TOWARDS IMPROVING ROBOTIC SOFTWARE REUSABILITY WITHOUT LOSING REAL-TIME CAPABILITIES

Frederic Pont and Roland Siegwart
Autonomous System Lab
Swiss Federal Institute of Technology Lausanne (EPFL)
1015 Lausanne, Switzerland

Keywords: autonomous mobile robots, real-time systems, software reusability, Linux

Abstract: We aim at improving sharability and reusability of software for autonomous mobile robots without sacrificing real-time capabilities. As a first step towards this goal, we focus on real-time Linux and we introduce the concept of a robotic hardware abstraction layer that provides for software reusability on different types of hardware and in real-time or non real-time context. We also present a preliminary implementation using RTAI Linux on the tour-guiding robot RoboX.

1 INTRODUCTION

As the complexity of autonomous mobile robots increases, it becomes difficult for researchers with different backgrounds and motivations to master all involved disciplines. The software running on complex systems often glues together these different disciplines in order for autonomous robots to achieve their complex missions.

In an effort to better organize the multiple pieces of software on mobile robots, software architectures have been studied for many years, from Brook's Subsumption (Brooks, 1986) to the three-layer architectures (Gat, 1997).

During the last few years, the idea of leveraging existing robotic software has emerged. The vision of being able to reuse reference implementations and to glue them with a new piece of code to obtain a robot able to perform complex tasks is appealing. One would also be able to compare its research against reference implementations of well-known methods.

Existing efforts focusing on better software reusability include the PlayerStage project (Gerkey et al., 2001) which proposes a hardware abstraction useful for reusability, but lacks support for time-constrained applications and GenoM (Fleury et al., 1997), which provides tools to facilitate the definition and integration of functional modules. The CLARAty architecture (Volpe et al., 2001) also provides a framework for reusable robotic components.

As we consider real-time capabilities a necessity for autonomous systems, we propose to tackle the reusability problem using a bottom-up approach. We will study the constraints and limitations that have to be defined for producing reusable software without losing the option to use the software with real-time constraints.

In this paper[1], we introduce the concept of a real-time capable robotic hardware abstraction layer. Such a layer will provide for reusability of software accessing a robot sensors and actuators, both in real-time and non real-time context. We propose to consider Linux extended for hard real-time as the underlying real-time operating system.

With this quest for reusable, sharable and portable software, we hope to improve the general quality of autonomous systems by making it easier for researchers to share and reuse quality pieces of software written by specialists.

The remainder of this paper is organized as follows. In the next section we argue that Linux extended with real-time capabilities is a candidate for being a reference real-time operating system for autonomous mobile robots. In section 3 we present how the concept of hardware abstraction layer can be used on top of real-time Linux for providing improved software portability and reusability, while preserving real-time capabilities. In section 4, we briefly present our initial implementation on the tour-guiding robot RoboX using

[1]The work reported in this paper has been supported in part by the IST Project RECSYS (IST-2001-32515).

RTAI Linux. Finally, we present possible future research orientations and some concluding remarks.

2 REAL-TIME LINUX

In the context of our quest for improved reusability of robotic software, we evaluate key criterias for a real-time operating system (RTOS) for autonomous mobile robots, and we argue that Linux is an excellent candidate.

Hard real-time capabilities are required to successfully complete autonomous missions. Key features including precision and security depend on them.

The standard Linux kernel does not provide hard real-time and was developed for maximal global throughput. Two different approaches exist to improve the standard Linux kernel response times (Dankwardt, 2002a). The first approach consists in improving the kernel preemptability to provide more responsiveness to applications (Dankwardt, 2002b) but does not provide hard real-time. The second approach is to insert a real-time micro kernel on top of the hardware and to run the standard Linux kernel as its lowest priority task that can be preempted at any time to run real-time tasks. This solution provides hard real-time and has been adopted by the main open-source real-time Linux projects, RTLinux (Hilton and Yodaiken, 2001) and RTAI Linux (Mantegazza et al., 2000). We will refer to this sub-kernel approach as real-time Linux.

Portability across different platforms is key. The same RTOS shall be available for most of the robotic platforms. If standard Linux is available for most commonly used platforms, portability of real-time extensions is still work in progress but is available for the main platforms like x86 and PowerPC.

Scalability from low-power or FPU-less CPUs to more powerful systems is required. Linux can be adapted to run on a wide range of systems. Linux's footprint is still relatively larger than specialized RTOS like VxWorks, but can usually be made small enough through careful configuration.

Reliability is very important for autonomous systems that will run for long periods of time without human intervention. Linux has became famous over the years for its reliability.

The above criterias can be considered as requirements that are all met by Linux with real-time extensions. Other advantages of using real-time Linux are worth mentioning, like *cost*, *available software base* and *desktop-target integration*. *Support* is sometimes seen as a weak point compared to commercial RTOS, but the support provided by the open-source community is usually excellent, even for mission critical developments (Norris, 2004).

In the next section, we present how we can take advantage of real-time Linux features to define a real-time capable robotic hardware abstraction layer that provides for better robotic software reusability.

3 REUSABILITY WITH REAL-TIME CAPABILITIES

One of the consequences of choosing real-time Linux as our RTOS is that real-time capabilities are only accessible from kernel space unless RTAI LXRT (Mantegazza et al., 2000) is available.

As we aim for real-time capable reusability, each reusable piece of code shall be written taking into account the usual limitations of kernel space development (Rubini and Corbet, 2001), like the non availability of libraries like libc. Pieces of code using such user space libraries can not be reused in real-time context. A list of requirements the reusable code shall comply with to ensure reusability in kernel space will have to be defined.

As a first step towards improving robotic software reusability without sacrificing real-time capabilities, we will consider how reusable code can access functionalities provided by the hardware it is running on, through a concept of hardware abstraction to extend reusability to different types of hardware. We will study in further research how a reusable piece of code can access functionalities provided by other reusable pieces of code.

The usual approach for improving portability across hardware platforms is to use a hardware abstraction layer (HAL). A HAL is a layer between the physical hardware of a computer and the software that runs on that computer. The function is to hide differences in hardware and therefore provide a consistent platform to run applications on.

The exact definition of the types of hardware abstracted by the HAL is out of the scope of this document. At this point, let us assume that the HAL abstracts sensors and actuators commonly used on autonomous mobile robots, like motors, encoders, bumpers or lasers.

Such an abstraction layer has already been defined for mobile robots (Vaughan et al., 2003), but it is not suitable for usage in kernel space, therefore limiting its availability to non real-time applications. In this document we introduce the concept of a HAL usable both in real-time and non real-time context.

We propose to implement this HAL as a set of C functions that can be access both from Linux user space and from kernel space. This allows code that uses this set of C function (HAL) to be reused in user space when no real-time is required, and in kernel space when real-time is necessary.

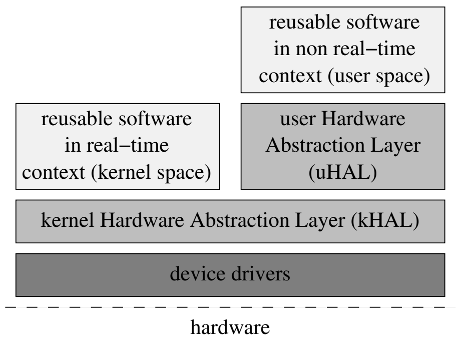
reusable software in non real−time context (user space)
reusable software in real−time context (kernel space)
user Hardware Abstraction Layer (uHAL)
kernel Hardware Abstraction Layer (kHAL)
device drivers
hardware

Figure 1: Reusable software with and without real-time needs on top of a multi-level hardware abstraction layer (HAL).

As illustrated on figure 1, we introduce a multi-level HAL that provide hardware abstraction usable in real-time context in kernel space, and in non real-time context in user space.

In kernel space, we define the kernel hardware abstraction layer (kHAL) that will be implemented by one or more kernel modules. The kHAL will provide access to the hardware through device drivers. The kHAL will be accessed directly by reusable code executed in a real-time task, and will provide an interface for communicating with user space.

In user space, we define the user hardware abstraction layer (uHAL) that we implement in the form of a library that can be linked with a reusable piece of code that needs access to HAL. The uHAL is responsible for accessing the hardware through the kHAL.

In order to provide the exact same functions, uHAL and kHAL share the same header file that describes the available functions. This header file is to be included by all reusable piece of code that want to access hardware functions through HAL.

By providing the exact same functions (HAL) in both user space and kernel space, we ensure that reusable pieces of code developed for accessing hardware through HAL can be executed both in non real-time (user space) and real-time (kernel space).

The communication between uHAL (user space) and kHAL (kernel space) can be done in different ways. In our first implementation, we have chosen to use the device file interface for transferring data between the user space uHAL and the kernel space kHAL.

Unix-like operating systems like Linux use the same system calls to interact with regular files on disk and to interact with I/O devices (Bovet and Cesati, 2003). For each type of abstracted hardware the HAL provides an interface to, kHAL is responsible for creating a character device file (e.g. */dev/hal/encoders* for encoders). uHAL can then use the appropriate device file to perform the usual file operations (*open(), close(), read(), write()*) to communicate with kHAL. The file operations executed by uHAL on a device file will be handled by the appropriate kHAL module. For each type of abstracted hardware, the type of data to be transferred between kHAL and uHAL has to be defined.

Note that the communication mechanisms between uHAL and kHAL can be improved without any consequences on the reusable code relying on HAL.

As an example, let us consider a simple piece of reusable code that reads the encoders of a mobile robot to perform some computation (e.g. odometry). If this piece of code uses HAL to read the encoders values, it can be used in the context of two different periodic tasks (real-time and non real-time), as illustrated on figure 2. Let us assume that the C function provided by HAL for reading encoders values is called *encoders_read()*.

If we reuse this piece of code within a task that needs real-time capabilities, the task will be executed in kernel space and the call to *encoders_read()* will be served directly by kHAL. If this reusable code is used in the context of a non real-time (user space) task, figure 2 illustrates how the encoders values are obtained using the uHAL library:

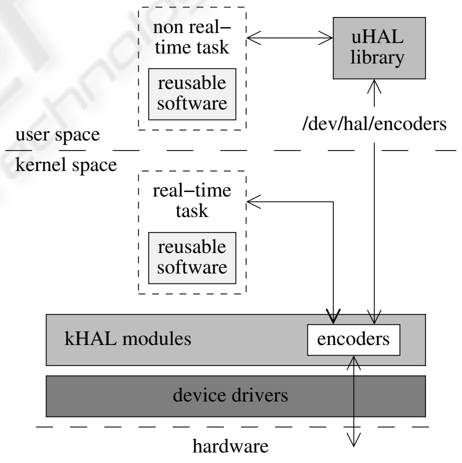

Figure 2: Reusable software accessing encoders through HAL executed in real-time and non real-time context.

1. the user-space application has been linked with the uHAL library.
2. the application calls a function, say *encoders_read()*, provided by the uHAL library.
3. the implementation of the *encoders_read()* function in the uHAL library opens a device file specific to the encoders (e.g. */dev/hal/encoders*) to communicate with kHAL.
4. kHAL reads the encoders values from its underlying hardware (through the appropriate device driver) and sends the values back to uHAL.

5. the user-space application obtains the encoders values.

This simple piece of reusable code can be executed in different context without any change to the code.

4 IMPLEMENTATION

The platform selected for the first implementation of the concepts presented above is the tour-guiding robot RoboX (Siegwart et al., 2003).

RoboX includes a PowerPC 750 clocked at 400MHz that was used for safety-critical tasks like navigation and obstacle avoidance, and a Pentium III dedicated for interaction. As a first step, we have replaced the current real-time operating system running on the PowerPC with RTAI Linux and developed the necessary device drivers. Then, we have defined and developed a HAL providing interfaces to the two laser range sensors (SICK LMS-200), to the eight surrounding bumpers, and to the two wheels motors and encoders.

The first reusable piece of code that was developed is a simplistic navigation algorithm that uses the interfaces provided by HAL to the laser range sensors and motors to move and avoid obstacles.

The reusable piece of code was successfully executed in the context of a non real-time periodic task in user space (using uHAL) and in the context of a real-time periodic task in kernel space (using kHAL).

Note that the algorithm reused in this first implementation is not interesting by itself, but its reusability in both real-time and non real-time context while accessing RoboX hardware is a confirmation that robotic software reusability with hardware abstraction and real-time capabilities is possible.

5 CONCLUSION

In this paper, we have presented how a robotic hardware abstraction layer (HAL) with real-time capabilities can be defined on top of real-time Linux for providing reusability of pieces of software accessing robot sensors and actuators. We have also described our preliminary implementation using RTAI Linux on the tour-guiding robot RoboX.

Future challenges include the definition of the abstraction interfaces provided by HAL and the extension of the concept for communication between reusable pieces of software. We will also investigate further the impact of robot configuration on HAL, as well as HAL integration into a higher level software architecture.

We hope that the concepts we have proposed in this paper will help robotic specialists focusing on their specific fields of interest by providing means for reusing existing quality software.

REFERENCES

Bovet, D. P. and Cesati, M. (2003). *Understanding the Linux Kernel, Second Edition*. O'Reilly and Associates, Inc.

Brooks, R. A. (1986). A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1):14– 23.

Dankwardt, K. (2002a). Real-time and Linux. *Embedded Linux Journal*, 7:6–10.

Dankwardt, K. (2002b). Real-time and Linux, part 2: The preemptible kernel. *Embedded Linux Journal*, 8:14–17.

Fleury, S., Herrb, M., and Chatila, R. (1997). GenoM: a tool for the specification and the implementation of operating modules in a distributed robot architecture. In *Proceedings of the International Conference on Intelligent Robots and Systems*, pages 842–848, Genoble, France.

Gat, E. (1997). On three-layer architectures. *Artificial Intelligence and Mobile Robots. MIT/AAAI Press.*

Gerkey, B., Vaughan, R., Sty, K., Howard, A., Sukhatme, G., and Mataric, M. (2001). Most valuable player: A robot device server for distributed control. In *Proceedings of the International Conference on Intelligent Robots and Systems*, pages 1226–1231, Wailea, Hawaii.

Hilton, E. F. and Yodaiken, V. (2001). Real-time applications with RTLinux. *Embedded Linux Journal*, 1:18–20, 22–25.

Mantegazza, P., Dozio, E. L., and Papacharalambous, S. (2000). RTAI: Real time application interface. *Linux Journal*, 72.

Norris, J. S. (2004). Mission-critical development with open source software: Lessons learned. *IEEE Software*, 21:42–49.

Rubini, A. and Corbet, J. (2001). *Linux Device Drivers: Second Edition*. O'Reilly and Associates, Inc.

Siegwart, R., Arras, K. O., Jensen, B., Philippsen, R., and Tomatis, N. (2003). Design, implementation and exploitation of a new fully autonomous tour guide robot. In *Proceedings of the 1st International Workshop on Advances in Service Robotics*, Bardolino, Italy.

Vaughan, R. T., Gerkey, B. P., and Howard, A. (2003). On device abstractions for portable, reusable robot code. In *Proceedings of the International Conference on Intelligent Robots and Systems*, pages 2121–2427, Las Vegs, Nevada.

Volpe, R., Nesnas, I., Estlin, T., Mutz, D., Petras, R., and Das, H. (2001). The CLARAty architecture for robotic autonomy. In *IEEE Aerospace Conference Proceedings*, pages 121–132, Big Sky, Montana.