The SPIF project

Bertrand Dupouy, Olivier Hainque, Laurent Pautet, Samuel Tardieu

Ecole Nationale Supérieure des Télécommunications
Département Informatique - CNRS URA 820,
46 rue Barrault, 75634 Paris Cedex 13, France.

Abstract. This paper introduces the current developments of the SPIF (Système de Prototypage à Implantation rapide et Faible coût) project. The goal of SPIF is to provide a low cost environment for quick prototyping of embedded distributed real-time applications. The hardware platform is built with reusable, standard off-the-shelf components. SPIF is the name of the testbed itself, a mobile autonomous robot controlled by an embedded real-time system, SPIF-OS. Ada95 is supported as an high level efficient tool to engineer real-time embedded software. Ada95 extends the original model by providing distribution capabilities.
Both hardware and software have been developed at the ENST Computer Science Department.

1 Introduction

1.1 Overview

Our concern is to address the problem of prototyping real-time embedded applications for computer science students as well as for engineers.

- In real-time applications the correctness of the results depends not only on the logical correctness of their computation but also on the time at which they are produced. In real-time embedded systems these timing constraints are set by the sensors and actuators which allow the application to communicate with the external world. If data inputs and outputs are not processed within the expected deadlines the applications will not be able to meet their objectives. It is worth stressing that embedded applications are not only software but comprehensive systems, tightly joining hardware and software. The hardware devices involved are of a great variety: miscellaneous sensors and actuators, engines, etc.
- The hardware board sets additional constraints: the available memory has generally a small size and there is no disk. The whole hardware must be compact, reliable and its power consumption must be low.
- The embedded software must guarantee that tasks activated by sensors or tasks triggering actuators will meet their timing requirements. In distributed embedded systems, the system must also address the problem of communicating with remote tasks while still offering timeliness guarantees.

- Embedded systems are developed in a specific way: the system and the high level applications are engineered on a host computer using a cross development suite and then downloaded to the target hardware platform. Simulation on the host computer checks the correctness of the overall design and the general behavior of the application. But the only way to prove the timeliness of external events handling and to assess the reliability of the system under unexpected situations is to download it on a hardware test platform.
- For embedded applications, prototyping is a mandatory stage after simulation because development and debugging are then handled under significant conditions: the system has to prove its correctness with regards to external stimuli, it does not get its input from simulated data. The developers can check and validate the behavior of both hardware and software under real-time critical situations.

1.2 Objectives

In fact, crafting an embedded system is a global approach wherein software and hardware are jointly designed under time, cost, physical performances and reliability constraints.

In many cases, embedded real-time applications are developed using low level languages and very difficult to maintain or to reuse on a different target. The aim of SPIF is to offer students and developers means to elaborate and test reusable hardware and software embedded components. The design of a hardware platform and its embedded software will be achieved by putting together basic hardware and software bricks. This allows a modular and scalable approach which fulfills the objectives of low cost and quick development.

To meet these objectives, we decided to use Ada95 as a development tool and to port an Ada95 run-time to the top of SPIF-OS.

Our prototyping architecture is layered as follows:

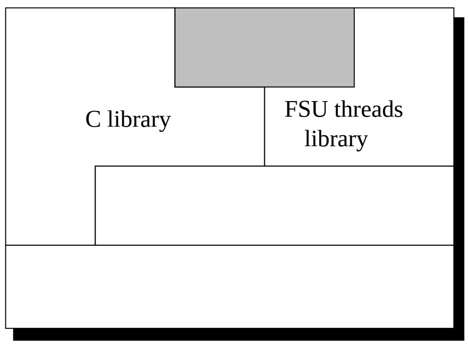
- the lower level: the hardware platform, a scalable mobile robot driven by a Motorola MC68302 processor,
- the low level software layer: the kernel, SPIF-OS, a real-time operating system,
- the "portability" layer: a C library and a POSIX [5] threads library. We chose the Florida State University one [4],
- the high level layer: an Ada95 run-time, the GNAT run-time.

This modular architecture makes it easy to change one of the layers: for instance we ported RTEMS [10] on top of the SPIF hardware.

The three upper layers make up the software platform. Figure number 1 shows the global architecture of SPIF.

1.3 Structure of the paper

Section 2 describes the first layer of the SPIF architecture: the hardware platform. This platform is a mobile robot. The modularity and scalability of the



(a) SPIF layers

Figure not available

(b) SPIF robot

Fig. 1. The SPIF architecture

design gave the ability to build several versions of robots fitted with different sensors and actuators. In section 3 we present the second layer: the real-time kernel. The current one is SPIF-OS, developed at ENST. SPIF-OS implements real-time scheduling algorithms and synchronization policies in a simple and efficient way. The section 4 is a brief overview of the third layer which includes the POSIX thread library and the development tools: the GNU suite. In section 5 we explain why we chose to port a GNAT run-time library on top of the SPIF architecture. This section points up how Ada95 extended the original model, especially by providing distribution tools. Section 6 deals with the current teaching applications and works in progress: the use of GLADE and the porting of RTEMS on top of the SPIF hardware. Finally, section 7 concludes and presents the ongoing works.

2 The hardware platform

2.1 Design principles

The SPIF hardware platform is an autonomous mobile robot. All the components involved to build this robot fulfill the constraints of low cost, low consumption and are as close as possible to industry standards. The input/output devices are designed as hardware bricks plugged on the board giving the capability to easily reconfigure the hardware platform.

Indeed, our hardware offers a set of simple and cheap reliable sensors and actuators which can easily be combined to achieve skillful works.

2.2 Components

The current SPIF robot is driven by a Motorola MC68302 [6] processor equipped with 1M bytes of E-PROM and 1M bytes of RAM. At the moment, the following sensors are plugged on the board: an ultrasound device and a luminous sensor. One low consumption engine is in charge of the steering and another one powers the driving wheels. The power is supplied by an off-the-shelf battery.

Communications with remote real-time applications take place on a radio line. Two serial lines are used for tests and debugging.

Up to now, three different hardware boards have been built using the same processor and software platform, strengthening the expected scalability of the hardware components. We plan to design a new board equipped with a more powerful chip by the end of 1997.

Figure not available Figure not available

(a) Robot 1 (b) Robot 2

Fig. 2. SPIF different hardware platforms

More hardware information is available at : http://www.enst.fr/~spif.

3 The software platform

We briefly describe the low level software platform: SPIF-OS.

3.1 The kernel

Kernel design. SPIF-OS is a real-time kernel dedicated to embedded systems. We decided to keep it as simple as possible to reduce the overhead during basic kernel operations such as scheduling, synchronization and memory management. This kernel, the core of which is written in C, supports real-time scheduling (HPF, EDF) and synchronization policies (PCP) [2].

A special care has been given to input/output management: standard device drivers are provided (engine, steering, ultrasound, luminous sensor, serial line, radio line) or being developed (camera). A high level input/output library is built on top of these drivers.

Scheduling. Threads are managed at kernel level: the basic process of SPIF is very similar to a POSIX thread. The following scheduling algorithms have been implemented and tested: HPF (High Priority First), EDF(Earliest Deadline First). A sporadic server [1] has also been implemented and tested.

Synchronization. The semaphores can be given a priority in order to use the Priority Ceiling Protocol (PCP). This protocol has been implemented and tested with tasks scheduled by EDF.

Time management and alarms. To operate properly, many devices need to receive and send signals at a fairly high frequency. The ultrasound device, for instance, should be sampled at 100 microseconds (10 kHz). If the interrupt timer is set to this frequency, the system (running on MC68302) is overloaded with clock interrupts and is not fast enough to handle them. Time management is based on the joint handling of two counters: one dedicated to the regular clock and another one for the handling of fast events i.e events frequency of which is greater than the one of the regular timer. On the actual MC68302 platform, the main timer is set at 10 milliseconds, and the second one can be as accurate as 100 microseconds.

Input output. A real-time system can be regarded as a three stage machine: data acquisition from sensors, data processing and then data output to actuators. Indeed, input/output support must provide efficient, timely, fast and reliable operations. In real-time embedded architectures input/output management is the essential issue and must be carefully handled. SPIF offers high level sensors and actuators management routines written in Ada.

4 Libraries and development tools

At this level, the industry de facto standards have driven our choices.

The FSU POSIX threads library. For compliance with emerging standards, the FSU POSIX [4] threads library has been ported to the SPIF kernel.

Development tools. We are using the GNU development suite to build our system. We chose this suite for cost, efficiency and portability reasons. It's a fully integrated development environment and Ada95 is available in gcc. A loader has been developed to download the target code, and a shell make it possible to interact with the robot through a serial line during the tests.

C library. A C library is supplied with all the usual facilities.

5 Ada95 for SPIF

In order to give the developers (students or engineers) high level programming tools that provide well defined time, task and synchronization management, we decided to provide support for Ada95 on our embedded system.

5.1 Engineering embedded software

While engineering real-time software, developers have to tackle hard timing and synchronization problems [11]. They need to describe these constraints in a simple and precise way. If they are given a language like C, they will have to handle themselves the low level synchronization mechanisms by using system calls. They won't get any semantic checking to prevent them from usual mistakes. They will spend a lot of time to develop basic synchronization and time tools instead of focusing on their end application.

The real-time annex, the protected types, the selective accept and asynchronous transfer of control are powerful and simple tools that give developers the capability to express clearly the requirements of real-time tasks.

Indeed, Ada95 is a powerful tool for engineering embedded software:

– Ada's portability allows to test and develop the software modules that will run on the target in a self-hosted environment before testing them in the cross-compiled environment.
– Developing real-time applications in C language requires to use the threads library for synchronization, time management and events handling. With Ada95 this sensitive work can be carried out at language level, ensuring a greater consistency.
– Fault tolerance can be implemented through the Ada exception mechanism,
– Data concurrency can be handled by many facilities, including protected types.
– Very low level data structures as memory or registers location and their contents can be precisely defined, preventing any compiler side effect. Reading inputs from the actuators or sending outputs to the actuators is no more handling byte streams but processing data unit, size and dynamic of which are precisely defined and checked.

The Ada compiler we chose for SPIF is GNAT (The GNU Ada compiler) because it fully supports Ada95 and its real-time and distributed annexes and because it is integrated in the gcc suite which offers a free efficient cross development environment for most of market processors.

5.2 The port of a GNAT run-time

The GNAT (version 3.0.7) run-time library is roughly split in two parts : routines that interface with the underlying operating systems and the other ones (callable by the user or those for internal use only). Almost all the routines are written in Ada. The first part is the package System.OS_Interface. This package contains all the kernel dependent routines (time management, etc) and is different for each operating system. The key point is that there is a POSIX [3] version of System.OS_Interface. If the OS is POSIX compliant, the port of the GNAT run-time is quite straightforward.

We used the GNAT run-time library provided for Linux, which is POSIX compliant, and we successfully implemented it on top of the FSU SPIF POSIX threads library.

Our tests focused on the following services of the SPIF Ada run-time: exception handling, protected types, task management, basic memory management.

5.3 GNAT extends the original model

Ada95 extends the original SPIF model by adding new capabilities: semantic control, high level input/output and, as we detail in the next paragraph, distribution.

Distributing embedded real-time using GLADE. GLADE[1] is the implementation of the distributed annex, Annex E, for GNAT [12] [13] . In the SPIF architecture, the medium supporting the distribution is a bidirectional half-duplex radio-line.

The GLADE run-time is organized around protocols; a protocol is actually an object (in the object oriented sense) which exports a few primitive operations such as Connect or Send. A particular protocol object is in charge of communicating with the same protocol on another partition. It offers a reliable way to blindly transmit byte streams between partitions without operating on the content of the stream. A new protocol can be easily added by defining a new object and overriding the necessary primitive operations.

For the SPIF project, a new protocol called Serial has been designed and implemented. This protocol deals with non-reliable and possibly half-duplex serial lines (such as a wire or a radio) and is based on well-known token passing algorithms [7]. Once this protocol has been tested, we just had to add it into the GLADE run-time to be able to use the full power of the distributed systems annex without further coding.

The TriAda demonstration. During the last TriAda conference held in Philadelphia we set up a demonstration that showed the current state of our works. An outline of this demonstration is given in figure 3. An Ada application on the robot communicates with an Ada application on the workstation using a radio line. This bidirectional communication is achieved using either messages or RPC's. The main task of the robot is to avoid obstacles. It executes commands, sends its speed, direction and distance of the closest obstacle as detected by the ultrasound sensor. The size of the whole embedded software is about 250 Kbytes. The mobile autonomous robot used is the one displayed on figure 1(b).

6 Applications

6.1 Teaching

The GNAT run-time on top of the SPIF architecture is a convenient platform to acquire significant knowledge about crafting embedded real-time systems. The

[1] GLADE is maintained by ACT Europe (contact@act-europe.fr)

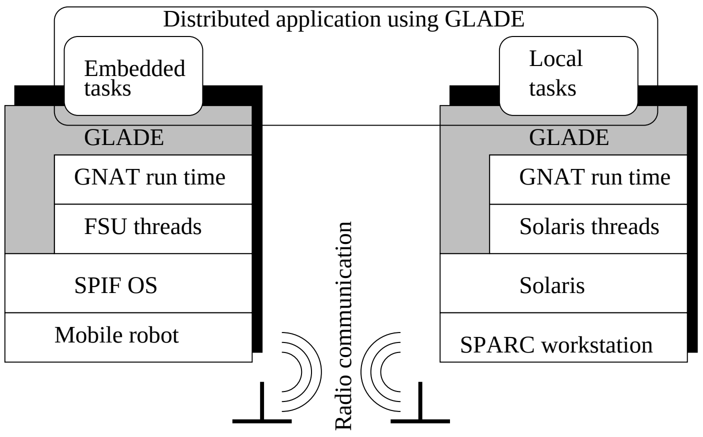


Fig. 3. Organization of the demonstration

software is engineered under a methodic approach: the timing and synchronization constraints are consistently defined, the devices interfaces and the format of the data sent and received are precisely specified. The sensors and actuators behavior is non ambiguously described and this enables a positive feed-back from the software to the hardware design.

SPIF is one of the competitor of the yearly TV broadcast challenge where robots designed by high-school students friendly fight against each other. Before the porting of the GNAT run-time, all the embedded software controlling the robot was written in C, task and time management were handled at low level by calls to the thread library. Starting this year, critical parts of the fighting strategy will be programmed in Ada.

6.2 Research

GLADE. Addressing the problems of distributing real-time embedded software is a major improvement provided by the distributed annex. We demonstrated the capabilities of GLADE by building a distributed application involving tasks running on the ground station and the mobile robot which communicate by remote procedures calls.

RTEMS. We ported RTEMS [8] [10] [9] on the SPIF hardware platform and we are now testing a GLADE/GNAT/RTEMS toolset on one of our robots.

7 Conclusion

The joint usage of Ada95 and SPIF layers shows that it is possible to design scalable reusable components for both hardware and software on embedded systems. Students and researchers are given the opportunity to efficiently build complex systems using basic software and hardware bricks. The design and tests of a new embedded system are achieved by adding features in a modular and incremental way.

The integration of the GNAT run-time into our prototyping architecture not only improves the abilities of our toolbox for designing, prototyping and testing real-time embedded software, but also extends the original model. The Ada95 distributed annex provides great capabilities for straightforward implementations of distributed applications. Porting GNAT and GLADE to the SPIF platform using whether RTEMS or SPIF-OS allows to address the problem of distributing real-time applications.

8 Acknowledgements

The authors are very grateful to Patrick Busch and Gerard Mouret who are in charge of the whole hardware layer. They designed and realized the different versions of the mobile robot and their various peripherals keeping the whole board scalable and modular.

References

1. J. Lehoczky B. Sprunt, L. Sha. Aperiodic task scheduling for hard-real-time systems. *The Journal of Real-Time Systems*, 1989.
2. A. Burns and A. Wellings. *Real-time systems and their programming languages.* Addisson-Wesley Publishing, 1989.
3. T.P. Baker E.W. Giering, F. Mueller. Implementing ada 9x features using posix threads: Design issues.
4. Florida State University Franck Mueller. Pthreads library interface, 1995.
5. IEEE. *P1003.1c POSIX Draft - Part1: System Application Program Interface*, 1994.
6. Motorola. *MC68302 Integrated Mutliprotocol Processor User's Manual*, 2nd edition, 1991.
7. S. J. Mullender. *Distributed Systems.* ACM Press, 1994.
8. Online Applications Research Corp. *RTEMS - C Applications Users' Guide*, January 1996.
9. Online Applications Research Corp. *RTEMS - Motorola MC68xxx C Applications Supplement*, January 1996.
10. M.Johannes P. Acuff, R. O'Guin and J. Sherril. *A Reusable Ada Real-Time Multiprocessing Executive for Military Systems.* Online Applications Research Corp., 1994.
11. J. Stankovic and K. Ramamritham. *Advances in Real-time Systems.* IEEE Computer Society Press, 1993.

12. L. Pautet Y. Kermarrec and E. Schonberg. Design document for the implementation of the distributed annex of ada9x in gnat. *Technical report, NYU*, 1995.
13. L. Pautet Y. Kermarrec and S. Tardieu. Garlic: generic ada reusable library for interpartitions communication. *Proceedung of the TriAda conference, Anaheim*, 1995.