure of such an established system to conform to one of the metaOSs, such as MOOS or ROS, would be difficult and require significant redevelopment; however adapting components from the open source metaOSs may more easily add functionality to the system.

The existing software system is implemented using C++ targeted directly for the WinXP OS, which meets the non real-time requirements of the current platform deployment. Microsoft has now withdrawn support and maintenance for this OS, so a consideration is required of migration to an alternative OS. One option is to retain the current OS with 10 years of experience in its operation, though the maintenance of up-to-date hardware drivers may become increasingly difficult as providers reduce their support. With a low top speed and limited need to operate in a cluttered environment, the requirement for real-time processing is low, reducing the need for changing the software structure to a hard real-time OS. Conversion of the code to operate in the Windows 7 (or 8) environment is likely to cause the least disruption, as some of the current communications and user interface code is not platform independent. It would require significant effort to convert to a different OS such as the Linux based Ubuntu, or Mac OS, which could provide more consistent execution times [19]. If the platform was to be operated at high speed in an environment with many other vessels, such as at an operating port, a greater consideration of a hard real-time OS such as QNX may become important, though the conversion and verification of the software would take considerable time.

6 Conclusions

Unmanned Underwater Vehicles operate in a dynamic ocean environment that is made more challenging by limited underwater communications. An Autonomous Underwater Vehicle must therefore have reliable software and hardware to ensure effective operation where there is little capability for operator input. This document has identified a range of features that are required of an Operating System and meta Operating System that are used as the basis for developing AUV software. A range of potential OSs and metaOSs that have been previously used in robotics were then compared using these features to investigate their suitability for use on AUVs. It was found that for AUVs, the factors which have the most influence on the selection of the OS and metaOSs are whether the operating conditions require critical real-time decision making and the experience level of

the software development team. Most of the other factors were found to be implemented in all of the candidates explored, or had a significantly lower impact upon the progress of software development.

Many current AUV systems are designed and operated as research platforms operating in open ocean environments, where delays of up to a few seconds in movement decisions or activation of actuators have a limited impact on vehicle operation. For such operations, real-time OSs may add little to the reliability of the vehicle, but increase the complexity of the software development and testing required. Where the AUV software development team has experience with the selected OS and metaOS, this can dramatically reduce the development time as it reduces the effort required to learn the intricacies of the new software system. For these reasons many research groups have developed an Ubuntu based system utilising the MOOS metaOS where they often have previous experience, or where there exists a collaborative community to assist those groups with very limited experience.

Where an AUV may operate in a cluttered environment, such as a harbour with multiple moving vessels, real-time decision making becomes significantly more important. These AUVs tend to use the real-time focussed OSs, such as QNX, with software developed specifically for that AUV, though such software may draw on ideas developed in other areas. Where a group has an existing AUV with an existing software system, it is generally very time consuming for them to adapt that software to work on an alternative OS, though they may use ideas or components from the freely available metaOSs when developing new capabilities.

Acknowledgments

Initial research included into this report was conducted under a DSTO research contract with PADJ PL by Dr Christopher Madden.

References

1. The Navy Unmanned Undersea Vehicle (UUV) Master Plan. Published by the US Department of the Navy, (2004).

2. Yildiz, O., Yilmaz, A.E., and Gokalp, B. (2009) State-of-the-Art System Solutions for Unmanned Underwater Vehicles, *Radio Engineering*, **18**(8)

3. Johannsson, H., Kaess, M., Englot, B., Hover, F., and Leonard, J. J. (2010) Imaging Sonar-Aided Navigation for Autonomous Underwater Harbor Surveillance, *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*

4. Williams, S.B., Pizarro, O., Webster, J.M., Beaman, R.J., Mahon, I.J., Johnson-Roberson, M., and Bridge, T.C.L. (2010) Autonomous underwater vehicle assisted surveying of drowned reefs on the shelf edge of the Great Barrier Reef, Australia, *Journal of Field Robotics*, **27**(5), 675–697

5. Brutzman, D. (2007) Autonomous Unmanned Vehicle (AUV) Workbench Rehearsal and Replay: Mapping Diverse Vehicle Telemetry Outputs to Common XML Data Archives. In: *15th International Symposium on Unmanned Untethered Submersible Technology (UUST)*

6. Horner, D.P., McChesney, N., Kragelund, S., and Masek, T. (2009) 3D Reconstruction with an AUV Mounted Forward Looking Sonar. In: *International Symposium on Unmanned Untethered Submersible Technology (UUST)*

7. SAUC-E (2012) Student Autonomous Underwater Challenge - Europe [Accessed 17 March 2012]; `sauc-europe.org/`

8. AUVSI (2013) AUVSI robosub competition [Accessed 21 January 2013]; `http://www.auvsifoundation.org/Competitions/RoboSub`

9. Coxhead, M., Graham, P., Neill, R., Price, P., Travers, A., Wharington, J., and Wright, G. (2002) A Report Card on Wayamba, DSTO's New Uninhabited Undersea Research Vehicle, *Proceedings of Undersea Defence Technology (UDT) Korea*

10. Madden, C., Gilbert, J., and Knox, B (2012) *Integrating the Spectre Autopilot with the Wayamba Research Platform*. DSTO–TN–1124, Melbourne, Vic., Defence Science and Technology Organisation (Australia)

11. Chitre, M., Shahabudeen, S., Freitag L., and Stojanovic, M. (2008) Recent Advances in Underwater Acoustic Communications and Networking. *Marine Technology Society Journal*. **42**(1)

12. Ridao, P., Battle, J., Amat, J., and Roberts, G.N. (1999) Recent trends in control architectures for autonomous underwater vehicles. *International Journal of Systems Science*, **30**(9) 1033–1056.

13. Strutt, J.E. (2006) *Report of the inquiry into the loss of Autosub2 under the Fimbulisen*, National Oceanography Centre Southampton Research and Consultancy Report, 12, National Oceanography Centre, Southampton

14. Dowdeswell, J.A., Evans, J., Mugford, R., Griffiths, G., McPhail, S., Millard, N., Stevenson, P., Brandon, M.A., Banks, C., Heywood, K.J., Price, M.R., Dodd, P.A., Jenkins, A., Nicholls, K.W., Hayes, D., Abrahamsen, E.P., Tyler, P., Bett, B., Jones, D., Wadhams, P., Wilkinson, J.P., Stansfield, K., and Ackley, S. (2008) Autonomous Underwater Vehicles (AUVs) and investigations of the ice-ocean interface in Antarctic and Arctic waters. *Journal of Glaciology*. **54**(187)

15. Craighead, J., Murphy, R., Burke, J., and Goldiez, B. (2007) A Survey of Commercial & Open Source Unmanned Vehicle Simulators. In: *Proceedings of International Conference on Robotics and Automation (ICRA)*

16. Harris, A., and Conrad, J.M. (2011) Survey of Popular Robotics Simulators, Frameworks, and Toolkits. In: *Proceedings of IEEE Southeastcon*

17. dawson (2011) Scientific Linux [Accessed 22 March 2012]; `https://www.scientificlinux.org`

18. Microsoft News Center (August 2001) Windows XP to Take the PC to New Heights, Windows Press Release [Accessed 24 July 2012]; http://www.microsoft.com/en-us/news/press/2001/aug01/08-24winxprtmpr.aspx

19. Ramamritham, K., Shen, C., Gonzalez, O., Sen, S., and Shirgurkar, S. (1998) Using Windows NT for Real-Time Applications: Experimental Observations and Recommendations, In: *Proceedings of Real-Time Technology and Applications Symposium*:102–111

20. McGann, C., Py, F., Rajan, K., Thomas, H., Henthorn, R., and McEwen, R. (2007) T-REX: A Model-Based Architecture for AUV Control, In: *Proceedings of International Conference on Automated Planning and Scheduling Workshop*

21. Parodi, O., Lapierre, L., and Jouvencel, B. (2009) Hardware-in-the-loop Simulators for Multi-vehicle Scenarios: Survey on Existing Solutions and Proposal of a New Architecture, In: *International Conference on Intelligent Robots and Systems (IROS)*

22. Brutzman, D. (2010) NPS AUV Workbench: Rehearsal, Reality, Replay for Unmanned Vehicle Operations, In: *Oceanic Engineering Society's conference on Autonomous Underwater Vehicles (OES AUV)*

23. Apple (2012) Mac Technology Overview [Accessed 12 June 2012]; https://developer.apple.com/technologies/

24. Litayem, N., Achballah, A.B., and Saoud, S.B. (2011) Building XenoBuntu Linux Distribution for Teaching and Prototyping Real-Time Operating Systems. *International Journal of Advanced Computer Science and Applications.* **2**(2)

25. Oxford Mobile Robotics Group (2011) MOOS Home Page [Accessed 8 March 2011]; www.robots.ox.ac.uk/~mobile/MOOS/wiki/pmwiki.php/Main/HomePage

26. TullyFoote (2013) Robot Operating System (ROS), [Accessed 16 March 2013]; www.ros.org/wiki/

27. Gerkey, B. (2011) The Player Robot Device Interface [Accessed 16 March 2011]; http://playerstage.sourceforge.net/doc/Player-2.1.0/player/

28. Calutt, S., Weatherill, D., Rickenbach, J., Ridge, A., Swirski, L., Crosby, J., Vincent, H., Pritchard, A., Barton, R., Bremner, S., Silaghi, D., Luk, J., and Dunietz, J. (2010) *Cambridge AUV 2010*, Technical Report published by Cambridge University

29. Veerabahu, M. (2011) Windows CE 6.0 vs 7.0, Published by E-Con systems [Accessed 16 March 2011]; http://www.e-consystems.com/WindowsCE6vs7.asp

30. Applied Acoustics (2011) EasyTrak Technical Specification [Accessed 12 March 2011]; http://www.easy-trak.com/pdf/EasytrakPortableAndLite.pdf

31. ENEA (2011) Enea OSE [Accessed 6 March 2011]; http://www.enea.com/solutions/rtos/ose/

32. Yodaiken. V. (1999) The RTLinux Manifesto, In: *5th Linux Conference Proceedings*

33. Kang, D., Lee, W., and Park, C. (2007) Kernel Thread Scheduling in Real-Time Linux for Wearable Computers, *Electronics Telecommunications Research Institute (ETRI) Journal* **29**(3)

34. Wantanabe, K. (2005) Compact AUV Platform System Designed for the Experiment of Underwater Multi-agent Development, In: *International Conference on Computer Applications in Shipbuilding*

35. wind river (2011) Wind River VxWorks: Embedded RTOS with support for POSIX and SMP. [Accessed 17 March 2011]; `www.windriver.com/products/vxworks/`

36. Wind River (2011) Wind River Aerospace and Defence: Space [Accessed 15 March 2011]; `www.windriver.com/solutions/aerospace-defense/`

37. Valavanis, K.P., Gracanin, D., Matijasevic, M., Kolluru, R., and Demetriou, G.A. (1997) Control Architectures for Autonomous Underwater Vehicles, In: *IEEE Control Systems*

38. QNX (2008) Realtime Programming under QNX Neutrino: Course Notes

39. Bian, X., Zou, H., Chang, Z., Zhao, D., and Wang, Z. (2005) Multi-thread Technology's Application in the Decision-making System of AUV. In: *Proceedings International Conference on Mechatronics and Automation*

40. K.J. Higgins (2008) Secure OS Gets Highest NSA Rating, Goes Commercial, Published by Dark Reading, [Accessed February 2011]; `http://www.darkreading.com/applications/secure-os-gets-highest-nsa-rating-goes-c/212100421`

41. Mills, E. (2008) Green Hills spins off Integrity operating system, Published by CNet, [Accessed February 2011]; `http://news.cnet.com/8301-1009_3-10102784-83.html`

42. Bradski, G. and Kaehler, A. (2008) Learning OpenCV: Computer Vision with the OpenCV library. **O'Reilly Media**

43. Gerkey, B., Vaughan, R., and Howard, A. (2003) The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems, In: *Proceedings of the International Conference on Advanced Robotics (ICAR)*

44. Collett, T., MacDonald, B., and Gerkey, B. (2005) Player 2.0: Toward a Practical Robot Programming Framework, In: *Australasian Conference on Robotics and Automation*

45. Benjamin, M.R., Schmidt H., Newman, P., and Leonard, J.J. (2013) An Overview of MOOS-IvP and a Users Guide to the IvP Helm - Release 13.5 [Accessed 24 May 2013]; `http://oceanai.mit.edu/moos-ivp-pdf/moosivp-helm.pdf`

46. Jiang, D., Pang, Y. and Qin, Z. (2010) Coordination of Multiple AUVs Based on MOOS-IvP, In: *International Conference on Control and Automation (ICCA)*

47. Benjamin, M.R., Schmidt, H., Newman, P.M., and Leonard, J.J. (2010) Nested Autonomy for Unmanned Maritime Vehicles with MOOS-IvP, *Journal of Field Robotics* **27**(6): 834–875.

48. Eickstedt, D.P., and Sideleau, S.R. (2009) The Backseat Control Architecture for Autonomous Robotic Vehicles: A Case Study with the Iver2 AUV, In: *OCEANS, Marine Technology for Our Future: Global and Local Challenges*

49. Baillie, J-C (2008) Urbi 2: Introduction to Concurrent Real-time Programming, In:*Proceedings of the Fourth International Workshop on Software Development and Integration in Robotics*

50. Evolution Robotics (2011) ERSP 3.1 Robotic Development Platform [Accessed 17 March 2011]; `http://www.evolution.com/products/ersp/`

51. Chrysanthakopoulos, G., and Singh, S. (2005) An Asynchronous Messaging Library for C#, In: *Proceedings of the Workshop on Synchronization and Concurrency in Object-Oriented Languages*

52. Sehgal, A., and Cernea, D. (2010) A Multi-AUV Mission Simulation Framework for the USARSim Robotics Simulator, In: *18th Mediterranean Conference on Control and Automation Congress*

53. Senarathne, P.G.C.N., Wijesoma, W.S., Kwang, W.L., Kalyan, B., Moratuwage, M.D.P., Patrikalakis, N.M., and Hover, F.S. (2010) MarineSIM: Robot simulation for marine environments, In: *International Conference on OCEANS*

54. Craighead, J., Burke, J., and Murphy, R. (2008) Using the Unity Game Engine to Develop SARGE: A Case Study, In: *Simulation Workshop at the International Conference on Intelligent Robots and Systems (IROS)*

55. Koenig, N., and Howard, A. (2004) Design and Use Paradigms for Gazebo, An Open-Source Multi-Robot Simulator, In: *IEEE/RSJ International Conference on Intelligent Robots and Systems(IROS)*

56. Rodgers, J., Wharington, J., Tynan, A., and Coxhead, M. (2008) A concept for the deployment of unmanned maritime systems from submarines: MURULA integration, impact modelling and results, *Undersea Defence Technology (UDT) Pacific*

57. Knox, B. (2012) *The WANE Manual*, DSTO–GD–0705, Melborune Vic., Defence Science and Technology Organisation (Australia)

DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION DOCUMENT CONTROL DATA	1. CAVEAT/PRIVACY MARKING

2. TITLE	3. SECURITY CLASSIFICATION
An Evaluation of Potential Operating Systems for Autonomous Underwater Vehicles	Document (U) Title (U) Abstract (U)
4. AUTHOR	5. CORPORATE AUTHOR
C. Madden	Defence Science and Technology Organisation 506 Lorimer St, Fishermans Bend, Victoria 3207, Australia

6a. DSTO NUMBER	6b. AR NUMBER	6c. TYPE OF REPORT	7. DOCUMENT DATE
DSTO–TN–1194	AR-015-676	Technical Note	February 2013

8. FILE NUMBER	9. TASK NUMBER	10. TASK SPONSOR	11. No. OF PAGES	12. No. OF REFS
2011/1026884/1	FUSW CERP	CDS	24	57

13. URL OF ELECTRONIC VERSION	14. RELEASE AUTHORITY
`http://www.dsto.defence.gov.au/publications/scientific.php`	Chief, Maritime Platforms Division

15. SECONDARY RELEASE STATEMENT OF THIS DOCUMENT

Approved for Public Release

OVERSEAS ENQUIRIES OUTSIDE STATED LIMITATIONS SHOULD BE REFERRED THROUGH DOCUMENT EXCHANGE, PO BOX 1500, EDINBURGH, SOUTH AUSTRALIA 5111

16. DELIBERATE ANNOUNCEMENT

No Limitations

17. CITATION IN OTHER DOCUMENTS

No Limitations

18. DSTO RESEARCH LIBRARY THESAURUS

Underwater Vehicles Operating Systems Robotics
Software Systems

19. ABSTRACT

This document explores the Operating System (OS) features that are required for the automation of Unmanned Undersea Vehicles (UUVs), from the perspective of a new system development. Recent improvements in system components and sensors allow for increased vehicle capability; however reliable automated software is required to perform extended missions due to the communications limitations with submerged vehicles. The first step of software development for such systems is selecting an appropriate Operating Systems which allows the system to support the software required to perform real-time sensor analysis, and control its behaviour to achieve the mission within the local environmental conditions. This document defines key features for the evaluation of potential Operating Systems for the UUVs with autonomous or semi-autonomous capabilities. Using these features, it explains why some UUVs use a real-time Operating System with custom designed software, but others use the open source Ubuntu Operating System combined with the Mission Oriented Operating System (MOOS) robotics software package, or metaOS. It also emphasises that reuse of existing software and skills are important to advancing any AUV project more rapidly.