European Air Traffic Flow Management: Porting a Large Application to GNU/Linux

Gaetan Allaert
Thales IS sa/nv
Chaussée de la Hulpe, 177
B-1170 Watermael-Boitsfort
Brussels, Belgium
gaetan.allaert@thales-is.com

Dirk Craeynest
Aubay Belgium
Gatti de Gamondstraat, 145
B-1180 Ukkel
Brussels, Belgium
dirk.craeynest@aubay.be

Philippe Waroquiers
Eurocontrol
Rue de la Fusée, 96
B-1130 Haren
Brussels, Belgium
philippe.waroquiers@eurocontrol.int

ABSTRACT

Computer hardware evolves very quickly. To benefit from cheaper and more powerful systems, big applications have to be ported to new environments. The Ada language has been designed for portability, making such migrations easier. However, today's applications often complement their main implementation language by various extra technologies: shell scripts, direct usage of OS primitives, different programming languages to access some libraries e.g. for graphical user interfaces, etc. These technologies are not always standardized or as portable as Ada so it is important to have indications about the portability of the various languages, libraries and tools.

ETFMS, the Enhanced Tactical Flow Management System, is a large Eurocontrol/CFMU application written mainly in Ada. This paper reports the findings of an exploratory port of ETFMS from HP-UX PA-RISC to GNU/Linux Intel. The performance figures obtained on both platforms are compared and several conclusions about portability are given.

Categories and Subject Descriptors

D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement—*Portability*; D.3.0 [**Programming Languages**]: General—*Standards*

General Terms

Standardization, Languages, Performance, Economics

Keywords

Ada, C, C++, Korn shell, COTS, GNAT, POSIX, GNU, Linux, HP-UX, Intel 80x86, HP-PA RISC, Air Traffic Management, Eurocontrol, CFMU, ETFMS, Portability, Performance

SIGAda'03, December 7–11, 2003, San Diego, California, USA.

1. INTRODUCTION

Eurocontrol, the European Organization for the Safety of Air Navigation [1], was tasked in the late 1980's by its control body - the ministers of transport of its member states - to establish a Central Flow Management Unit. The CFMU is responsible for the following activities:

- Flight Plan Processing: receiving flight plans filed by aircraft operators, validating and if necessary correcting those flight plans - automatically or manually - then redistributing the flight plans to the aircraft operators and the overflown airspace control centers.
- Air Traffic Flow Management: when the planned traffic load exceeds the capacity of Air Traffic Control (ATC), the CFMU is responsible for balancing the number of flights with the available ATC capacity, the objective being the optimum use of European airspace and the prevention of air traffic congestion.

To support these activities, the CFMU has developed IFPS (Integrated initial Flight Plan processing System) and ETFMS (Enhanced Tactical Flow Management System) [2]. Both were developed with common tools and techniques: HP-UX PA-RISC servers and workstations, Ada [14], Motif, Oracle, ... A large proportion of the code is common between the two systems.

In response to rising Air Traffic load and new requirements these two systems are continuously being improved. They have been described in more detail in [15, 16, 5, 3].

1.1 ETFMS architecture

ETFMS is a distributed multi-process application. The human-machine interface (HMI) runs on workstations and communicates using middleware (UNAS) with multiple processes running on a server. The hardware is PA-RISC workstations and multi-processor servers, all running HP-UX 11i. The server side is multi-process for performance and reliability reasons (critical and non-critical activities are taken in charge by different processes).

Ada is widely used [6], among others in space [4, 9], aeronautics [7, 11, 1, 12, 13] and air traffic management applications [10]. ETFMS is written mainly in Ada with some Korn

[1]For more information about Eurocontrol and CFMU, see http://www.eurocontrol.int.
[2]ETFMS is the successor of the TACT tactical system.

shell and a few low-level parts in C. The ETFMS code has already been migrated from Ada 83 to Ada 95 [16]. Table 1 shows the number of source lines (including comments and blank lines).

Language	Lines	Percentage
Ada	$1180K$	91%
C	$45K$	3%
Korn shell	$78K$	6%
Total	$1303K$	100%

Table 1: Language proportions

The software is decomposed into subsystems varying between 10K and 200K lines. The high level subsystems are providing the end-user functionality and are implemented using lower level subsystems. An extensive suite of automatic tests are used to validate the software after each build. Some tests are checking a specific subsystem while others are full system tests.

1.2 Goal of the exploratory port

In the last years, Intel based PCs have evolved very quickly, providing now powerful systems with an outstanding performance price ratio. GNU/Linux [17] is an attractive operating system: it is open source, allows a choice of supplier, etc. All this provides an opportunity to consider replacing the ETFMS servers and/or workstations by cheaper and/or more performant systems.

To evaluate the possibility and interest of switching to a new platform, an exploratory port of ETFMS has been launched. By investing a limited effort of maximum 10 person weeks, the objective was to obtain indications about the difficulty of porting and running parts or all of ETFMS on a GNU/Linux Intel based system. Although the initial goal was only to produce a list of porting issues, due to the high portability of the ETFMS Ada code, the end result was a tested and running system.

Some aspects were excluded from the scope of the exploratory port e.g. hardware redundancy support, supervision, development environment, ...

1.3 Overview of the porting activities

Porting an application implies more than just compiling the code. A lot of various activities have been done:

- Install the OS (GNU/Linux).
- Install the various Commercial Of The Shelf (COTS) software packages and tools needed by the application (Oracle, Motif, GNAT, ...).
- A big application (millions of lines) requires structured and controlled build and test procedures. These tools and procedures give a specific development environment that has to be ported as well.
- Port each subsystem one at a time, solving the non-portable aspects and verifying the subsystem with its automatic tests. As the low level subsystems are ported and tested first, problems are detected in a simpler context than if they would be revealed in the complete application.
- Compare the performance important for development (compile time) and operations (application execution time).

The above activities are discussed in the next sections, mainly focusing on the difficulties encountered.

2. GNU/LINUX CONFIGURATION

The platform used for the port is an HP Compaq PC with a clock speed of 2 GHz and 1 GB of memory.

A standard RedHat 8.0 configuration was initially installed. The configuration was adapted afterwards to solve various problems:

- The original kernel provided with the RedHat 8.0 distribution made Oracle loop. The installation of a newer kernel version (2.4.19) solved this problem.
- The default setup of the X server, the GNOME environment and the NVIDIA Quadro graphic card entered into conflict with the usage of colors by the HMI. The ETFMS HMI was developed using pseudo-color (i.e. an indirect color map indexed using 8 bits). The HMI consumes a high number of the color map entries (either to draw the needed colors or indirectly by using bit planes to implement fast highlighting). On HP, the graphical card can display at the same time one pseudo-color application (the HMI) and other applications using 16 bit colors (e.g. the HP CDE Common Desktop Environment). On the PC, this was not possible. It would have been possible to start the HMI using a private color map but this made the screen flash when the mouse moved over or off the HMI. To solve the problem, the PC was configured to start two X servers, simulating two different hardware screens, switchable via keyboard. One X server is running GNOME (used for development), the other is running the Motif Window Manager which is less color hungry. In addition to that, it was needed to install a specific version of the NVIDIA driver in order to disable the X server render extension as this also uses a lot of colors.

  Note that this triggered a memory leak in the GNOME terminal, preventing its use on the second X server. The leak was so large that various processes were killed by the kernel, as explained in [8].
- By default, the RedHat distribution sets `LANG=en_US.-UTF-8`. Amongst other effects this modifies behavior of the sort command compared to HP where `LANG=C`. So `LANG=C` was also set on Linux.

3. COTS

ETFMS uses COTS software that needs to be installed on Linux.

- The Ada compiler. The compiler used on Linux is the same as on HP-UX: GNAT 5.00a with the POSIX pthreads based runtime. The GNAT compiler was installed in a different directory than the standard RedHat gcc. If needed, this makes it possible to use different gcc versions for Ada and for other languages (e.g. C++) using the PATH environment variable.

- The Oracle database. ETFMS on HP is currently based on Oracle version 8, which is no longer available on Linux. Hence, the Oracle 9i client and server were installed on the PC. The client can be used to either access a local PC Oracle 9 database or an HP Oracle 8 database. Note that the performance measures on Linux (see section 6.3) were done by accessing the Oracle 8 database on HP. ETFMS accesses Oracle through a specific Ada binding, providing a higher level interface than the standard Oracle C based access. Switching between Oracle 8 and 9 required no change in this binding.

- The GTK graphical toolkit, the GtkAda binding, the BOOCH components and the XML/Ada library. All compiled without problems on Linux.

- On HP, we use several GNU tools (e.g. awk, sed, grep, ...) as they have more features than the corresponding HP tools. These tools were grouped in a special subsystem and built using similar procedures as the application subsystems. As most of these tools are standard on RedHat, it was not needed to build them on Linux.

- The HMI is based on GTK and Motif. On HP, we use the provided Motif 2.1 library. On Linux, we use OpenMotif, the open source version of the Motif toolkit.

- The UNAS middleware implements a communication layer used by ETFMS to connect all the processes together. The UNAS Ada code compiled in a straightforward way but the C code had to be adapted.

4. DEVELOPMENT ENVIRONMENT

4.1 Compiler configuration

The ETFMS development environment supports the use of multiple compilers or configurations of the same compiler. These different configurations are used for development builds (non-optimized code, many optional checks enabled) and for operational builds (optimized code, only the standard Ada checks enabled). This flexibility is also very useful for evaluation of new compiler releases, alternatives switch settings, ...

Three sets of compiler switches were used on the Linux-PC:

IP **Initialize Scalars**. This is the default CFMU setup for development [5]. It ensures fast compilation and detects as many problems as possible by enabling a wide range of runtime checks. Among others, each scalar variable is initialized, if possible to an invalid value. In the latter case, this will raise a Constraint_Error if the variable is used before it is assigned.

O2P **Optimized**. The `-O2` option is used to generate more efficient code. Only the standard Ada checks are kept.

OP **Operational**. This further optimizes the code by using the backend inlining option `-gnatn`.

4.2 Build tools

To support the development process, a set of Korn shell scripts were developed inside CFMU. These scripts enforce a standard way to develop. They also isolate the developers from the complexity of some tools such as ClearCase. These scripts support:

- automatic build and non-regression testing of a full system: either for a private developer build or an integration build; these can be done using the various compiler configurations (e.g. IP, OP, ...);

- modification of the source code: this allows a modification to be developed and tested without affecting other developers;

- packaging and deployment of the operational ETFMS;

- source code control: parallel development, maintenance branches, ...

To be able to compile, run and test ETFMS on Linux, two approaches were envisaged:

- develop a new minimal set of `bash` (the default Linux shell) scripts,

- or re-use the set of existing `ksh` scripts.

It was decided to re-use and adapt the needed `ksh` scripts on Linux. To limit the work, only the scripts needed for automatic build and test were adapted. This was sufficient to reach the objective of the exploratory port.

5. PORTING EXPERIENCES

This section summarizes the encountered portability problems by domain. These problems were preferably solved by avoiding the non-portable construction. Only when this was not possible, both HP and Linux specific code fragments had to be kept. This way of working also increases the portability of the code to other operating systems.

5.1 Korn shell scripts

The encountered shell script problems fall into two categories. The first relates to the Korn shell itself: language syntax, statement execution, use of system calls, ... The second relates to the commands started by the shell: some are only available on one platform, some have different names and some have different options or behavior.

- The language syntax is handled differently by the HP-UX and Linux `ksh`. For example the command substitution without a trailing backquote is accepted on HP but not on Linux: `if [ "‘basename $0" != "Ccontext" ]`. Another example is the use of the `continue` instruction outside a loop.

- The pipe `|` is executed differently on HP-UX and Linux. The Korn shell on Linux forks a new Korn Shell for the command on the right side of the pipe (POSIX shell compliant). On HP-UX, the new Korn Shell is forked for the command on the left side of the pipe. This can lead to different results during the execution. For example, the next command is interpreted differently on HP and Linux: `export A="Empty"; echo`

`"Full" | read A; echo $A`. The output is `Empty` on Linux but `Full` on HP because, on Linux, `A` is modified in a sub-shell and then the modification becomes invisible in the current shell.

- For security reasons, Linux accepts a set-user-id bit only for compiled programs, not for scripts. Hence, a script that needs to run with a specific user id cannot be called directly. Instead, we used a small compiled wrapper program having the set-user-id bit set to launch the script.

- The typeset -L builtin command is not implemented on Linux.

- The behavior of the builtin command `echo` is not the same on Linux and HP-UX when the string contains `\n`.

- The `mktemp` command has different arguments. A trivial `cfmu_mktemp` script encapsulates `mktemp` to isolate our scripts from these differences.

- The `test` command on Linux only accepts one file argument. The command `test -f *.adb` fails on Linux when *.adb expands to more than one file. HP silently ignores extra file arguments.

- On Linux, the `rm symlink` command will prompt for confirmation if the symbolic link references a read-only file; on HP-UX it does not. Hence on Linux the command needs to be replaced by `rm -f symlink`.

- Some missing or non equivalent commands, different names, arguments or output: `chatr` is replaced by `ldd`, `model` is replaced by `cat /proc/cpuinfo`, `ar` and `as` use different options, `cc` is replaced by `gcc` and the compiler options need to be adapted, Linux outputs an extra leading blank before the value returned by the command `wc -l`, differences in the output format of `ipcs`, ...

- The `ps` command arguments and output varies widely between operating systems. This was already noticed between between two HP-UX releases. For this reason, the use of the `ps` command was already isolated in a few higher level commands. These were easily adapted for Linux.

5.2 C++ code

ETFMS contains a limited amount of C++ code to interface to a C++ COTS library used to generate static geometrical data in binary format. This data is also represented in Ada types that support both ASCII and binary I/O. Linux binary files were generated by converting these files to ASCII on HP and converting back from ASCII to binary on Intel. It was thus not necessary to acquire the C++ library for Linux and port our C++ code.

5.3 C code

Many bugs in C code are not detected at compile time or cause an implementation dependent behavior at run-time. Hence, such bugs can easily stay hidden until something in the environment changes e.g. compiler, operating system, ...

Some aspects of the C language are not rigorously defined or easily lead to non-portable code. In particular, the C environment is a jungle of standards as illustrated by the "obeyed" standards for `strlen`. On Linux: SVID 3, POSIX, BSD 4.3, ISO 9899. On HP: AES, SVID2, SVID3, XPG2, XPG3, XPG4, FIPS 151-2, POSIX.1, ANSI C.

- On HP, calling `strlen` with `NULL char*` returns 0. Similarly, HP `strcpy` interprets `NULL` as a valid pointer to a zero byte. On Linux, both result in a memory fault.

- Some code was directly accessing `argv` without checking the number of arguments with `argc`. When going out of range, this gave a NULL pointer on HP and a core dump on Linux.

- A `\0` must explicitly be added at the end of the result of the `readlink` system call. This was working by chance on HP.

- The way the header files provided by the operating system are included may vary. Some `#include` need to be added, others need to be removed. In the worst cases, the includes need to be ordered differently to prevent conflicts (for example between `varargs` and `stdarg`).

- `abort ()` is a more portable way to generate a core dump than `raise (_SIGSYS)`.

- `stderr` is defined under Linux as a macro; it is a call to a C function that returns the address of the standard error FILE. Under HP-UX, `stderr` is a static address. In C, a function cannot be called to initialize a global variable, preventing our C code from compiling on Linux. Initializing the global variable is now done dynamically.

5.4 Ada interfacing with C

A large number of problems were triggered by some C code directly called from Ada (e.g. when the Ada code makes use of services provided by the operating system libraries).

- Differences between the C system constants under HP-UX and Linux. A C program was written to generate an Ada spec (`os_constants.ads`) containing system constants. This program only has to treat constants that are not already present in the POSIX Ada binding [2].

- In an Ada 83 package, not yet converted to Ada 95, replace the name of the linker symbols that count and contain the arguments given to a program (`__argc_value` and `__argv_value`) with the less platform specific names provided by GNAT (`gnat_argc` and `gnat_argv`).

- The name of the linker symbol (used for profiling) for the beginning of the text segment of an executable is not the same on HP-UX (`_text_start`) and Linux (`_start`). The `_start` symbol can be used under Linux because the text segment is in fact the first segment of the executable. This is not the case with HP.

- To isolate the Ada code from non standardized C structures, whenever possible, the Ada interface code was defined using opaque types, e.g. we represent the C type `regex_t` defined in `regex.h` by an array of bytes.
- Some regular expressions have a different meaning on HP and Linux. For example, the regular expression `a|` is refused on HP but matches any string on Linux as the empty expression after `|` matches everything.
- The `tm` structure in `times.h` contains more fields on Linux than on HP-UX. They have been added to the Ada interface record type. Those unused fields are not set when the code is running on HP-UX.
- To test if a socket is connected, a read of zero bytes was performed. On HP-UX, this returns an error if the socket is not connected. On Linux, it is a no-op. The Ada code has been made more portable using the `getsockopt` function after a `select` on the socket.
- Explicitly add `ASCII.Nul` at the end of strings to get a valid C string. Although some were missing, this was working by luck on HP.

5.5 Ada code

The only problems encountered in the Ada code were due to differences in the processor architecture - HP-PA RISC vs Intel.

- A type T needed to have a specific layout. This was implemented using a representation clause. By default the bit order is the same as the byte order, hence this order is different on HP-PA RISC (big endian) and Intel (little endian). So, to be portable, it was needed to add `for T'Bit_Order use High_Order_First;`.
- Little/big endian problems when mapping a structure on an 4 bytes integer with a representation clause. Some code interfacing to UNAS depends on the fact that specific fields are located in the least significant bits of the corresponding integer. The representation clause was parameterized with the endianess to ensure that each structure value was interpreted as the same integer value on both architectures.
- Generating a bit mask from an integer value must take into account the difference in the representation of an integer between Intel and HP-PA RISC; also a little/big endian problem.

Once the above problems were corrected, some tests were still failing on Linux. The reason was related to the difference in arithmetic accuracy between the HP-PA RISC and the Intel architectures:

- On Intel processors the intermediate results are computed with 80 bits accuracy and the final result is stored with 64 bits accuracy. On HP-PA RISC architecture, the intermediate results and the final result are computed with 64 bits accuracy.

  With A, B, C, D, E, F floats,

$$\begin{aligned} A &:= B/C \times D \\ E &:= B/C \\ F &:= E \times D \end{aligned}$$

  On HP-PA RISC, A and F will be always the same. On Intel, A and F will be the same if and only if E is the exact value of B/C.

- The above has an effect on the comparison of floating point numbers.

  Given variables A, B, C, and $A := B/C$, the following statement is non-deterministic: `if A < B / C then ...else ...end if;`. The result depends on the compiler and the compilation optimization level. Hence, for portability, such code must be rewritten.

Some regression tests were impacted by this difference in arithmetic accuracy. For each flight plan, ETFMS calculates a profile providing the time at each en-route position (latitude, longitude and altitude) rounded to 5 seconds. To validate the result, a textual print-out of the profile is compared using a "simple" `diff` to a reference file containing the expected result. When this test was run on Linux, a difference of 5 seconds was detected for one flight plan. This was initially considered a failure until detailed investigation showed it was caused by differences in arithmetic accuracy. Hence, for this test, the workaround could be to provide an HP and a Linux reference file.

When testing the optimized builds on Linux, similar differences were noticed: in the optimized code, the compiler directly uses the intermediate results without copying it from the 80 bits register to the stack using 64 bits. This again leads to differences in profile calculation results. A possible workaround could be to have reference files that depend on the configuration (Linux/HP, IP/O2P/OP), but this quickly becomes unmaintainable: each time the optimization algorithms of the compiler are improved, the reference files might have to be adapted. We have seen such differences between GNAT 3.15a, GNAT 5.00a and GNAT 5.01w. A better solution would be to implement a "clever" `diff` that takes time tolerances into account.

Note that this float accuracy problem on Intel has an impact on all languages that are compiled so that the resulting object code directly uses the Intel CPU floating point unit. Typically, similar problems are encountered for C, C++, ... Such direct usage of the floating point unit is required in order to obtain efficient numerical computations.

6. RESOURCES AND PERFORMANCE

The characteristics of the machines used are given in Table 2. Performance measurements are given for the three different compiler configurations described in section 4.1: Initialize Scalars (IP), Optimized (O2P) and Operational (OP). The compiler used was GNAT 5.00a. However, a bug in that release slowed down the compilations quite a lot, so we re-measured the compilation times with the just released GNAT 5.01a.

Machine type	Operating system	Processor(s)
HP Workstation	HP-UX 11i	$400MHz$
HP Server	HP-UX 11i	$875MHz$
Linux-PC	Linux RedHat 8.0	$2000MHz$

Table 2: Machine types

6.1 Size of the executables

ETFMS can run in two different modes: as various OS processes that communicate together (called an ETFMS network) or all the code in one process (called the `mono_process`). Both modes have been compiled and run on Linux.

Here, as an example, we compare the size of the `mono_process` using the Unix `size` command (text + data + bss).

- Comparison of the size of the executables on HP-UX using the IP configuration as reference (Table 3).

Build	Size	Relative
IP	89 MB	100%
O2P	56 MB	63%
OP	60 MB	67%

Table 3: Size on HP-UX

- Comparison of the size of the executables on Linux using the IP build as reference (Table 4).

Build	Size	Relative
IP	65 MB	100%
O2P	50 MB	77%
OP	54 MB	83%

Table 4: Size on Linux

- Comparison of the size of the executables on HP-UX and Linux using the IP build on HP-UX as reference (Table 5).

Build	HP-UX	Linux
IP	100%	73%
O2P	63%	56%
OP	67%	61%

Table 5: Size comparison

The Linux executables are smaller than the HP-UX executables. One factor explaining this difference is the zero-cost exception mechanism available on HP: executables contain pre-computed tables to improve the run-time performance when few exceptions are raised. The sizes should be remeasured with GNAT 5.01a which now supports zero-cost exception on Linux as well.

6.2 Compilation performance

To get an idea of the relative compilation speed, a number of significant files were compiled for different builds (IP, O2P, OP) on different platforms (HP-UX, Linux). Note that GNAT 5.01a was used for this section.

Table 6 gives the compilation times (user + system cpu) for different builds.

In Table 7, each IP build is used as reference to give relative times for each machine.

On both HP and Linux, the compilation time for an O2P build is roughly 50% more than for the IP build. Due to the back-end inlining, the OP build requires significantly more time: about 3 times as much as the IP build and 2 times as much as the O2P build.

Build	HP Workstation	HP Server	Linux-PC
IP	3m50s	2m16s	0m46s
O2P	5m53s	3m01s	1m18s
OP	12m47s	6m26s	2m48s

Table 6: Compilation times

Build	HP Workstation	HP Server	Linux-PC
IP	100%	100%	100%
O2P	153%	133%	170%
OP	333%	284%	365%

Table 7: Relative compilation time per machine

The IP compilation on HP Workstation is used as reference in Table 8.

Build	HP Workstation	HP Server	Linux-PC
IP	100%	59%	20%
O2P	153%	79%	34%
OP	333%	168%	73%

Table 8: Relative compilation time

The tables 6 and 8 show that compiling on a regular PC is significantly faster than on HP workstations and even on HP servers.

The last table (Table 9) is adapted to take into account the machine's CPU speed, i.e. the figures have been adjusted by CPU clock frequency to indicate what might be expected at equal processor MHz.

Build	HP Workstation	HP Server	Linux-PC
IP	100%	129%	100%
O2P	153%	173%	170%
OP	333%	367%	365%

Table 9: Relative compilation time per MHz

For compilations, we see that 1 Intel MHz on a Linux PC is roughly equivalent to 1 PA RISC MHz on an HP-UX machine. In other words, the difference in the instruction set and operating system has no significant impact on the compilation speed. The main factor influencing the compilation speed is the raw processor speed (of course if the machines have enough memory to avoid paging).

As GNAT supports parallel compilations, it is thus interesting to see if a multi-CPU server can be used to compile a big application faster. Experiments done on a 4 CPU HP server have shown that parallel compilation scales very well. From our experience, a good guideline is to launch N+1 parallel compilations on a N CPU system; this is valid even for a single CPU system.

6.3 Execution performance

A heavy test reprocessing a significant set of ETFMS input flight and radar data was used to get an idea of the relative execution speed. This was done with the `mono_process` and hence does not depend on the number of processors

on the machine. The execution time for different kinds of build (IP, O2P, OP) on different platforms (HP-UX, Linux) is given in Table 10.

Build	HP Workstation	HP Server	Linux-PC
IP	29m18s	14m26s	6m03s
O2P	12m03s	6m07s	3m07s
OP	11m09s	6m00s	3m00s

Table 10: Execution times

In Table 11, each IP build is used as reference to give relative times for each machine.

Build	HP Workstation	HP Server	Linux-PC
IP	100%	100%	100%
O2P	41%	42%	52%
OP	38%	42%	50%

Table 11: Relative execution time per machine

On both HP and Linux, an optimized executable runs roughly twice as fast as the IP version. This is explained partially by the overhead for additional checks in the IP configuration.

The back-end inlining appears to only improve performance marginally. Similar comparisons done a few years ago with an earlier GNAT version were giving a difference of around 15% between O2P and OP. Based on that, we then selected OP builds for operational use. However, as seen in the previous section, OP takes significantly longer to compile than O2P. It is also more difficult to debug. The above observation may lead us to revisit this decision, though more study is required: we need to check more closely how effective back-end inlining is and whether front-end inlining might give better results.

Table 12 uses the IP build on HP Workstation as reference.

Build	HP Workstation	HP Server	Linux-PC
IP	100%	49%	21%
O2P	41%	21%	11%
OP	38%	20%	10%

Table 12: Relative execution time

The final table (Table 13) takes into account the CPU clock speed.

Build	HP Workstation	HP Server	Linux-PC
IP	100%	107%	105%
O2P	41%	46%	55%
OP	38%	44%	50%

Table 13: Relative execution time per MHz

Here again, we can conclude that the execution time is proportional to the CPU speed: 1 Intel MHz on a Linux PC is roughly equivalent to 1 PA RISC MHz on an HP-UX machine.

7. FUTURE OPTIONS

The exploratory port has shown the feasibility and interest to have part or all of ETFMS running on a GNU/Linux Intel platform. This opens a range of possibilities. This section examines some options to consider and the remaining work for each of them.

7.1 Maintain code portability

To maintain code portability during the evolution of the ETFMS project, the following steps could be taken regularly:

1. Install new versions of the compiler on Linux to keep in sync with the GNAT version on HP.
2. Rebuild the complete ETFMS in IP, O2P and OP mode to check the code still compiles under Linux and no incompatibilities have been introduced during new development.
3. Regenerate the binary data to run ETFMS on Linux, when changes to that data structure have been made in the code.
4. Run all tests to validate ETFMS on Linux. If necessary, adapt the tests to Linux/Intel specific aspects.
5. Run a replay of a day's traffic in the current ATM environment [3]. It is required to validate ETFMS in operational-like conditions, e.g. with 25000 flights and realistic generated delays.

The effort needed to maintain code portability is estimated to be quite low (in the order of 1 person-day per month). It consists mainly of the administrative work needed for building and testing ETFMS on Linux (cleanup of old builds, check the results, ...).

7.2 Hardware replacement strategy

The exploratory port has shown that a Linux Intel PC might be a cost-effective alternative to the currently used platform. A wide range of strategies are possible regarding the evolution of the current ETFMS hardware and need further study:

- Keep HP servers, replace workstations by Linux PCs.
- Replace both servers and workstations by Linux PCs.
- Use Linux PCs for non-critical instances of ETFMS. Non-critical instances are full ETFMS systems that are used e.g. for off-line activities such as evaluation of new airspace structures, studies of alternative operational parameters, demonstration of ETFMS at international conferences and exhibitions, ... These non-critical instances have a lower availability requirement than the instance handling the on-line data.

7.3 Additional work

Depending on the hardware strategy chosen, some additional work is needed to ensure a smooth usage of the new platform. Fully transitioning ETFMS to a new environment implies more activities than just porting the application. A

[3]Environment: geographical information, route network, airspace sectorization and similar, all updated regularly.

lot of these activities are implied by the criticality of the ETFMS system that has a very high availability requirement. Some of the remaining work for this is:

- An operational center is using more than just the "business" application. A lot of other support tools are used e.g. for backup, capacity planning, ... These tools must be searched for, evaluated and installed on the new server platform.
- The critical instance of ETFMS is monitored 24h on 24 by a technical operator team. This team is using a supervision application implemented using HP OpenView. If the critical instance of ETFMS runs on GNU/Linux, then it could imply the need to port this supervision application on Linux (maybe using a product similar to OpenView). An alternate approach would be to ensure that the HP OpenView based supervision application can supervise ETFMS running on GNU/Linux. This supervision application is not mandatory for non-critical ETFMS instances, though.
- If we choose for a mixed environment (i.e. an HP server with GNU/Linux workstations), then we must ensure that all the binary messages exchanged are properly taking into account the conversion needed between big and little endian. Most of the messages exchanged are being encoded/decoded using a central "streaming" package. This means that the conversion to a standard integer format can be done at a central place. The messages that are not using this streaming package must be converted to use them.
- CFMU already has a deployment procedure to install ETFMS on an operational platform running HP-UX. The deployment tool needs to be ported to Linux to be able to install an ETFMS on an operational Linux platform. In case a mixed environment is chosen, this tool must support the simultaneous installation of 2 different kinds of build (HP and Linux builds).
- If the decision is taken to completely transition to GNU/Linux, then it means that the full set of development tools has to be ported. During the exploratory port, only the part that performs the build and test was ported. The parts to be ported and/or rewritten provide mainly the link with the source code control system.

8. CONCLUSIONS

8.1 Portability and languages

This experience confirms that the Ada language provides a very good basis to obtain portable code. Despite the usage of sometimes advanced language constructs (multi-tasking, extensive usage of generics, exceptions, complex data structures, ...), we encountered only a very limited number of portability problems in more than one million lines of Ada source code. This is even more impressive knowing that the system was developed exclusively on HP-UX and that no special guideline was imposed during development to make the code portable.

This shows that code written using a strongly *standardized* language like Ada, which includes a lot of checking during compilation and execution, leads not only to safer and more robust programs but also to highly portable code.

In contrast to the above, a lot more portability problems were encountered in the C code and especially in the Korn shell code. Due to the little number (or complete absence) of compile time and run-time checks, code written using such loosely defined languages contains more non portable constructs and hidden bugs. The code may be "OK" in a specific environment, but the hidden bugs are triggered when surrounding conditions are changed (e.g. a change in the version of the operating system, a change in the compiler version, a port to a new platform, or in the worst case just a difference between the test and the operational environment).

8.2 Portability and COTS

Using COTS software may speed up development as it brings already developed functionality to the application. On the other hand, it often enforces a specific way to develop, and implies additional costs (learning curve, administration work, license costs, bugs difficult to isolate or fix, ...). COTS also introduces additional dependencies, causing potential portability problems: the COTS is not necessarily available on a new target platform or may behave differently.

Therefore, before deciding to use COTS software, it is important to balance the potential benefits, costs and risks.

In our case, ETFMS uses few COTS products, all are available on Linux and gave few problems.

8.3 Portability and operating systems

Most of the problems encountered in the Ada code were due to the need to interface to some C libraries, e.g. providing operating system services.

The best way to further increase the portability of Ada code is to avoid interfacing C code directly. Such interfacing problems are minimized if various services (e.g. system calls, regular expressions, time management, sockets, ...) are made accessible through standardized high level bindings ("thick" bindings). As an example of portable operating system access, there was no need to adapt the Ada code written on top of the POSIX Ada binding.

Implementing portable multi-threaded applications is considered a difficult challenge due to the difference in the scheduling, priority handling, non-standardized definition of threads, interaction between threads and exceptions, ... Our experience shows that the isolation provided by the high level definition of Ada tasks and the seemless integration between tasks and the other language constructs facilitates the development of portable multi-task applications. We encountered no problem in the quite complex Ada tasking code part of ETFMS.

8.4 GNU/Linux

Due to their high quality, ETFMS was already using many GNU tools for development on HP-UX, the most noticeable one being the gcc GNAT compiler. The port to Linux has confirmed that GNU/Linux is a high quality environment, both for development and deployment of an operational application. An important advantage is the open nature of GNU/Linux which helps to solve possibly encountered problems. A wide range of help and information is available on the Internet.

Linux is running on cheap PC hardware, provided by a wide range of vendors. Similarly, the software services (e.g. support) can be bought from multiple suppliers. Such a wide availability decreases the risk and cost which may be implied by a too strong link between a critical system like ETFMS and its operational platform. CFMU will now further evaluate the possible quality and cost advantages provided by moving ETFMS partially or completely to a GNU/Linux environment.

9. ACKNOWLEDGEMENTS

Many thanks to Andrew Hately, currently at Eurocontrol, CEATS Research, Development and Simulation Centre, Ferihegy 1 'A' porta, 1185 Budapest, Hungary (andrew.hately@eurocontrol.int), for his detailed review of an earlier version of this paper.

10. REFERENCES

[1] The Boeing 777 flies on 99.9% Ada. *www.adaic.org/atwork/boeing.html - Ada Information Clearinghouse*.

[2] *IEEE Std 1003.5b-1996 Standard for Information Technology - POSIX Ada Language Interfaces - Part 1: Binding for System Applications Program Interface (API)*. The Institute of Electrical and Electronics Engineers, 1996.

[3] E. Briot, F. Gasperoni, R. Dewar, D. Craeynest, and P. Waroquiers. Exposing memory corruption and finding leaks: Advanced mechanisms in Ada. In *Proceedings of 8th International Conference on Reliable Software Technologies - Ada-Europe 2003, Toulouse, France, June 16-20, 2003, Jean-Pierre Rosen, Alfred Strohmeier (Eds.)*, volume 2655 of *Lecture Notes in Computer Science*, pages 129–141. Springer-Verlag, 2003.

[4] R. Dewar. Case study: Space station robot embeds Ada. *COTS Journal*, pages 87–91, March 2002.

[5] R. Dewar, O. Hainque, D. Craeynest, and P. Waroquiers. Exposing uninitialized variables: Strengthening and extending run-time checks in Ada. In *Proceedings of 7th International Conference on Reliable Software Technologies - Ada-Europe 2002, Vienna, Austria, June 17-21, 2002, Johan Blieberger, Alfred Strohmeier (Eds.)*, volume 2361 of *Lecture Notes in Computer Science*, pages 193–204. Springer-Verlag, 2002.

[6] M. B. Feldman. Who's using Ada? Real-world projects powered by the Ada programming language. *www.seas.gwu.edu/∼mfeldman/ada-project-summary.html, Washington, DC, USA, SIGAda Education Working Group*, August 2002.

[7] B. Frisberg. Ada in the JAS 39 Gripen flight control system. In *Proceedings of 3rd International Conference on Reliable Software Technologies - Ada-Europe 98, Uppsala, Sweden, June 08-12, 1998, Lars Asplund (Ed.)*, volume 1411 of *Lecture Notes in Computer Science*, pages 288–296. Springer-Verlag, 1998.

[8] M. Gorman. Understanding the Linux virtual memory manager. *www.csn.ul.ie/∼mel/projects/vm/*, 2003.

[9] N. Holsti and T. Långbacka. Impact of a restricted tasking profile: The case of the GOCE platform application software. In *Proceedings of 8th International Conference on Reliable Software Technologies - Ada-Europe 2003, Toulouse, France, June 16-20, 2003, Jean-Pierre Rosen, Alfred Strohmeier (Eds.)*, volume 2655 of *Lecture Notes in Computer Science*, pages 92–101. Springer-Verlag, 2003.

[10] J. Klein. Ada is alive and well in air traffic management. In *Proceedings of the 1998 Annual ACM SIGAda International Conference, Washington, DC, USA, November 8-12, 1998, Ed Seidewitz, William Thomas, Michael Feldman (Eds.)*. ACM Press, 1998.

[11] B. Lewis, S. Vestal, and D. McConnell. Modern avionics requirements for the distributed systems annex. In *Proceedings of 3rd International Conference on Reliable Software Technologies - Ada-Europe 98, Uppsala, Sweden, June 08-12, 1998, Lars Asplund (Ed.)*, volume 1411 of *Lecture Notes in Computer Science*, pages 201–212. Springer-Verlag, 1998.

[12] B. Pflug. Ada after 10 years of usage - Is there a commercial future? In *Proceedings of 5th International Conference on Reliable Software Technologies - Ada-Europe 2000, Potsdam, Germany, June 26-30, 2000, Hubert B. Keller, Erhard Plödereder (Eds.)*, volume 1845 of *Lecture Notes in Computer Science*, page 4. Springer-Verlag, 2000.

[13] V. Santhanam. The anatomy of an FAA-qualifiable Ada subset compiler. In *Proceedings of the 2002 Annual ACM SIGAda International Conference, Houston, Texas, USA, December 8-12, 2002, Salih Yurttas, John McCormick (Eds.)*, pages 40–43. ACM Press, 2002.

[14] S. T. Taft, R. A. Duff, R. L. Brukardt, and E. Plödereder. *Consolidated Ada Reference Manual. Language and Standard Libraries, International Standard ISO/IEC 8652:1995(E) with Technical Corrigendum 1*, volume 2219 of *Lecture Notes in Computer Science*. Springer-Verlag, 2001.

[15] P. Waroquiers. Ada tasking and dynamic memory: To use or not to use, that's the question! In *Proceedings of International Conference on Reliable Software Technologies - Ada-Europe 1996, Montreux, Switzerland, June 10-14, 1996, Alfred Strohmeier (Ed.)*, volume 1088 of *Lecture Notes in Computer Science*, pages 460–470. Springer-Verlag, 1996.

[16] P. Waroquiers, S. Van Vlierberghe, D. Craeynest, A. Hately, and E. Duvinage. Migrating large applications from Ada83 to Ada95. In *Proceedings of 6th International Conference on Reliable Software Technologies - Ada-Europe 2001, Leuven, Belgium, May 14-18, 2001, Dirk Craeynest, Alfred Strohmeier (Eds.)*, volume 2043 of *Lecture Notes in Computer Science*, pages 380–391. Springer-Verlag, 2001.

[17] M. Welsh and L. Kauffman. *Running Linux*. O'Reilly, 1995.