



Defence Research and  
Development Canada

Recherche et développement  
pour la défense Canada



# **Unmanned Ground Vehicle Software Development Environment**

*A Flexible and Portable Environment for Developing UGV Software*

G. Broten, S. Verret and B. Digney  
Defence R&D Canada – Suffield

Technical Memorandum  
DRDC Suffield TM 2004-060  
June 2004

Canada



# **Unmanned Ground Vehicle Software Development Environment**

*A Flexible and Portable Environment for Developing UGV Software*

G. Broten

S. Verret

B. Digney

Defence R&D Canada – Suffield

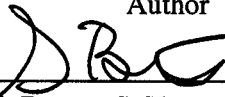
**Defence R&D Canada – Suffield**

Technical Memorandum

DRDC Suffield TM 2004-060

June 2004

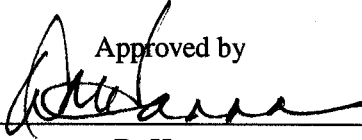
Author



---

G. Broten, S. Verret, B. Digney

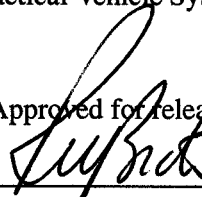
Approved by



---

D. Hanna  
Head/Tactical Vehicle System Section

Approved for release by



---

R.W. Bide  
Chairman/Document Review Panel

© Her Majesty the Queen as represented by the Minister of National Defence, 2004

© Sa majesté la reine, représentée par le ministre de la Défense nationale, 2004

## Abstract

---

The Autonomous Intelligent Systems (Land) initiative has been tasked with researching and developing innovative autonomous vehicles that will assist the Canadian Forces in performing their duties in the 22nd century. This research will continue for many years on many different types of unmanned ground vehicles. Thus it requires a software development environment that will protect what will become a substantial investment in intellectual property that is implemented in software algorithms. This environment must be based upon standards that are as universally supported as possible. The software development environment must also be compatible with the Autonomous Land Systems (ALS) project Unmanned Ground Vehicle Electronic Hardware Architecture[1]. This report describes the software development environment developed for the ALS project. The environment is based upon open standards and meets the diverse requirements of the project. This environment is currently being used by all applications being developed for unmanned ground vehicles (UGV) at DRDC Suffield under the ALS project.

## Résumé

---

L'initiative des Systèmes intelligents autonomes (terrestres) a reçu la mission d'entreprendre des recherches et de mettre au point des véhicules autonomes novateurs qui procureront, au 22<sup>e</sup> siècle, une assistance aux Forces canadiennes durant l'exercice de leurs fonctions. Cette recherche se déroulera durant plusieurs années sur plusieurs types de véhicules terrestres sans pilote. Ceci exige un environnement d'élaboration de logiciels visant à protéger ce qui deviendra un important investissement en propriété intellectuelle, implémentée par des algorithmes logiciels. Cet environnement doit être basé sur des normes aussi universellement reconnues que possible. Cet environnement d'élaboration de logiciels doit aussi être compatible avec les Systèmes terrestres autonomes (STA) du projet de l'Équipement électronique de l'architecture des véhicules terrestres sans pilote[1]. Ce rapport décrit le développement de l'environnement de logiciels élaborés pour le projet STA. L'environnement est basé sur des normes ouvertes et satisfait aux diverses exigences du projet. Cet environnement est actuellement utilisé par toutes les applications développées pour les véhicules terrestres sans pilote à RDDC sous le projet STA.

## Executive summary

---

**Background:** Defence R&D Canada, on behalf of Canadian Department of National Defence, is developing new and innovative land vehicles. An important aspect of this research is the investigation and development of autonomous unmanned ground vehicles to augment and/or replace existing tele-operation capabilities. This research is being conducted under the auspices of the Tactical Vehicles Systems Section (TVSS) as part of their ALS project. This is an on-going field of research that is expected to define the future capabilities of autonomous unmanned ground vehicles. This research is expected to yield valuable intellectual property (IP) with a significant portion of this IP being implemented as software algorithms. In order to protect this investment in IP the ALS project has defined a software development environment. The environment defines set of standards to follow and the tool chain to be used in the software development process. The use of this software development environment will help ensure the long term viability of all software developed for the ALS project and thus for AIS (Land) research initiative.

**Principle Results:** A software development environment has been selected for the ALS project. This environment defines five keys aspects of the software development process:

1. The ALS project supports the Open Source Model and will use Open Source tools wherever possible.
2. The C/C++ languages will be used for all software development related to the core competence of the ALS project.
3. All software will use the GCC tool chain for compiling the high level languages into the machine code instructions that are used at the processor level.
4. The Linux operating system will be used for software that has soft real-time or lesser constraints. For software that interacts with external devices that impose hard real-time restraints the RTEMS operating system will be used.
5. All software developed will adhere to the POSIX 1003.1 and 1003.1b extensions where possible.

This software development environment defines a set of standards, tools, languages and applications that are well suited to meet the needs of the ALS project. The proposed environment defines a model for developing software that protects DRDC's investment in the intellectual property that the software represents while maintaining the flexibility and scalability that is required for compatibility with UGV Electronic Hardware Architecture[1].

This software development environment will allow the software developed under the ALS project to be used across a range of UGV platforms that feature diverse requirements. It will allow the researchers working on the ALS project to concentrate on developing algorithms that enable and enhance UGV autonomy with minimal concerns about algorithm compatibility between the various types of UGV platforms.

G. Broten, S. Verret, B. Digney. 2004. Unmanned Ground Vehicle Software Development Environment. DRDC Suffield TM 2004-060. Defence R&D Canada – Suffield.

## Sommaire

---

**Contexte :** R & D pour la défense Canada, au nom du ministère de la Défense nationale, met actuellement au point des véhicules terrestres nouveaux et novateurs. Un aspect important de cette recherche consiste à investiguer et à mettre au point des véhicules terrestres autonomes sans pilote afin d'augmenter et / ou de remplacer les capacités de téléopération existantes. Cette recherche est conduite sous les auspices de la Section des systèmes de véhicules tactiques et fait partie de leur projet STA. Il s'agit d'un domaine de recherche de longue durée devant définir les capacités futures des véhicules terrestres autonomes sans pilote. On s'attend à ce que cette recherche produise une propriété intellectuelle intéressante dont une partie importante sera implémentée par des algorithmes logiciels. Le projet STA a défini l'environnement du processus d'élaboration de logiciels qui vise à protéger cet investissement en propriété intellectuelle. L'environnement définit un ensemble de normes à suivre et la chaîne d'outils à utiliser dans le processus d'élaboration des logiciels. L'utilisation de cet environnement d'élaboration de logiciels permettra d'assurer la viabilité à long terme de tous les logiciels développés pour le projet STA et ainsi pour l'initiative de recherche SIA (terrestre).

**Résultats de principe :** Un environnement d'élaboration de logiciels a été sélectionné pour le projet STA. Cet environnement définit cinq aspects clés du processus d'élaboration de ces logiciels :

1. Le projet STA soutient le Modèle du logiciel ouvert et utilisera autant que possible des outils ouverts.
2. Les langages C/C++ seront utilisés pour tout le développement des logiciels associés à la compétence fondamentale du projet STA.
3. Tous les logiciels utiliseront la chaîne d'outils de la collection de compilateurs GNU pour compiler les langages de haut niveau dans le code des instructions d'ordinateur qui seront utilisées au niveau processeur.
4. Le système d'exploitation Linux sera utilisé pour les logiciels qui ont des contraintes en temps réels souples ou des contraintes moindres. Des systèmes d'exploitation en temps réel en mode multitâche seront utilisés pour les logiciels qui interagissent avec des appareils externes imposant des temps réels à forte contrainte.
5. Tous les logiciels élaborés adhéreront autant que possible aux extensions POSIX 1003,1 et 1003,1 b.

L'environnement d'élaboration de logiciels définit un ensemble de normes, d'outils, de langages et d'applications bien adaptés aux besoins du projet STA. L'environnement proposé définit un modèle d'élaboration de logiciels qui protège l'investissement en propriété intellectuelle de RDDC, représenté par le logiciel, tout en maintenant la souplesse et la variabilité dimensionnelle requise pour être compatible avec l'Équipement électronique de l'architecture des véhicules terrestres sans pilote[1].

Cet environnement d'élaboration de logiciels permettra au logiciel développé pour le projet STA d'être utilisé pour toute la gamme des plates-formes de véhicules terrestres sans pilote qui possèdent des exigences diverses. Il permettra aux scientifiques de travailler sur le projet STA en se concentrant sur le développement d'algorithmes. Ces derniers permettront et amélioreront l'autonomie des véhicules terrestres sans pilote et ne présenteront que des préoccupations minimales au sujet de la compatibilité des algorithmes entre les différents types de plates-formes STA.

G. Broten, S. Verret, B. Digney. 2004. Unmanned Ground Vehicle Software Development Environment. DRDC Suffield TM 2004-060. R & D pour la défense Canada – Suffield.



# Table of contents

---

Abstract . . . . .	i
Resume . . . . .	i
Executive Summary . . . . .	ii
Sommaire . . . . .	iii
Table of contents . . . . .	v
List of tables . . . . .	vii
1. Introduction . . . . .	1
1.1 Electronic Hardware Architecture . . . . .	1
2. Software Architecture . . . . .	1
2.1 Interaction with Hardware . . . . .	2
2.2 Software Modes . . . . .	2
2.2.1 Consumer Software . . . . .	3
2.2.2 Industrial Software . . . . .	3
2.2.3 The Open Source Model . . . . .	4
2.2.3.1 Open Source Definition . . . . .	4
2.2.3.2 Significance and Advantages of the Open Source Model . . . . .	5
2.2.4 Open Source and the ALS Project . . . . .	6
3. ALS Development Environment . . . . .	6
3.1 Software Languages . . . . .	6
3.1.1 Java . . . . .	7
3.1.2 C# . . . . .	7
3.1.3 Ada . . . . .	8

3.1.4	C and C++ . . . . .	8
3.1.5	Conclusions . . . . .	8
3.2	C/C++ Compilers . . . . .	9
3.2.1	GNU C/C++ Compiler . . . . .	10
3.2.2	Intel Compiler . . . . .	10
3.2.3	Microsoft Visual C++ .NET . . . . .	10
3.2.4	Fujitsu Compiler . . . . .	10
3.2.5	IBM VisualAge . . . . .	10
3.2.6	Summary . . . . .	11
3.3	Operating Systems . . . . .	11
3.3.0.1	Soft and Hard Real-time Capabilities . . . . .	12
3.3.0.2	Operating System Portability . . . . .	12
3.3.0.3	Operating System Evaluation . . . . .	13
3.3.1	Linux . . . . .	13
3.3.2	FreeBSD . . . . .	14
3.3.3	RTLinux and RTAI . . . . .	14
3.3.4	MontaVista Linux and Linux 2.6 Kernel . . . . .	15
3.3.5	TimeSys Linux . . . . .	15
3.3.6	eCos . . . . .	16
3.3.7	RTEMS . . . . .	17
3.3.8	Linux Operating System Performance . . . . .	18
3.3.9	Summary . . . . .	19
4.	Conclusions . . . . .	21
	References . . . . .	23

## List of tables

---

Table 1. Candidate Operating Systems . . . . .	12
Table 2. Kernel Preemption Latency Comparison[2] . . . . .	18
Table 3. RedHat Benchmarks of the Linux Scheduler[3] . . . . .	19

This page intentionally left blank.

# 1. Introduction

---

Defence R&D Canada, on behalf of the Canadian Department of National Defence, is developing new and innovative land vehicles. An important aspect of this research is the investigation and development of autonomous UGVs to augment and/or replace existing tele-operation capabilities. This research is being conducted under the auspices of the TVSS as part of their AIS (Land) research initiative. This is an on-going field of research that is expected to define the future capabilities of autonomous UGVs.

The key aspect of this research is the autonomous capabilities that it will develop. There is a taxonomy of operational regimes for UGVs<sup>1</sup> ranging from a fully autonomous through to a semi-autonomous system[4]. A robot is called fully autonomous when it operates without the need for full-time external human control. A semi-autonomous robot requires a full-time human operator but is permitted to make certain decisions on its own. Within the semi-autonomous division there are two sub-categories: tele-robotic and tele-operated. A tele-robotic system has levels of software that interpret the operator's commands while a tele-operated robot is directly and completely controlled by the operator.

## 1.1 Electronic Hardware Architecture

The Unmanned Ground Vehicle Electronic Hardware Architecture[1] memorandum describes an electronic hardware architecture that has the flexibility and extensibility to support a wide range of UGV platforms. It achieves this flexibility by ascribing to a distributed paradigm which enables the use of multiple scales of processors. The Electronic Hardware Architecture is applicable to small indoor platforms with limited payloads and it easily scales to support large platforms that do not have payload limitations. The hardware architecture does not exist in isolation since it provides services and functionality to software algorithms. It is the software that creates autonomous capabilities to the UGV. These autonomous capabilities are implemented by software algorithms such as: the representation of the world; path planning; obstacle avoidance; and others. Additional software will be involved in commanding the various motors that allow the UGV to traverse the selected path.

Given the important role software algorithms play in developing a UGV the software must work hand in hand with hardware to implement an overall architecture that provides the scalability and flexibility required by the ALS project.

## 2. Software Architecture

---

The ALS program is researching and developing UGVs that will assist the Canadian Forces in performing a variety of tasks. A key component of this research is the

---

<sup>1</sup>The terms unmanned ground vehicles, autonomous ground vehicles, robots and mobile robot are used synonymously throughout this report.

development of software algorithms that create autonomous capabilities on a UGV. It is probable that the algorithms developed today will be in use for many years and perhaps even decades. To a large extent these algorithms encapsulate the knowledge acquired through an intensive research and development process. Given the large scale investment that these algorithms will represent it is essential that they be as *future proof* as possible. The term *future proof* refers to following requirements:

- The programming language(s) of implementation must be relevant and in use for many years/decades.
- The compiler(s) must have a high probability of continued use and advancement well into the future.
- The operating system(s) must be of a nature that they will have a long lifespan.

The prime goal of the software architecture is protect the ALS program from expensive ports or re-writes that occur when the support for a given component is no longer available. While it is preferable to use a single programming language, compiler and operating system, this may or may not be achievable. Certain programming languages are well suited for larger projects, while others are more suitable for embedded devices and it may be advisable to use a set of complimentary programming languages.

## **2.1 Interaction with Hardware**

The high level goals of the software architecture described in Section 2. are also influenced by the hardware requirements of the ALS project. As outlined in Section 1.1 the ALS project requires a flexible and scalable hardware architecture. To assist in meeting these flexibility and scalability requirements the software architecture should support a wide range of processors, from a variety of manufactures. The capability to support differing processors allows the ALS project to use the appropriate processor for the particular task at hand[1]. Thus, the programming language(s), compiler(s) and operating system(s) must be portable across a range of processor options, ranging from high speed number crunching processors to specialized embedded micro-processors.

## **2.2 Software Modes**

Three major software modes can be identified. The software mode is defined by the major features that characterize the software. The three modes are:

- Consumer Software
- Industrial Software
- Open Source Software

### **2.2.1 Consumer Software**

The world of desktop/laptop computers is representative of consumer software. This world features ever increasing performance and short product lifespans. Consumer software has a strong tendency to follow the latest available hardware technology and thus it is in a constant state of change. Given the huge market size this software services, the prices tend to be relatively inexpensive. The dominance of Microsoft in this market results in a propensity to adopt common standards which results in good cross product compatibility.

This market has traditionally been characterised by the user demanding improved computer performance. This demand for superior performance has led to an upgrade cycle that has often benefited both the vendor of applications and the user. The user has received better and more powerful applications than the previous versions while the vendor has profited from developing and selling these new or improved applications.

The bottom line for this market is that the vendor's prime goal is the generation of a revenue and profit. In some cases this prime goal of the vendor is in direct conflict with the requirements of the user of the application. When an application has matured the addition of more processing power does not lead to a substantial improvement of the product. At this point the vendor still has an incentive to convince the user to upgrade to next release of the application whether or not this is in the best interests of the user. This motivation to convince the user to upgrade is often associated with the practice on not keeping applications compatible with previous versions. To add an additional incentive to upgrade the vendor sometimes drops support for previous versions of the application and leaves the user with no avenues to solve problems or fix bugs. Hence, there is a very strong tendency in the world of consumer applications for the user to be forced into upgrades that are neither desired or required.

### **2.2.2 Industrial Software**

Industrial software, in contrast to consumer software, features much longer lifespans. It caters to industries such as the automotive and forestry sectors where the infrastructure costs are high. Software used in this setting is expected to have a much longer lifespan. Significant investments are made in developing specialized software for the factory floor and office. In order to receive a good return on this investment this software is often expected to have a lifespan in the realm of the lifespan of the equipment on the factory floor. These special requirements that the industrial sector impose on software has had the effect of forcing the software vendors into product cycles that are appropriately matched with the needs of their customers and thus offering software with longer lifespans than found in the consumer market.

While industrial software has the positive feature of longer lifespans it unfortunately suffers from a very fragmented market. No single vendor like Microsoft dominates the market. A large group of vendors lead by VxWorks, CMX, US Software, Accelerated Technology, Metroworks, GE Fanuc, Allen Bradley and many others have offerings in this market. Unfortunately there are no widely implemented standards that all vendors support that allow for cross vendor transportability. The IEEE has sanctioned the Portable Operating System Interface (POSIX) standardization effort that defines common standards for operating systems. While the POSIX standard is often supported, many vendors support is only a subset of the POSIX standard. Even if a vendor supports the POSIX standard there is a plethora of other applications, such as debuggers and editors, that the vendor supplies that may not be compatible other vendor's products. Each vendor has an incentive to keep clients and it is not in their interest to allow a user to easily switch to a competitor's product.

### **2.2.3 The Open Source Model**

The open source model has numerous advantages over the consumer and industrial models, but before expanding on these advantages it is first necessary to define the open source model.

#### **2.2.3.1 Open Source Definition**

The Open Source Initiative (OSI) is a non-profit corporation dedicated to managing and promoting open source software. They define open source as having the following attributes:

1. Free redistribution
2. Source code
3. Derived works
4. Integrity of the author's source code
5. No discrimination against persons or groups
6. No discrimination against fields of endeavor
7. Distribution of license
8. The license must not be specific to a product
9. The license must not restrict other software
10. The license must be technology neutral



A detailed explanation of the significance of each of the attributes can be found on the Open Source Initiative website<sup>2</sup>. In summary the goals of open source software are to give software developers access to the source code and to allow them to modify the software.

### **2.2.3.2 Significance and Advantages of the Open Source Model**

The most significant aspects of the open source model are:

- Community ownership.
- The open and collaborative development model.
- Support of open standards.

These three aspects create loyal user and developer communities that have an ongoing stake in the software. To appreciate the significance of these aspects it is useful to contrast this type of community with the commercial software products and its associated upgrade cycle. The open source, by contrast, is driven by different motivations. While the open source model is not immune to upgrade cycle, the forces driving the release of new versions are significantly different. A new release of an application under the open source model is mostly driven by the demands of the users and advances in technology. Given that the open source upgrade cycle is driven by the user there is a very strong incentive to keep backwards compatibility.

The open source model relies on a community approach to software development and thus there is a strong motivation for the community to use open standards where possible. Open standards are advantageous to the open source model since they represent a common rule set which all developers can abide by. In contrast it is less attractive for commercial vendors to support open standards since open standards allow the user to migrate to a competitor's product.

The open source user, in the final analysis, always has the option of not upgrading and continue to in-house support of application in question. The source code is available and a cost benefit analysis can be performed comparing the cost of the upgrade with the cost of performing in-house<sup>3</sup> support. This option for in-house support

---

<sup>2</sup>OSI website definition of Open Source can be found at <http://www.opensource.org/docs/definition.php>.

<sup>3</sup>The in-house support could be contracted to an outside developer.

is not available for closed source commercial or industrial applications.

#### **2.2.4 Open Source and the ALS Project**

The ALS project will use the open source applications for components of the software architecture wherever they meet the needs of the project. The program accrues a number of benefits by choosing open source model. The program requires software applications that are very long lived. For the reasons listed in Section 2.2.3.2 the life expectancy and compatibility of open source software applications are superior to most offerings from commercial vendors. As for industrial software, while its lifespan is superior to straight commercial software, the fragmentation of the industrial software market has lead to major incompatibilities between its numerous vendors.

Finally it is important to note that the quality of open source tools are in most cases equivalent or superior to commercial offerings. There is no penalty to pay by choosing open source applications. In many important instances the open source offerings are the best that are available.

For all the reasons outlined in Section 2.2.3 the ALS project concluded that the open source model is the best model to ensure the longevity of the software algorithms developed.

### **3. ALS Development Environment**

---

The ALS software development environment defines the software language(s), compiler(s) and operating system(s) that are to be used when developing software algorithms. This development environment tries to meet the various needs of the ALS project as detailed in the previous sections.

#### **3.1 Software Languages**

The ALS project had a variety of software languages to chose from when selecting the language(s) to use for software development. A list of relevant languages was compiled and each language was investigated. Each language was graded by the following criteria:

- Good longevity, meaning that it will be in use many years into the future.
- A very large and significant user base.
- Processor independence, in terms of not being restricted to a particular processor architecture.

- The language's applicability to use in real-time systems.

The languages considered included: Java, C#, Ada, C and C++.

### **3.1.1 Java**

Java is a very promising language with its roots in platform independence. It has a large user base, especially among web programmers where Java applets are used extensively. Java has a two part implementation: the platform independent Java byte code and the platform dependent Java Virtual Machine. While in theory Java is very attractive in practice it suffers from limitations. Java is a relatively new language and the Java Virtual Machine is not widely available across processing platforms. There are many processors, especially in the embedded realm, for which the Java Virtual Machine does not yet exist. Additionally, standard Java is not particularly well suited for real-time applications. The automatic garbage collection process of standard Java is unrestrained and thus can cause problems for real-time systems. The Real-Time Specification for Java (RTSJ), also known as JSR-1, is an implementation of Java for real-time systems but it is not widely available and it has not been proven to be capable and robust. Finally, Microsoft does not support Java but has instead released a competing language which is known as C#. As a result the verdict is not in yet on the long term acceptance of this language.

### **3.1.2 C#**

The C# language is Microsoft's response to Java and is in many ways similar to Java. C# promises to be extremely popular on systems supporting Microsoft operating systems. It has a large and growing user base due to its incorporation into Microsoft's development environment and it can be expected that Microsoft will continue to use this language well into the future. At the present time C# is almost exclusively tied to Microsoft operating systems. An open source clone of the C# language and .Net development platform, called Mono, is underway but has yet to be completed. Microsoft C# is tied exclusively to the Intel line of processors. The Mono project, if successful, may expand the C# language to include support for all processors support by Linux. It must be noted that the Mono project is not funded, supported or approved by Microsoft and the success of an open source implementation of C# is far from assured. From the real-time perspective the capabilities of C# are weak. Lutz and Laplante tested the real-time capabilities of C# and the .Net Framework and concluded that it was not suitable for hard real-time systems but with the appropriate precautions it could be used under soft real-time conditions[5].

### 3.1.3 Ada

Ada is a modern language that was originally developed for the U.S. military. It features code that is: portable, modular, reusable, reliable and maintainable. Additionally, Ada has been specifically engineered for real-time applications. Major Ada applications include: strategic military embedded systems; NASA's Space Shuttle and Space Station environments; and commercial jets and air traffic control systems. While Ada has many admirable attributes its general acceptance has been limited. Ada is not widely used outside the niche sectors where reliability is a paramount issue, hence it has a limited user base. While military UGVs could benefit from such reliability, the ALS project is at a stage where sharing algorithms with other research institutions is important and the use of a niche language such as Ada would be a hindrance to the goals of the ALS project.

### 3.1.4 C and C++

C and C++ are exceptionre closely related programming languages that are probably the most widely used languages in the world. C has its roots in the early 1970s when it was created as a language to code the Unix operating system. C++ started life as C with object classes in the early 1980 before it was eventually released as C++ three years later. These languages boast huge developer bases and have been used to develop an innumerable number of applications. C has been the workhorse language for real-time and embedded systems for decades. C++ is well suited for large scale software development projects and can be used under real-time conditions if precautions are taken. Both of these languages have been used for years and will be continued to be used for the foreseeable future. The C language can be used on probably all processors and micro-processors that currently in production. C++, which requires more resources in terms of processing power and memory, is not as well suited for micro-processors, but is supported by general purpose processors. Given that C and C++ are such commonly used programming languages they also facilitate the co-operation between research institutions.

### 3.1.5 Conclusions

The ALS project has selected the C/C++ combination of programming languages as the standard languages to be used for software development. They both have a long history of use with very large user communities who will help ensure that these languages continue to thrive long into the future. C is the arguably the most portable language in the world given that it was specifically developed as a language for creating operating systems<sup>4</sup> and as a

---

<sup>4</sup>To this day the major of all major operating systems are developed using C.

result is available on probably every processor/micro-processor<sup>5</sup> in production. C++ is also very widely supported across numerous processor architectures. Both languages can be used with real-time systems though C has a performance advantage over C++. The other languages surveyed did not have the breadth of support that is found with the C/C++ combination. Java is limited by processor architecture for which the Java Virtual Machine exists. C# is heavily tied to the Intel processor architecture which precludes its use on embedded systems featuring micro-processors. Ada is excellent for real-time applications but does not have a large acceptance outside of niche applications. In the final analysis the C/C++ languages have the longevity, user base, processor support and real-time capabilities that are demanded by the ALS project.

### **Other Programming Languages**

While the ALS project has selected the C/C++ combination of languages, software that is not related to the core competence of the ALS project may use other programming languages. This exception would allow specialized applications to integrate into the UGV architecture on an as needed basis. The merits of accepting non-standard software would be reviewed on a case by case basis. Under these instances a preference would be given to languages that fall within the C family of languages.

## **3.2 C/C++ Compilers**

For a computer language to be useful it must have a compiler to parse the high level language and turn it into machine code that a processor is capable of executing. The high level languages to be used by the ALS project are the C/C++ combination and thus a C/C++ compiler is required. The use of a single compiler is required since using the same compiler, for all code, simplifies the moving of coded algorithms across processor architectures. This simplification occurs since the compiler uses the same constructs in terms of defining the processor, debugging, optimization options and other compiler directives.

There are many C/C++ compilers available including: Fujitsu, GNU, Intel, IBM and Microsoft to name a few. For the ALS project one of the most important attributes of the compiler is the range of processors that the compiler supports. Flexibility and scalability are key aspects of the ALS project and the compiler must support a variety of processor architectures to meet these goals.

---

<sup>5</sup>The micro-processor claim should be qualified by the note that certain, small, 8 bit processors may only support assembly language programming.

### **3.2.1 GNU C/C++ Compiler**

The GNU C/C++ compiler is an open source compiler suite for the C and C++ languages. It can run on a multitude of platforms ranging from mainframe system all the way down to embedded systems<sup>6</sup>. This compiler is also capable of being configured as cross compiler where it runs under one processor architecture and compiles code for a different target architecture. It is portable across a wide range of processors include: Intel x86, PowerPC, VAX, Motorola 68K, MIPS, DEC Alpha, Texas Instruments TMS320 DSPs, Motorola Mcore, ARM architecture, Sun Sparc, and other less used processor models. The GCC compiler runs under numerous flavours of unix including: Linux, AIX, HP-UX, FreeBSD, SCO, SGI, IRIX, Digital Unix and Solaris. The GCC compiler is also available under Microsoft operating systems via Cygwin project.

### **3.2.2 Intel Compiler**

The most recent release of the Intel C++ compiler is version 7.1 and it supports both the Windows and Linux operating systems. The Intel C++ compiler supports only the Intel processor architecture including the x86, Itanium and XScale processor lines.

### **3.2.3 Microsoft Visual C++ .NET**

The Visual C++ compiler is only available for systems running the Windows operating system and the Intel or AMD x86 Pentium class of processors. It is geared to support the .NET initiative from Microsoft which aims to allow internet capabilities to be easily incorporated into Windows applications.

### **3.2.4 Fujitsu Compiler**

The Fujitsu C/C++ Express V2.0 compiler suite is targeted for the Intel x86 architecture running Linux and the Sun SPARC architecture running the Solaris operating system.

### **3.2.5 IBM VisualAge**

VisualAge C++ V6.0 is an advanced C/C++ compiler developed by IBM. It runs under the AIX and Linux operating systems. The VisualAge C++ compiler supports the x86, RS/6000, PowerPC and Power4 processor architectures.

---

<sup>6</sup>A partial list of supported operating systems includes: Linux, FreeBSD, Solaris, HPUX, Darwin and Microsoft Windows.

### 3.2.6 Summary

The capability to support a wide range of processors is an important attribute for the ALS project since the program defines a distributed hardware architecture that currently uses multiple different processor types[1]. Only one compiler among those surveyed has the flexibility to support multiple processor architectures and this is the GNU C/C++ compiler. The GNU C/C++ compiler is also the only compiler that scales from the small embedded micro-processors all the way to high-end, number crunching processors. The GNU C/C++ compiler gives the ALS project the ability to use most, if not all, of the processors/microprocessors that commonly in use at this time. The Intel and Microsoft compilers are limited to x86 compatible processors<sup>7</sup>. The Fujitsu compiler targets the x86 and Sparc processor architectures, while the IBM compiler is focused on the x86 and PowerPC processor architectures. None of the other compilers surveyed have the flexibility and scalability that is found with the GNU C/C++ compiler. While the GNU C/C++ compiler has been specified for general use, the use of other compilers would be acceptable if their application falls outside the core competence of the ALS project.

## 3.3 Operating Systems

The selection of an operating system is a difficult task. The ALS project defines a scalable and flexible architecture and these two requirements are traits that are difficult to find in operating systems. The operating system interacts directly with the hardware of the processor, which results in a tendency to make operating systems processor centric. Programming languages are insulated from the processor hardware via the compiler and thus they are easy to make universal in nature. Compilers must understand the architecture of a processor in order to be able to translate a high level language into machine code. While this process of converting the abstract high level language to machine code is not a simple endeavor it does not have to deal with the intricacies of dealing directly with hardware. The processor underlying the operating system has hardware and timing related issues and in order for the operating system to be portable it must deal with these issues in a manner that minimizes their significance.

Operating systems can define a Hardware Abstraction Layer (HAL) to isolate the operating system from the details of the processor's hardware. The HAL consists of code that is architecture, platform and implementation specific. It abstracts processor specific items like: interrupt delivery, context switching, CPU startup, platform startup, timer devices, I/O register access, interrupt controller and on-chip devices. When porting the operating system from one processor architecture to another, the developer concentrates on the HAL code. Once the HAL layer is ported the conversion of the remaining operating system code is straight forward.

---

<sup>7</sup>It should be noted that the Intel XScale processor is not x86, but is a license for the StrongARM architecture from ARM.

For the reasons explained under Section 2.2.3 the ALS project will use the Open Source model as a key factor when selecting an operating system. This restriction means that proprietary operating systems such as Microsoft Windows derivatives or the real-time Wind kernel from VxWorks are excluded from the list of potential candidates. The list of open source operating systems that were reviewed is summarized in Table 1.

Operating System	Type	Comments
Linux	General Purpose	The original Open Source Unix like OS
FreeBSD	General Purpose	A free implementation of the Berkeley Unix
RTAI	Real-time Extensions	Runs Linux as a processor under a real-time kernel
RTLinux	Real-time Extensions	Runs Linux as a processor under a real-time kernel
Timesys	Real-time Extension	Modifications to the Linux kernel adding real-time capabilities
Montavista Linux	Real-time Extension	Modifications to the Linux kernel adding real-time capabilities
Ecos	Embedded Linux	Scaled down version of Linux for embedded systems
RTEMS	Real-time	An open source hard real-time OS

*Table 1: Candidate Operating Systems*

### **3.3.0.1 Soft and Hard Real-time Capabilities**

Real-time operating systems are often split into two distinct categories, those with soft real-time characteristics and those who meet hard real-time deadlines. Hard real-time systems feature deterministic response times throughout all their capabilities. The interrupt, context switch, and scheduler latencies are all deterministic in nature with an upper end that is bound by a maximum response time that will never be exceeded. Soft real-time systems on the other hand feature average latency times that are responsive in nature, but their maximum latencies may exceed the average latency values by a couple of orders of magnitude. Hard real-time operating systems are essential where the missing of a deadline will have drastic consequences, where as soft real-time operating systems are useful where responsiveness is a desirable quality, but the missing of a deadline does not lead to serious consequences. Systems that have a multitude of interactions with external devices, such as encoders and motors, are better served by a hard real-time operating system.

### **3.3.0.2 Operating System Portability**

POSIX stands for the Portable Operating System Interface and is a standard that is being jointly developed by the IEEE and The Open Group. It defines a standard operating system interface and environment. The goal of the POSIX standard is to allow software to be portable across differing operating systems. When POSIX



compliant functions are used they allow code to become portable from one POSIX compliant operating system to another.

### **3.3.0.3 Operating System Evaluation**

The key criteria with respect to the evaluation of the operating systems will be:

- The acceptance and long term viability.
- The size of the user base.
- The support for multiple processor architectures.
- The support of hard real-time operation and/or the support of soft real-time operation.
- Support of third party applications<sup>8</sup>.

The following sections discuss each operating system in detail.

### **3.3.1 Linux**

Linux is a very popular open source clone of the Unix operating system. Linux was created by Linus Torvalds<sup>9</sup> in the early 1990s and Linux is considered the poster child for a successful open source project. Standard Linux supports multiple processor architectures including: Intel x86, AMD x86, PowerPC and Sun's Sparc. Linux boasts a huge community of developers who advance and maintain the kernel. Estimating the market share for Linux is difficult to derive since it is free to download and a single download may correspond to multiple installations[6]. Surveys have shown that in 2001 Linux servers held between 8.6% and 27% of the office server market depending on the who conducted the survey<sup>10</sup>. A 2003 survey of small and mid-size business found that 19% of businesses use an open source operating system on the desktop[7].

Linux is renown for its reliability and its highly configurable kernel. Reliability translates into infrequent system crashes and reconfigurability allows the operating system to be tailored to the requirements of an UGV. It supports the POSIX 1003.1 standard and has partial compliance to the 1003.1b standard. A huge variety of third party applications are available for Linux.

---

<sup>8</sup>This refers to applications such a graphical interfaces, GUI development tools, communications packages, etc which all relate to the ease of use and development of software.

<sup>9</sup>With the assistance of developers around the world.

<sup>10</sup>A 2001 Gartner Group survey estimated the Linux share at 8.6%. The 2001 IDC survey estimated Linux at 27%, Windows 41% and Netware 17%.

### 3.3.2 FreeBSD

FreeBSD<sup>11</sup> is in many ways similar to Linux. It is an open source implementation of the Berkeley Unix operating system. It supports many of the same tools and applications that are used under Linux such as X Window, Apache and the GNU C/C++ compiler. While Linux and FreeBSD are similar in many ways, their development models differ. The Linux development model is open and allows input from a large and diverse set of developers. FreeBSD on the other hand has a more closed development model. This development model has resulted in a smaller developer's community than the one associated with Linux and as a result the BSD support for peripheral devices is significantly inferior to that of Linux. Due to their unix heritages, most applications that are available for Linux are also available for FreeBSD.

### 3.3.3 RTLinux and RTAI

Both RTLinux and RTAI use a sub-kernel approach to imparting real-time capabilities on the Linux operating system. The sub-kernel approach creates a small sub-kernel that is fully preemptive and thus real-time responsive. This sub-kernel is a small operating system that provides the real-time capabilities for the Linux kernel. The Linux kernel is simply a special low priority process that is scheduled to periodically run. All interrupts are controlled by the sub-kernel. Communications between the sub-kernel and the Linux occur through special communication channels. The sub-kernel environment implements only a very basic low level functionality and does not support standard C or math libraries. It also does not support standard Linux drivers or file systems.

The basic philosophies of both these operating systems are that only time critical tasks are implemented at the sub-kernel level, while a majority of the application runs as an ordinary program under Linux user space. The major impediment with this approach is the difficulty of writing device drivers at the sub-kernel level. This approach supports a sub-set of the CPU's and platforms supported by Linux including the following: X86, PowerPC, Alpha and MIPS.

Software development for these RTOS's is accomplished using the GNU tool chain. The C language is the primary software language for software development at the real-time sub-kernel level. Almost any other language could be used for the development of applications to run in Linux user space. Under the Linux user space all third party Linux applications are available. RTLinux and RTAI service niche markets and do not have large, developed user bases. The verdict is still out as to where this approach will become widely accepted by mainstream users of Linux.

---

<sup>11</sup>FreeBSD, NetBSD and OpenBSD are all related variants of Berkeley Unix.

### 3.3.4 MontaVista Linux and Linux 2.6 Kernel

MontaVista extends preemptibility to the Linux kernel through the use of the SMP<sup>12</sup> locking mechanism that is available in 2.4 or greater kernel version. The SMP locking mechanism was incorporated into the Linux 2.4 kernel to allow the kernel to operate in multi-processor configurations. The SMP lock allows the kernel to protect critical segments of code or memory from unauthorized access by a different processor of the SMP configuration. MontaVista extended the use of the SMP lock mechanism to the uniprocessor configuration and added additional preemption checks to the SMP lock and interrupt processing, thus creating a more responsive Linux kernel. The SMP lock mechanism indicates whether or not preemption is permissible. When an interrupt occurs or a SMP lock is released the scheduler checks the status of the SMP lock counter. If it has a value of zero then preemption is allowed but if the value is non-zero then preemption is disabled. When preemption is disabled all processes including those with a high priority must wait. MontaVista also makes small modifications to the Linux scheduler to improve its performance.

The MontaVista approach is implemented as a patch to the existing Linux kernel and thus allows almost all of parts of the Linux operating system to benefit for these enhancements. Linux drivers gain the most benefit from this added preemptibility. If a driver is SMP safe then it will automatically accrue this preemption capability. Drivers that are not SMP safe may not function correctly with the MontaVista patch to the Linux kernel. In accordance with the GNU Public License (GPL) principles MontaVista has made these changes available to the general Linux community. Linus Torvalds has included the MontaVista patch into the next release of the Linux kernel and thus these enhancements are available in all versions of the kernel later than version 2.5.4-pre6. The 2.5 series of kernels are alpha releases. The next stable version of the Linux kernel that will incorporate the MontaVista enhancements will be the 2.6 version of the kernel.

MontaVista Linux, given that it is supplied as a patch to the regular Linux kernel, is capable of running on all the platforms that are currently supported by Linux including the following: AMD, ARM, Hitachi SH, PowerPC, Intel x86 and MIPS. For software development MontaVista uses the same standard tool chain used by generic Linux and all Linux applications will run.

### 3.3.5 TimeSys Linux

TimeSys uses an approach to enhancing the responsiveness of the Linux kernel that is similar to the technique implemented Montavista. The TimeSys changes build upon the existing SMP locking mechanism but the TimeSys

---

<sup>12</sup>SMP - Symmetric Multi-Processing

implementation is significantly more sophisticated than the MontaVista patch. The TimeSys enhancement replaces the SMP lock mechanism with kernel mutex locks. While a SMP lock disables all preemption, a mutex lock is a mechanism for controlling access to shared resources and thus does not disable all kernel threads and processes. With a mutex lock only the kernel threads or processes that are vying for access to the same resource are affected by the mutex lock. This nuance is best explained by way of an analogy provided by Timesys. “As an analogy, in a city with many traffic lights, using a spinlock or disabling preemption to ensure that only one car is in any intersection at any time would freeze all traffic in a city whenever any car entered any intersection. In contrast, using a mutex would only stop traffic from entering the specific intersection that the car is passing through. Traffic would continue to move everywhere else.”[2] While the TimeSys enhancements to the Linux kernel are theoretically superior to the changes championed by MontaVista, the TimeSys implementation has significant implications in terms of its compatibility with existing Linux device drivers. These implications are evident in the TimeSys release for the Embedded Planet EP405 board where the TimeSys kernel broke support for the PCI bus and the USB bus.

The Timesys patch is available under the GPL though unlike the MontaVista patch there does not seem to be any momentum to include the TimeSys changes into future releases of the Linux kernel. The commitment of TimeSys Inc. to the principles of the Open Source community is unclear. Though TimeSys has made its source code available with the distribution of its version of the Linux kernel, it has not made these changes readily available on the internet.

TimeSys Linux runs under a wide variety of architectures, including: AMD, ARM, Hitachi SH, PowerPC, Intel x86/xScale, Sparc and MIPS. As with all Linux based systems TimeSys uses the standard GNU tool chain and supports all standard Linux applications<sup>13</sup>. The TimeSys commercial model directs all support to a paid TimeSys programs. They do not seem to support or encourage input from external sources and this makes it difficult to judge TimeSys’s market acceptance. The best clue about TimeSys’ acceptance is found in its lack of acceptance by Linus Torvalds. Linus has chosen to incorporate MontaVista’s kernel modifications into the 2.6 kernel of Linux and has passed on using the TimeSys approach.

### **3.3.6 eCos**

eCos stands for the “Embedded Configurable Operating System” and is an open source, royalty-free, license-free, real-time operating system for deeply embedded applications. It is partially supported by RedHat and has a strong

---

<sup>13</sup>Assuming these applications are not dependent upon a device driver and hardware.

developer community. eCos has been designed to support applications with real-time requirements, providing features such as full preemptibility, minimal interrupt latencies and all the necessary synchronization primitives, scheduling policies and interrupt handling mechanisms needed for these types of applications. eCos supports a wide range of target architectures including: ARM, Hitachi SH3/4, MIPS, NEC V850, Panasonic AM3x, PowerPC, SPARC and x86.

The GNU tool chain is the standard development environment for eCos. The support for third party applications under eCos is limited. This is mainly due to the processors that eCos targets which are limited in processing power and memory. eCos is a relatively new entry that has somewhat of a checkered past. It was originally developed by Cygnus, who was then acquired by RedHat. RedHat dropped official support for eCos and has instead given eCos to the user community. How well the eCos user community will fair in the long run as an unknown at this point.

### 3.3.7 RTEMS

RTEMS stands for the “Real Time Executive for Multiprocessor Systems”. It was originally developed by OAR Corp. for the U.S. DoD<sup>14</sup>. It eventually followed the open source model and is now licensed under a GPL variant. Even though RTEMS is now an open source operating system the company that developed it, OAR Corp., is still actively involved with enhancing this operating system.

Unlike a majority of the operating systems reviewed in this section, RTEMS is a true real-time operating system (RTOS) that has been designed from the ground up with real-time capabilities. It is targeted for embedded systems that may be memory limited in nature. It has a modular design that allows the operating system to be compiled with a varying levels of capabilities<sup>15</sup>. The pure real-time design of RTEMS allows all system services and libraries to be directly available to any application task. RTEMS is compliant to the POSIX 1003.1b API, the uITRON 3.0 API as well as its own RTEID/ORKID based classic API. Additionally, RTEMS has been certified for use in military systems by the U.S. DoD. This certification could be a potential benefit if/when UGV's are enabled to use lethal force.

RTEMS supports a wide range of CPU architectures including: Motorola 68k, ColdFire, Hitachi SH, Intel i386, Intel i960, MIPS, PowerPC, SPARC, AMD A29k and HP PA-RISC. This wide range of supported architectures is a testament to RTEMS portable design which eases its porting to new architectures.

---

<sup>14</sup>At one time the M in RTEMS signified Missile, then was changed to signify Military before finally becoming Multiprocessor.

<sup>15</sup>For example if an ethernet interface is not available RTEMS can be compiled without networking support.

RTEMS uses the GNU Tool Chain with the primary programming languages being C/C++ and Ada. Like eCos the list of available third party applications is limited for RTEMS. RTEMS has a significant user base that includes mainly military and research type institutions<sup>16</sup>. With RTEMS having a core client with the U.S. military its use in both military and non-military applications should continue to grow.

### 3.3.8 Linux Operating System Performance

Various studies have been undertaken to quantify the performance of the standard Linux kernel as well as the performance of the competing enhancements to the kernel that make it more responsive. This performance is usually measured in terms of the average latency and the maximum latency of Linux processes<sup>17</sup>. Latency under Linux usually refers to the scheduler latency since it is this latency that dominates all others. Scheduler latency is the time between a wakeup signaling that an event has occurred and the kernel scheduler getting an opportunity to schedule the thread that is waiting for the wakeup to occur.

Benchmarks conducted by TimeSys compare the performance of the TimeSys enhancements to various other Linux configurations such as standard Linux<sup>18</sup>. Table 2 shows the results of these tests.

Operating System	Average Latency (us)	Maximum Latency (us)
Standard Linux	<10,000	100,000
Linux with Preemptible Kernel	<1,000	100,000
TimeSys Linux GPL	<50	1,000
TimeSys Linux/Real-Time	<10	51

*Table 2: Kernel Preemption Latency Comparison[2]*

The Linux scheduler was bench-marked by RedHat in the spring of 2002 to compare the performance of two different lower latency patches with the standard Linux kernel. These benchmarks were performed for the 2.4.17 kernel on an AMD 700 MHZ Duron system with 360 MB RAM and a 20 GB Western Digital IDE drive. The tests were performed while stressing the Linux kernel by simultaneously executing six different stressing applications. The tests were executed for a period of 41 minutes. Table 3 shows the benchmarks derived by the RedHat investigations.

For these benchmarks the preemptive patch is the same as the MontaVista

<sup>16</sup>RTEMS is use extensively the physics community in controlling high enery particle accelerators.

<sup>17</sup>Interrupt latency on recent x86 hardware in in the order of 10us, the interrupt handler latency is in the low 100's us, while the scheduler has a latency in the realm of a few microseconds[3].

<sup>18</sup>For this comparison TimeSys did not define the kernel revision used, the hardware platform nor their definition of kernel preemption latency.

Metric	Standard Linux	Preemptive Patch	Low Latency Patch
Ave. Latency ms	0.0883	0.0529	0.0543
Max. Latency ms	232.6	45.2	1.3
<0.1 ms	92.84%	97.95%	96.66%
<0.2 ms	97.08%	99.56%	99.21%
<0.5 ms	99.73%	99.97%	99.996%
<0.7 ms	99.84%	99.99%	99.9998%
<1 ms	99.94%	99.997%	99.99999%
<5 ms	99.97%	99.9995%	
<10 ms	99.98%	99.9998%	
<50 ms	99.986%		
<100 ms	99.988%		

**Table 3:** RedHat Benchmarks of the Linux Scheduler[3]

changes to the Linux kernel. The low latency patch was introduced by Ingo Molnar and focuses on introducing explicit preemptions points into blocks of code where the kernel may execute for long stretches of time. While the MontaVista approach is kernel independent, the low latency patch requires a detailed profiling of each kernel revision and tweaking of the kernel code.

The TimeSys and RedHat benchmarks show that modifications can be made to the Linux kernel that significantly improve its performance. The MontaVista preemptive patch, the low latency patch and the TimeSys enhancements improve the performance of the Linux kernel such that it is eminently usable under soft real-time conditions. The RedHat benchmarks document the exact configuration used during the tests but unfortunately the TimeSys results make no reference to either the kernel versions used, the hardware platform or the stress testing placed on the Linux kernel. As a result it is not easy to make comparisons between the two sets of benchmarks. While TimeSys Linux seems to have the advantage of a lesser maximum latency, the average latency of all these three approaches seem to be very similar<sup>19</sup>. The MontaVista patch has the advantage that it has been given the nod by Linus Torvalds to be included in the next release of the Linux kernel, which will be version 2.6.

### 3.3.9 Summary

This section has reviewed a selection of open source operating systems. The ALS project at DRDC Suffield requires an operating system(s) that will compliment the scalability and flexibility of the corresponding UGV Electronic Hardware Architecture [1]. The above survey shows that it is not possible for a single operating system to meet all the requirement listed in the evaluation criteria. Operating systems that suitable for hard real-time

<sup>19</sup>Since TimeSys did not publish their benchmark configuration this statement can not be made conclusively.

applications, on small embedded micro-processors, do not have a large pool of third party applications available. Conversely, operating systems that have a large based of third party applications are not well suited for hard real-time operations. This paradox is not surprising since general purpose operating systems, which have a large pool of third party applications, cater to largest user group which is the realm of the desktop computer user. While on the other hand, pure real-time time systems target an audience where performance is the key factor and applications are less significant. The RTLinux and RTIA operating systems try to bridge this divide by offering both real-time performance with access to all the capabilities and applications associated with a general purpose operating system. The major deficiencies of this approach is the limited support processor architectures, the difficulty of using the real-time sub-kernel and the lacked of compatibility with standard Linux device drivers<sup>20</sup>. Timesys Linux attempts to bridge this divide by converting Linux into a true real-time, pre-emptible operating system. While this approach is theoretically the most appealing, the difficulties in converting a monolithic general purpose kernel to a responsive real-time kernel are large. The Timesys approach has introduced incompatibilities with existing Linux device drivers which has thus far limited its appeal. While Timesys has had success, Linus Torvalds has opted to use approach pioneered by MontaVista Linux which probably means that the Timesys modifications to the Linux kernel will never enter the main stream of use.

Given the distributed nature of the UGV Electronic Hardware Architecture[1] it is feasible to implement a mixture of soft and hard real-time operating systems, using the hard real-time capabilities only where required and using soft real-time capabilities for less time sensitive applications. This approach allows UGV's to benefit from both the real-time performance and the ease of use perspectives. It also allows for the selection of operating systems that are optimum for the roles they will perform.

The UGV Electronic Hardware Architecture has a requirement for hard real-time capabilities at the control layer where software is interacting, in real-time, with external devices. At this layer the RTEMS operating system will be used. RTEMS has all the capabilities that will enable software to respond to external devices in a deterministic and real-time manner. The design of the RTEMS kernel is such that it is fully preemptible thus giving it latencies that are amenable to the interactions with external devices. RTEMS also has a military heritage that could eventually a useful asset for the ALS project. RTEMS was chosen over eCos mostly due to its long history of stable use and its established user community.

At the High Level and Intermediate Level Intelligence Layers of the UGV

---

<sup>20</sup>In clarification standard Linux device drivers operate but they execute under the Linux thread, and thus they do not meet real-time constraints.



Electronic Hardware Architecture, where only soft real-time capabilities are required, the ALS project will use standard Linux. If the Linux 2.4 kernel version is used, the MontaVista preemptive patch will be applied. When the Linux 2.6 kernel becomes available the ALS project will standardize on it since it will incorporate the preemptive patch that was pioneered by MontaVista.

The use of the POSIX standards will assist the ALS project in achieving the portability, flexibility and scalability that is required to allow the program to seamlessly support multiple UGV platforms. RTEMS completely complies to the POSIX 1003.1b standard while Linux fully supports the 1003.1 standard and partially supports the 1003.1b standard. Patches are available for Linux that enhance its support of the 1003.1b standard. By using the POSIX API the process of moving code between the Linux and RTEMS operating systems will be a fairly seamless process that will not require a significant amount of effort. Software developed for the ALS project will adhere to the POSIX standard where possible and feasible.

## 4. Conclusions

---

This document has described the software development environment that will be used by the ALS project for developing autonomous unmanned ground vehicles. The software development environment defines five key aspects of the software development process:

1. The ALS project supports the Open Source Model and will use Open Source tools wherever possible.
2. The C and C++ languages will be used for all software development related to the core competence of the ALS project.
3. All software will use the GCC tool chain for compiling the high level languages into the machine code instructions that are used at the processor level.
4. The Linux operating system will be used for software that has soft real-time or lesser constraints. For software that interacts with external devices that impose hard real-time restraints the RTEMS operating system will be used.
5. All software developed will adhere to the POSIX 1003.1 and 1003.1b extensions wherever possible.

This software development environment defines a set of standards, tools, languages and applications that are well suited to meet the needs of the ALS project. The proposed environment defines a model for developing software that protects DRDC's investment in the intellectual property that the software represents while maintaining the flexibility

and scalability that is required for compatibility with UGV Electronic Hardware Architecture.

This software development environment will allow the software developed under the ALS project to be used across a range of UGV platforms that feature diverse requirements. It will allow the researchers working on the ALS project to concentrate on developing algorithms that enable and enhance UGV autonomy while having minimal concerns about algorithm compatibility between the various types of UGV platforms.

## References

---

1. Broten, G. and Monckton, S. (2004). Unmanned Ground Vehicle Electronic Hardware Architecture. Technical Memorandum. DRDC. Suffield TM 2004-122.
2. Locke, D. (2002). A TimeSys Perspective on the Linux Preemptible Kernel. No. V1.0. TimeSys Corporation.
3. Williams, C. (2002). Linux Scheduler Latency. Technical Report. Red Hat Inc.
4. Dudek, G. and Jenkin, M. (2000). Computational Principles of Mobile Robotics, Cambridge, UK: Cambridge University Press. p. 10.
5. Lutz, M.H. and P.A., Laplante (2003). C# and the .NET Framework: Ready for Real Time? IEEE Software Real Time Systems - <http://www.computer.org/software/homepage/2003/s1lap.htm?SMSESSION=NO>.
6. Petrele, N. (2003). Debunking the Linux-Windows market-share myth. LinuxWorld - <http://www.linuxworld.com/story/32648.htm>.
7. Gonsalves, A (2003). Microsoft Faces Linux Threat in SMB Market. Internetweek - <http://www.internetweek.com/story/showArticle.jhtml?articleID=12800636>.



Unlimited

<b>DOCUMENT CONTROL DATA</b>		
<small>(Security classification of title, body of abstract and indexing annotation must be entered when document is classified)</small>		
1. ORIGINATOR (the name and address of the organization preparing the document. Organizations for whom the document was prepared, e.g. Centre sponsoring a contractor's report, or tasking agency, are entered in section 8.)  <b>Defence R&amp;D Canada – Suffield PO Box 4000, Medicine Hat, AB, Canada T1A 8K6</b>	2. SECURITY CLASSIFICATION (overall security classification of the document including special warning terms if applicable).  <b>UNCLASSIFIED</b>	
3. TITLE (the complete document title as indicated on the title page. Its classification should be indicated by the appropriate abbreviation (S,C,R or U) in parentheses after the title).  <b>Unmanned Ground Vehicle Software Development Environment</b>		
4. AUTHORS (Last name, first name, middle initial. If military, show rank, e.g. Doe, Maj. John E.)  <b>Broten, G. ; Verret, S. ; Digney, B.</b>		
5. DATE OF PUBLICATION (month and year of publication of document)  <b>June 2004</b>	6a. NO. OF PAGES (total containing information. Include Annexes, Appendices, etc).  <b>33</b>	6b. NO. OF REFS (total cited in document)  <b>7</b>
7. DESCRIPTIVE NOTES (the category of the document, e.g. technical report, technical note or memorandum. If appropriate, enter the type of report, e.g. interim, progress, summary, annual or final. Give the inclusive dates when a specific reporting period is covered).  <b>Technical Memorandum</b>		
8. SPONSORING ACTIVITY (the name of the department project office or laboratory sponsoring the research and development. Include address).  <b>Defence R&amp;D Canada – Suffield PO Box 4000, Medicine Hat, AB, Canada T1A 8K6</b>		
9a. PROJECT OR GRANT NO. (if appropriate, the applicable research and development project or grant number under which the document was written. Specify whether project or grant).	9b. CONTRACT NO. (if appropriate, the applicable number under which the document was written).	
10a. ORIGINATOR'S DOCUMENT NUMBER (the official document number by which the document is identified by the originating activity. This number must be unique.)  <b>DRDC Suffield TM 2004-060</b>	10b. OTHER DOCUMENT NOS. (Any other numbers which may be assigned this document either by the originator or by the sponsor.)	
11. DOCUMENT AVAILABILITY (any limitations on further dissemination of the document, other than those imposed by security classification) <input checked="" type="checkbox"/> Unlimited distribution <input type="checkbox"/> Defence departments and defence contractors; further distribution only as approved <input type="checkbox"/> Defence departments and Canadian defence contractors; further distribution only as approved <input type="checkbox"/> Government departments and agencies; further distribution only as approved <input type="checkbox"/> Defence departments; further distribution only as approved <input type="checkbox"/> Other (please specify):		
12. DOCUMENT ANNOUNCEMENT (any limitation to the bibliographic announcement of this document. This will normally correspond to the Document Availability (11). However, where further distribution beyond the audience specified in (11) is possible, a wider announcement audience may be selected).		

Unlimited

Unlimited

13. ABSTRACT (a brief and factual summary of the document. It may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall begin with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (S), (C), (R), or (U). It is not necessary to include here abstracts in both official languages unless the text is bilingual).

The Autonomous Intelligent Systems (Land) initiative has been tasked with researching and developing innovative autonomous vehicles that will assist the Canadian Forces in performing their duties in the 22nd century. This research will continue for many years on many different types of unmanned ground vehicles. Thus it requires a software development environment that will protect what will become a substantial investment in intellectual property that is implemented in software algorithms. This environment must be based upon standards that are as universally supported as possible. The software development environment must also be compatible with the Autonomous Land Systems (ALS) project Unmanned Ground Vehicle Electronic Hardware Architecture. This report describes the software development environment developed for the ALS project. The environment is based upon open standards and meets the diverse requirements of the project. This environment is currently being used by all applications being developed for unmanned ground vehicles (UGV) at DRDC Suffield under the ALS project.

14. KEYWORDS, DESCRIPTORS or IDENTIFIERS (technically meaningful terms or short phrases that characterize a document and could be helpful in cataloguing the document. They should be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location may also be included. If possible keywords should be selected from a published thesaurus. e.g. Thesaurus of Engineering and Scientific Terms (TEST) and that thesaurus-identified. If it not possible to select indexing terms which are Unclassified, the classification of each should be indicated as with the title).

Unmanned ground vehicles,  
robotics,  
autonomous vehicles,  
software development environment

Unlimited