European ALMA Software Advisory Group

ALMA Memo #285

Choice of Real-Time Operating System for ALMA

P.T.Wallace
Rutherford Appleton Laboratory, UK
7 February 2000

---

1 Introduction

ALMA computing requirements span almost the entire available range of technologies, from custom chips to supercomputers. Consequently, we foresee a wide variety of machines and operating systems being used; there is clearly no serious prospect of adopting a single platform for all ALMA purposes. However, there is a category of ALMA application, which includes antenna and correlator control, that traditionally is implemented using a *real time operating system* (RTOS); it may at least be possible to agree on just one RTOS for the whole project.

Early discussions touched upon several possibilities, including Windows CE, eCos, LynxOS and OS9, but the only serious contenders to emerge were VxWorks and RT-Linux. Opinions were polarized between these two choices.

VxWorks, a proprietary RTOS from Wind River Systems Inc., has long been regarded as the premier development and execution environment for complex real-time and embedded applications on a wide variety of target processors. The system comprises (i) a high-performance scalable RTOS which executes on a target processor and (ii) a set of powerful cross-development tools which are used on a host development system. Various communications options are supported to link the host to the target. A typical configuration is a PowerPC as the target, connected by Ethernet to a Sun workstation acting as host. VxWorks contains support for a wide range of hardware.

Linux is a free and open-source implementation of Unix for x86 and Pentium computers. It supports a wide range of software, and for scientific data-analysis purposes a PC running Linux competes head-on with traditional Unix workstations such as Sun. Ports to non-x86 machines such as Alpha, SPARC and PowerPC also exist. **RTLinux** is a version of Linux that makes the Linux kernel act as the lowest-priority task of a much simpler real-time kernel. The RT kernel provides its other tasks and real-time interrupt handlers with scheduling and interrupt response times that are close to the limits of the underlying hardware. The RT kernel provides a basic set of mechanisms for the non-RT processes running under the Linux kernel to

communicate with the real-time components. Hardware support is patchy but growing fast.

VxWorks is expensive (and is typically used with expensive, “industrial strength”, hardware), but is a mature product and is generally agreed to be the safe bet in the short and medium terms. Long-term support depends on Wind River Systems or other commercial companies. RTLinux is free but still evolving; Linux itself appears unstoppable, and prospects for long-term support of RTLinux seem good.

VxWorks is used extensively by both NRAO and ESO, and both groups naturally feel comfortable recommending it be used for ALMA. This opinion is shared with others in the community who are already using the OS. However, a rival faction in the software discussions strongly favoured drawing a line under the VxWorks era and adopting RTLinux. The rest of this report reflects this split, and addresses the simple question: “Which should ALMA adopt: VxWorks or RTLinux?”

2 NRAO requirements

NRAO staff have provided a list of requirements for an ALMA RTOS. They point out that VxWorks has been the RTOS of choice at NRAO and ESO for many years. While acknowledging that its relatively high price and the unavailability of source code are important disadvantages, they point out that VxWorks has many strong features. Their list of requirements is to a large extent a compilation of those VxWorks features that they feel offer reasonable criteria to judge Linux (and, for that matter, any RTOS) for ALMA use.

2.1 RTOS requirements

- The RTOS must be able to have many (hundreds) of independent execution threads (tasks) with pre-emptive priority scheduling. The RTOS must possess communication facilities that allow the tasks to synchronize and coordinate their activities. The RTOS must be able to switch between the tasks quickly, on the order of 10 microseconds or less.

- The RTOS must have low interrupt latency. This is the time between posting an interrupt and when execution of the Interrupt Service Routine (ISR) begins. This should be less than 5 microseconds.

- The RTOS must include a complete I/O system providing access to all commonly used devices.

- The RTOS must provide network facilities to include Unix sockets, Remote Procedure Calls (RPCs), and Network File System (NFS).

- The RTOS must run on all popular processor types.

- The RTOS should have POSIX compatibility.

2.2 Development environment requirements

- The RTOS development environment must include a debugger with the following features:

    - A Graphical User Interface (GUI) with buttons for common debug activities.
    - A command-line interface for more complex and unpredictable needs.
    - The GUI can show a selected bit of the source code.
    - Breakpoints that can be set/removed by pointing to the source code lines.
    - Conditional breakpoints.
    - A simple way of displaying the current value of a selected variable.
    - Able to display structure, array, and container contents intelligently.
    - Support for debugging interrupt code.
    - C++-aware, e.g. name mangling and exceptions.
    - Capable of being run remotely, either over a TCP/IP network or serial line.

- The RTOS development environment should provide a mechanism whereby when the compiler finds an error the editor is started with the error line in view. (This can be done with EMACS.)

- The RTOS development environment should provide a graphical, hierarchical view of the full application and libraries to include source files, header files, and library components.

2.2 Antenna system requirements

These are presented as a guide to the tasks to be performed by the system using the RTOS.

The ALMA antenna systems will have about 200 control points, and 800 monitor points determined by a rough system design for major components. The engineers are now planning to provide monitor points for voltages and temperatures throughout the system, which is expected to at least double the number of monitor points.

The monitor points are in three different categories: time critical, medium, and slow. Two time critical monitor points are sampling the total power at rates up to 1 kHz, and collecting antenna positions at 20 Hz. Most of the remaining known monitor points will be collected about once per second. The voltage and temperature monitor points need to be sampled about every 5 minutes.

The control points can be categorized as either time critical or not. The known time critical control points are setting the antenna position at 20 Hz, and commanding of events synchronized by hardware signals.

There are two hardware signals used for synchronization. A 20 Hz tick used to synchronize fringe rotation and delay line models, and a 10-20 millisecond tick derived from the correlator chips' readout. The 10-20 millisecond tick is used to synchronize the receiver signal phase switching and the FIR filter personality.

The current system topology is an embedded, diskless computer at each antenna.

There is a CAN bus for communication between the antenna computer and the local devices, and an ATM network in a star configuration between antenna computers and the central control building. ATM circuits will be created from the antenna computers to the central computer for monitor data, antenna positions, and total power data. A circuit will also be established for commands from the central computer to the antenna. The command circuit needs to be high priority to guarantee receipt of real-time requests.

The antenna computers have a prioritized CAN queue with time critical monitor and control requests getting highest priority. Monitor programs will read data from the devices, time tag the data, and put the data into buffers to be sent back to the central computer when requested. The total power is monitored at the highest priority and the antenna position is next, all other monitor points having lower priority. The intention is to use excess CAN cycles for low priority monitoring and error checking. Control points will generally have higher priority than normal monitor points.

The concerns to be addressed at the antenna computer are:

- Standalone operation of the antenna computer while the antenna is being relocated. The system image needs to be easily upgraded.
- Prioritized CAN messages; multi-user CAN messages; synchronize user with message done.
- Real-time monitor tasks, especially total power at 1 kHz and position monitor at 20 Hz, interacting with the CAN queue and putting data back to the central computer.
- Real-time receipt of commands from the control computer placed into the CAN queue.

3 Linux and the NRAO requirements

3.1 Different views of what an RTOS is for

It is traditional to begin comparisons of rival RT operating systems with a look at the sophistication of their schedulers, for example in respect of the facilities for resource-locking and dynamic priority control. In these areas, VxWorks offers comprehensive

support, and real-time applications of great complexity and subtlety can be developed. However, there is another school of thought.

Many RT applications, when properly analyzed, turn out to contain remarkably little time-critical functionality. The application is a real-time one, to be sure, but the truly time-critical capabilities, if indeed there are any, are embedded in a mass of routine software with no performance requirements that cannot be met using a conventional non-RT operating system. Because a full-blown RTOS is inevitably a specialized tool, using one means that large amounts of routine software have to be written in a non-standard way, subject to peculiar limitations (such as lack of access-violation tracks, or the requirement that only one copy of a function can be present). This is a bad thing.

RT-Linux has the great advantage, compared with VxWorks,w that the bulk of the application can be written using conventional Unix techniques, with all the usual access to data storage peripherals and networks, and the full choice of languages and utilities. Only very small parts of the application need to run in the RT kernel, minimizing any difficulties caused by the currently rather primitive development tools. There is no separate Unix host, and communication between the “host” facilities and the RT kernel does not involve network links. You can have a complete RTLinux application in one box, complete with its program development environment.

3.2 Does RTLinux meet the NRAO requirements?

On switching time and interrupt latency, there is no evidence that RTLinux produces significantly better or worse figures than VxWorks. Both are limited only by the hardware. Both activate user-written code without significant intervention by the OS (in contrast to some RTOS architectures, where large amounts of system code are run before user-written code is finally called).

Similarly, there are no reasons why RTLinux should not support hundreds of threads.

RTLinux meets the I/O requirements in that it includes a complete Unix I/O system. I/O direct from the kernel is a question of available RT device drivers, as it is for VxWorks. Similar remarks apply to the network facilities. RTLinux has a full range of capabilities, but from the Linux process level (arguably the right place).

The range of processors supported by RTLinux is less than for VxWorks, but those that are supported are mainstream ones.

RTLinux is POSIX compliant.

The development environment requirements (most of which are desiderata rather than requirements) are met (in spades) by RTLinux, with the possible exception of support for debugging interrupt code, where VxWorks is much stronger (at present).

Regarding the antenna system requirements, there are no obvious reasons why RTLinux should not be capable of supporting the application.

4. RTLinux now and in the future

4.1 Variants

The case for adopting RTLinux is not helped by there being two extant variants, one from the USA, which I shall call “RTL” and one from Italy, which I shall call “RTAI”. (The term “RTLinux” is used generically here.) At the time of writing, the latest releases are RTAI v0.7 and RTL v2.0. (Certain other variants, for soft real time and called KURT and RED-Linux, are dormant and possibly dead.) For the 2.0.x Linux kernels, you *have* to use RTL. For the 2.2.x Linux kernels, you can use either, but RTAI has edged ahead of RTL. The situation is very fluid and if the RTL v2.0 release is truly stable, the playing field is once again level.

In practice, there is nothing to stop endless new kernels, or variants of the old ones, appearing, just as there is nothing to stop a new Linux kernel being developed. However, it seems that no-one is developing a new variant: people are sticking with RTL or RTAI. Both have reasonably well defined development paths and users are happy to have all this done for them. Where people are developing code, it is to add to the functionality of RTL or RTAI, not detract from it.

How similar an API do they, or will they, offer? For the FIFOs through which the RT kernel communicates with Linux, the API is exactly the same (the RTAI code is same as the RTL code). For the task facilities, the native APIs are similar but not quite matching; the `rt_task_init` function, for example, has different numbers of arguments and in different orders. Other features are very different: IPC, RPC, semaphores etc. They both have a Posix `pthreads` interface which, by definition, should be common.

The conclusion is that you have to commit yourself to either RTL or RTAI. By the time ALMA has to make a decision it is likely that the choice will either have become clear or won’t matter. Sensitivity to Linux kernel changes is another concern, of course, the danger being that in tracking the evolution of RTLinux we could accumulate so many kernel debug and development tools that we can never remember which ones work with what kernel. However, these aspects will be addressed by the companies supplying the development environments. We could, for example, buy the Zentropix V1.1 CD now, which includes a real-time debugger. It would be foolish subsequently to upgrade the kernel without getting a new CD, which would contain tools compatible with the new kernel (and with existing program development practices).

4.2 I/O support

The Achilles heel of RTLinux is limited support for I/O hardware. VxWorks is a very mature product and has excellent board support (as often as not due to contributions

from their customers—the Bancomm 635 driver was written at ESO, for example). The current status of I/O hardware support in RTLinux is as follows.

The COMEDI project (http://stm.lbl.gov/comedi) provides support for some boards and there is support for common devices. Here is a list of what can be RT-driven at present:

- *Analog Device:* RTI800/815, RTI802
- *National Instruments:* AT-MIO E series, PCI-MIO E series, PCI-60xx series
- *Data Translation:* DT2801 series, DT2811, DT2814, DT2815, DT2817, DT2821 series, DT3000 series (in progress)
- *Keithley Metrabyte:* DAS-1601/1602
- *ComputerBoards:* CIO-DAS08, CIO-DAS08/JR-AO
- *Quanser Consulting:* MultiQ
- *PC LabCard:* PCL-711, PCL-711b, PCL-725, PCL-726
- *Generic:* Intel 8255, PC parallel port
- *Advantech:* ACL-8122

There is native support for serial and parallel I/O and there is now a real time ethernet driver.

Of course, the question of hardware support goes to the heart of Linux itself—not just the real time community. What happens when a new board arrives and there is no Linux driver? In practice, someone, somewhere, writes the driver pretty quickly and makes it freely available. If, however, one is in the awkward position of writing a driver oneself the LDDK (Linux Driver Development Toolkit) provides a template machanism and environment for developing such code. For example, there is one implementor known to be using the PLXTECH development boards and LDDK to build a variety of real time Linux devices on a PCI bus.

The LXRT tool, which was delivered with RTAI v0.6 and is being actively pursued by, for example, Zentropix, is a great boon here. If the board has open source code (as many now do), the job of converting the driver to real time can be done using the LXRT mechanism which allows you to develop code in user space (with the rich set of debugging tools) and re-compile them as RT modules when the driver works properly.

Of course, not all hardware requires a real time driver. Clearly, the separation of real time and user side processes allows some drivers (for less intensive tasks) to run in user space communicating with the real time side via FIFOs or shared memory. This is in contrast to VxWorks, where every I/O device used by the application must have a device driver.

4.3 Processor types

RTAI runs on the x86 platform only at present. RTL runs on x86 and PowerPC.

4.4 The future

The consensus in the RTLinux community is that in three years time:

- there will be only one real time Linux and it will be native to the standard vanilla kernel (in other words, non-RT Linux users will be running RTLinux without realizing it, except perhaps when building a kernel and a message along the lines of “Enable hard real time support? Y/M/N/?” appears);
- it will have a Posix `pthreads` interface (so the same code should be pretty much interchangeable between VxWorks and real time Linux); indeed, Zentropix are developing VxWorks compatibility libraries and pSOS compatibility libraries; and
- RTLinux will be better documented than it is now.

There is an abundance of anecdotal support for RTLinux. For example, industry contacts believe that real time Linux will be the *de facto* standard in flight simulators by 2001. NW Airlines are converting all their 23 (full size) flight simulators at their training centre in Minneapolis/StPaul to real time Linux. These simulators are fully FAA compliant so if this project succeeds, there will be no reason for other developers to stick with expensive, proprietary code. In US astronomy, SOAR, WTTM, 4mAPS and CHARA are all using real time Linux, and the list is sure to grow. Anecdotes about VxWorks, on the other hand, tend to be negative. Poor support is one common gripe—witness the `%f` bug that is still there years after being reported by many users. The users certainly make this a big gripe when mentioning VxWorks. There are also a lot of complaints about scalability: once the system gets big (tens of tasks) one stray pointer can crash everything.

5 Linux and RTLinux Case Studies

3.1 IRAM antenna controlled with Linux

Alain Perrigouard has reported on progress with using Linux as a “soft real time” OS for the control of Antenna #4 of the Plateau de Bure interferometer. The objective is to replace the existing VME processor, running OS9, by a Pentium VME processor under Linux to meet the same control requirements. Under Linux control, the antenna is to be used as a normal member of the interferometer. (IRAM are also developing a Linux autocorrelator system, at Pico Veleta.)

The processor, from VMIC, is a VMIVME-7592 board (100 MHz Pentium MMX with 32 Mbytes of RAM). For the PCI/VME bridge (Tundra Universe II), two drivers have been tested, from J.Hannappel (Bonn University and CERN) and G.Paubert (IRAM) respectively.

The control requirements are the following:

- The frequency of the main axes' servo loops is 64 Hz. (Checks are made that no interrupts are lost: with the 64 Hz interrupt there is an 8-bit counter which identifies each 64 Hz interrupt with respect to the 1 Hz interrupt.)

- The micro may receive commands at periodic intervals (1Hz) and should return the antenna status to the coordinating computer though ethernet and by means of task-to-task communications. The status is 344 bytes long. Commands have a maximum length of 188 bytes, but several may be combined to be sent at the same 1 s time slot. (The system has been designed in such a way the status of each antenna is collected every 1 s. Originally, the commands were sent in an asynchronous way. Later, it was found easier to combine the commands and to send them only at specific times and only once per second. But it should be stressed that commands are not time dependent: in particular, they do not depend on the Ethernet delivery delay.)

- The main commands the micro may receive are the initialization of the incremental encoders and the subreflector motors, the absolute and relative rotations of the antenna in astronomical (equatorial, ecliptic ...) or horizon coordinate systems, the configuration of the pointing model, homology and refraction parameters and the position of the subreflector.

- When the position is requested in an astronomical coordinate system the micro calculates the astronomical angles Az and El. This implies knowledge of the sidereal time.

- For any position request, the pointing, homology and refraction corrections are applied.

- The sidereal time is calculated from the universal time kept updated by the micro. At initialization time, the micro requests to a clock server on the net the UT of the next 1 s pulse delivered by a time bus connecting all VME chassis on Plateau de Bure. The transformation from astronomical to horizon coordinates is done every 1 s.

- Each time a source position is requested, the coordinates of the Sun are provided. These values are used to find the optimal path to reach the source, staying out of the sun by a certain minimum angle.

- The position of the subreflector motors is calculated from the received requests (focus, tilt and vertical translation) and corrected with the homology parameters. (In fact the subreflector motors are controlled by another VME micro installed in the receiver cabin, and requested and actual positions are passed through ethernet between the two processors.)

The trials have been completely successful. The fast (64 Hz) interrupt service routine lasts 6 μs and under normal conditions starts from 6 μs to 15 μs after the interrupt

time. This performance is independent of whatever else the CPU is doing, apart from certain exceptional cases, namely where the window manager or Netscape are starting up: under these circumstances, delays of up to 110 µs have been observed. This increased latency not a problem for the application in question; if it were, RTLinux could be used instead, and the effect would almost entirely go away.

3.2 Hexapod mount controlled with RTLinux

Roland Lemke (Ruhr-Universität Bochum) is developing a hexapod telescope mount controlled by RTLinux. The Hexapod Telescope (HPT) is mounted on 6 independent extendable struts. The idea itself is successfully used in many technical applications, mostly depending on hydraulics as used on "Steward" platforms. Due to the limited accuracy achieved by hydraulics, a new driving technique had to be developed which uses high precision roller screw linear drives with a mechanical accuracy of about 0.05 arcseconds per step. The primary mirror consists of a 55 mm thin Zerodur glass-ceramic meniscus which is glued to a lightweight Carbon-Fibre-Compound (CFC) structure. Its shape is permanently controlled by 36 piezoelectric elements, allowing an optimal optical configuration to be maintained. Ring Laser Gyros (RLGs) are used as an intermediate pointing device while the telescope is slewing from one position to the next. For technical reasons it is not possible to use the gyros during observations. In this mode, three guiding CCDs will be employed.

For system control, which includes as well the sub-reflector (another hexapod), a combination of a SIEMENS P5 and a PC 486 running MSDOS are used at present. The PC houses a Motion Control Board from Creonics which controls the motors of the six legs. The control for the active optics and the laser gyros uses RS485 interfaces. Timing information is acquired from a GPS clock over RS232 and an optional 1 pps is available.

For the first tests of the telescope, the supplied control system will be used. The PC 486 is then to be replaced by a Pentium running RTLinux. The decision to use RTLinux was based mainly on the cost issue; doing so will save around 20 KDM in VxWorks licenses. Tools for RTLinux have been purchased from Zentropix. Included in the package are the RT extensions to Linux and the kernel debugger.

The system looks after eight distinct devices which require service at up to 100 Hz and real-time response down to a few microseconds. The 100 Hz case refers to the control loops running in the servo control boards which control the length of the six legs. There are no serious doubts that RTLinux on a Pentium PC is capable of handling this part of the application; the main problem is that it will probably be necessary to develop Linux and RTLinux drivers for the servo boards in-house. Drivers for other boards (AD/DA) have already been developed in-house, using the Linux Device Driver Development Kit from FH-Berlin, so there is confidence that a similar job can be done in the case of the servo boards, once the complete hardware documentation has been obtained.

The Bochum decision to use RTLinux was based on the following:

- Source code is available and free.
- The development operating system and the tools are free.
- RTLinux runs on cheap hardware (an old PC486 would do).
- In the future, more students/engineers will be trained RTLinux. Universities will use RTLinux for teaching purposes because it is freely available and runs on cheap hardware.
- Most of the hard real tasks are only a small portion of a complete control system. With RT-Linux it is more difficult to develop the real time part but for the non real time part you get all developments and debugging tools for free.
- On a Pentium dual processor machine you could run on one processor the real time part and on the other normal Linux tasks.
- Support can be arranged through companies like Zentropix and FSMlabs, and there is free support available through the RTLinux mailing list (currently there are about 20-30 emails going in every day).
- At the moment it is not clear if the real time capabilities will be implemented within the standard Linux kernel. However, with multimedia applications (video, audio) there is a need to have some kind of real time tasking mechanism in the kernel, and so it seems likely that RTLinux will sooner or later become the norm.
- To cooperate within the ALMA project it will be difficult for small institutes to invest into a minimum VxWorks system which would cost around 30 KDM for the Hardware and 20 KDM for a development license.

3.3 Zentropix RTLinux latency tests

Plans to test the interrupt latency and reliability of RTLinux under different load conditions, proposed by Patrick Wallace (RAL), were shelved when it was found that extensive tests along exactly these lines had been published by Zentropix. The tests are reported in http://www.zentropix.com/support/document/testdata.html.

The tests were run under both standard Linux (v2.0.29) and RTLinux (NM Tech), on a 120 MHz Pentium. The first test used the internal clock of the PC and the parallel port to generate a fixed period of time, in order to measure the variations that occur when Linux is performing synchronous tasks. The second test measured the time taken to respond to an external interrupt, in order to characterize performance Linux when handling asynchronous tasks. Both tests were performed under different kinds of load for the system, ranging from no load to conditions of heavy disc, network and CPU activity in various combinations.

Under standard Linux, the time interval (20 ms) was generated with a standard deviation of about 1.5 μs when the machine was otherwise idle. This grew to some tens of μs when the machine was loaded, reaching a maximum of just over 0.5 ms for the full-load case. Under RTLinux there was a dramatic improvement: for no load,

the standard deviation was 0.9 μs, while the worst case load increased this to only 2.5 μs. The worst single measurement was an error of 19 μs, which happened when there was heavy network activity.

The interrupt response test under standard Linux produced performance that was strongly dependent on load, which is to be expected. With no load, the mean delay was about 22 μs, with variations between 20.8 μs and 48.7 μs. Heavy loads had only a small impact on the minimum delay, which could be as small as 29 μs even under full load. However, the maximum delay increased to many ms. As for the time interval test, RTLinux produced a dramatic improvement. The no-load latency varied between 2.10 μs and 15.50 μs, giving a mean of 2.36 μs. Under the worst-case load, the latency varied between 2.50 μs and 25.00 μs, averaging 4.38 μs.

Details of precisely how these tests were performed are available on the Zentropix website.

4 Conclusions

There is no definitive criterion that will decide now whether VxWorks or real time Linux (or something else) is right for ALMA. Clearly, both can do the job, and there is no evidence that either has a significant performance edge. The arguments are mainly to do with whether the current interest in RTLinux will wane and the supply of drivers, development tools and support dry up as a consequence, and whether RTLinux will stabilize or remain a moving target. Perhaps the best we can do for the present is to record the views of some individual members of the European Software Advisory Committee, and leave it up to the reader to form his/her own opinions:

Alain Perrigouard: My experience suggests that there is a strong case for using Linux on ALMA for some real time applications but that the arguments in favour of RTLinux are less secure at this time.

A proposal for ALMA to adopt RTLinux would have to identify support services comparable with those available commercially with VxWorks, even if these commercial services do not provide everything. Developers need to feel that there is some backup and expertise around. That means either a "hotline" contract with a company specializing in RTLinux, or in-house expertise equivalent to the ALMA software engineering group or the common software group and in a position to advise and assist developers. Such an in-house group would release only tested versions of the real time kernel.

People with VxWorks (or OS9 edited by Microware) experience feel safe with a maintenance and/or hotline contract, confident that any system software bug or misunderstanding will be solved by experts. With Microware, I came up against the limits of their competence and their will to pursue my specific troubles. For RTLinux, a real support capability should be identified (internal or commercial) and Internet

contact alone should not be the only help the developers get. If an expert group exists, it would form the interface to RTL in the USA or RTAI in Italy, and shield the ALMA developers from contact at this level.

Malcolm Stewart: The separate paper *Choice of RTOS for ALMA – Another View* presents a pro-VxWorks case. Its conclusions are as follows:

"My overall view is that VxWorks meets the requirements, is well proven and is likely to be better supported over the lifetime of the ALMA Project. RTLinux is cheaper in terms of capital cost, though this is not a significant factor. However it is still an immature product and would impose unjustified risk on the ALMA Project, if its adoption did not include additional internal support effort. If and when RTLinux matures, it may become a better alternative to VxWorks, but the same could be said of other products such as LabView/NT.

There is no reason not to use Linux, both for high level software and device control where there are no demanding realtime requirements, though the availability of drivers could be an issue. Currently VxWorks is a safer choice that RTLinux, but RTLinux could be used safely if additional effort is allocated for systems support and writing drivers."

Patrick Wallace: I think RTLinux is a much better bet in the long term. Phase 1 in general and the prototype antenna evaluation in particular would have been an excellent opportunity to try out RTLinux, and it is a pity that the decision to use VxWorks had to be made before a proper debate could get underway. (Use of VxWorks is stipulated in later revisions of the Antenna Monitor and Control Interface (ICD 9), and VxWorks licenses have already been purchased for the antenna evaluation.)

5 Acknowledgements

This report makes extensive use of material provided by Philip Daly (NOAO), Brian Glendenning (NRAO), Alain Perrigouard (IRAM) and Roland Lemke (Bochum). An assortment of Web pages were plagiarized as well.

Appendix: RTLinux on the Web

General	http://www.realtimelinux.org
RTL	http://www.rtlinux.org
	http://www.fsmlabs.com
	http://www.synergymicro.com
RTAI	http://www.aero.polimi.it/projects/rtai
Zentropix	http://www.zentropix.com
Comedi	http://stm.lbl.gov/comedi