Computer based control with real-time capabilities

MICHAL BLAHO, SAMUEL BIELKO, ĽUDOVÍT FARKAS, PETER FODREK
Institute of Control and Industrial Informatics
Faculty of Electrical Engineering and Information Technology
Slovak University of Technology
Ilkovičova 3, 812 19 Bratislava
SLOVAK REPUBLIC
michal.blaho@stuba.sk

Abstract: - Control systems in many fields require perfect time accuracy and reliability. That is why guaranteed real-time behavior is needed. It is evident by the growing trend of the use of fieldbuses in building of Networked Control Systems. The software layer has a very significant influence on the system performance and requires thorough examination before the utilisation of a specific real-time system. The goal of this paper is to present and examine run-time environments that support real-time capabilities. We compare the performance of RTAI and Java Real-Time System in controlling of a sample plant with quick response and present our previously developed real control systems that are based on the use of RTAI that runs the real-time layer of the control system and Java that is responsible for the user interaction.

Key-Words: - Real-time control, RTAI, Java Real-Time System, RC-RC filter, Networked Control Systems, Fieldbus

1 Introduction

The trend in building control systems is the use of Networked Control Systems (NCS). This idea is based on digital systems for collecting data from sensors and controlling actuators. The elements of NCS are connected via industrial communication fieldbuses or via wireless connections [1].
However, these processes require real-time command execution and data acquisition, so appropriate real-time operating systems or software tools have to be used. Examples of other systems that require real-time execution are banking systems, military systems, avionics or robotics, and providing real-time control for them is a challenging task [2, 3, 4]. The task is particularly challenging due to the need to reconciliate the hardware and the software layer of the control system. The software has a very significant influence on the resources and that is why resource analysis and evaluation of real-time systems at the aspect of software are needed [5].
According to [5], the most urgent issues in this area are the need to model various complex relationships among all kinds of resources, resource scheduling and resource optimization. The authors propose process algebra as a formal method to describe and analyze concurrent, asynchronous and nondeterministic behavior of real-time systems.
There are some literature references of the possibility of using GPUs (Graphics Processing Units) in scheduling of real-time tasks [6, 7]. However, it is a challenging task, but if succesful, the use of GPUs for hard real-time scheduling would be an important contribution to hard real-time operating systems. It is due to the vast computational performance of the GPUs.
According to [8], when prorotyping a real-time control system, it is useful to set and abide standards for the interfaces between the components, because it facilitates rapid prototyping from the simulation stage to the implementation stage. The basic idea when using this technique is that the controllers would not know if the information comes from the simulation code or from the sensor itself.
Some systems that require hard real-time signal processing (e.g. motion control systems), traditionally employ dedicated processors like DSPs (Digital Signal Processors) or microcontrollers. Altough these processors are specifically designed to carry out signal processing tasks by executing code while keeping a guaranteed interrupt latency, the CPUs in the standard computers can do the same if they run a real-time operating system like RTAI [9].
There are several applications that present the usefulness of computer based control systems with real-time capabilities. The majority of these systems are extended to real-time capabilities by extending

their operating systems by patches like RTAI or by using real-time Java for example.

The authors of [10] demonstrated online realization of experiments with hard real-time control using Linux RTAI, RTAI-XML server, Comedi and jRTAILab. Matlab, Simulink, OPC toolbox and a PLC (Programmable Logic Controller) were used in [11] in a real-time model predictive control application. Real-time remote control of a robot manipulator using Java and Client-Server Architecture was developed in [12]. An overview of existing solutions to Java Embedded real-time systems was presented in [13]. Authors of [14] present a reservation-based real-time Java environment for Windows NT entitled Chocolate. A real-time Java hardware and software system for use in embedded applications was presented in [15]. The paper [16] was dedicated to the challenges in implementing the real-time specification for Java in a commercial real-time Java virtual machine.

This paper presents the possible use of RTAI and real-time Java in building of control systems. It explains the types of real-time systems, Java Real-Time System, RTAI and compares their performance in controlling a test plant. We also present our previously developed real control systems that are based on the use of RTAI that runs the real-time layer of the control system and Java that is responsible for the user interaction.

The rest of this paper is organized as follows. The next section addresses real-time systems in general. Part three is devoted to Java Real-Time System and to explaining its features. RTAI is presented in part four. Parts five and six present the controlled system and the results of the comparison between individual software controllers. The existing RTAI based real control systems are shown in part seven, while the last part is the conclusion of this paper.

2 Real-Time

There are several definitions according which we can say if our system or application behaves real-time (RT). One of those is: “With real-time programming, the overall goal is to ensure that a system performs its tasks, in response to real-world events, before a defined deadline. Regardless of whether that deadline is measured in microseconds or days, as long as the task is finished before that deadline, the system is considered real-time" [17]. To simplify it we can say that the system is real-time if it behaves exactly how we expect it to behave in the meaning that it has two key qualities: predictability and determinism.

There are several types of real-time systems depending on when the task is allowed to execute. In *soft RT systems,* the task has to execute before deadline, however occassional missing of the deadline may occur without causing major error. That tells us, that in soft RT there still is some task completion value after missing deadline but it‘s decreasing rapidly. In *hard RT systems,* none of the tasks can miss the deadline, because its task completition value after deadline is zero and it can cause a fatal error. The third type is *isochronal RT* which is similar to hard RT. In isochronal RT, there is a “response window“ and outside it, the value of the task being executed is zero.

If we want our PC to provide RT behaviour it is not only about the power of hardware but it needs software support as well. That means we need an operating system with RT behaviour such as Solaris and while building our applications, we need to have access and control over every resource we need. Java Real-Time System (JRTS) with its Real-Time Specifications for Java (RTSJ) provides exactly this, and it broadens the use of Java to new fields. Similarly the RTAI patch for Linux extends the capabilities of the Linux OS for use in RT demanding applications.

3 Java

The Java project was initiated in 1991 at Sun Microsystems and its first stable version was released in 1995. It has become very popular in a short time and it still holds its position as one of the most popular programming languages. It is further developed under the Oracle Corporation. Java is an object-orientated, familiar and architecture independent programming language [18]. This means that after compiling the code it doesn’t need to be recompiled before running on other architectures. Java is distributed in several editions like Java card, Java ME, Java SE, Java EE, Java FX and others. The most common is Java SE which we have chosen for the comparison of control software qualities in controlling fast systems with JRTS and RTAI.

3.1 Java SE

The most general answer to why Java SE can't be used as the core engine in RT applications, would be: “Because it wasn’t designed in this way.” Java SE is designed to take the most from the power provided by today’s PCs and servers and to have huge throughput. RT systems usually have lower

throughput to make sure that all threads don't miss their deadlines.
The garbage collector used in Java SE is the main culprit of latency and jitter. The other one is the Just-in-Time collector. Java SE also has lack of possibilities to prioritize threads, so the programmer doesn't have access and control over all resources. Java SE is also missing high resolution timers and also the Java Virtual Machine (JVM) is not designed to be jitter free and deterministic.
Altough Java SE can't be used as the core for RT applications, there are ways of how to use it as the user interface. We will discuss that matter later in this paper.

3.2 Java Real-time System

Java Real-time System (JRTS) is a Java implementation compliant with Real Time Specification for Java. The syntax stays the same as with Java SE, but this implementation brings several improvements in it's design, which make it completely deterministic and predictable and therefore suitable for developing RT applications. The last stable version released is 2.2. JRTS can operate only under three operating systems and these must be preemptable, must have high-resolution timers and support priority inheritance, interrupt shielding and schedulable interrupts [17]. This three Operating systems are: Solaris 10 (update 6, 7). Red Hat Enterprise Linux MRG 1.1 Errata and SUSE Linux Enterprise Server 10 SP2 with Real Time Extension and of course their later releases. And now we can briefly examine the RT features which JRTS brings.

3.2.1 RT Garbage Collector

Real-Time Garbage Collector (RTGC) in JRTS is based on RTGC from Roger Henriksson. The garbage collector (GC) postpones it's work until high priority threads finish their work so it doesn't affect their activity. RTGC in JRTS is fully concurrent, mark and sweep, parallel, non-generational, non-moving, non-compacting and self-tuning. It uses tri-color marking scheme, fixed-sized memory blocks and works in three modes according to the current needs [17]. The modes are: normal, boosted and deterministic. RTGC switches between these modes according to the memory needs and the switches of modes are defined by preset memory thresholds. These modes differ in the number of working GC threads and the influence they have on the activity of low priority threads.

3.2.2 Scheduling and RT Threads

To schedule objects properly, JRTS cooperates with the OS. Everything about scheduling begins with an object from the `Schedulable` class. There are several subclasses of this class which represent threads and asynchronous events. JRTS allows us to set priorities, importance and periodic, aperiodic or sporadic modes of execution for this objects.
As for threads, JRTS brings two new types of threads represented by objects from the `RealtimeThread` (RTT) and `NoHeapRealtimeThread` (NHRT) classes. In general the main difference between these two types of threads is that RTTs should be used for soft RT tasks with their guaranteed latencies of maximum 200μs, while NHRTs don't have access to heap memory region so they are not influenced by garbage collection and that makes them perfect for hard RT requirements. The latencies of NHRT tasks are guaranteed to be under 20μs plus there may be jitter +/- 10μs.

3.2.3 Memory management

There are four memory regions in JRTS which extend the abstract class `MemoryArea.` The first region, which is basically the same as in Java SE is *Heap memory*.
The second memory region is *Immortal memory* which is created when JVM starts and also the objects in it live for the whole time the JVM lives.
The third memory region is called *Scoped memory*. Immortal and Scoped memory regions are both not affected by garbage collections.
The last memory region is *Physical memory* which might be used in cases we need to communicate with specialized hardware.

3.2.4 Support for Synchronization and Asynchronous Events Handling

Synchronization is one of the key features in RT applications. When neglected, huge latencies and priority changes may appear. JRTS has therefore two basic rules to avoid these problems:

1. Threads which are ready to run get the access to synchronized resources first.
2. Priority inversion control is used to avoid unwanted latencies when accessing synchronized resources.

These rules are applied in Wait Free Queues which are used to exchange data between objects with hard and soft RT behavior.
But JRTS also thinks of events which are not periodic or we don't know when they occur. They are handled thanks to the Asynchronous Event Handler which allows us to bind the execution of our code to any event inside or outside JVM. Java RTS is capable of handling tens of thousands asynchronous event handlings and we don't even need to care about the resources, because JVM solves it for us.
These principles allow us also asynchronous transfer of control of one place in code to another as a response to some event and asynchronous thread termination.

3.2.5 High Resolution Clock
JRTS provides two high resolution time classes, which allow us to do arithmetic operations with time and also time and date operations. These classes are `AbsoluteTime` and `RealtiveTime`. The first one represents a specific point in time given as a combination of milliseconds and nanoseconds. The second class represents an interval in time, which is handy when setting the behavior of periodic objects.
As for timers, they can be set to perform periodically - `PeriodicTimer` or one time only - `OneShotTimer`. They are dependent on hardware so we can get different results when using the same code on different systems. It is also crucial, that the OS we run JRTS on provides access to high resolution timers.

4 RTAI
Another way of achieving RT operation of our programs is to use RTAI. RTAI is a shortcut for Real Time Application Interface for Linux, an open source project started in 1998 by professor Paolo Mantagezza from Politecnico di Milano. We have chosen it due to its hard RT capabilities, easy way of downloading and the fact that it's free of charge. On the other hand the installation of RTAI isn't very simple and requires thorough reading of the installation manual.
RTAI is not an operating system, but it's a modification to Linux kernel which compensates the lack of RT support and helps to make the system more predictable and with lower latencies [19]. In other words it changes ordinary Linux into Linux with industrial qualities. It has support for many platforms as x86, x86_64, ARM, PowerPC etc. Additionally, there is also a tool called RTAI-Lab which is perfect for working with block diagrams.
RTAI works in two modes: User Mode and Kernel Mode. Working with User Mode is simpler due to easier communication with the rest of Linux and access to more resources. For our work we decided to use the Kernel Mode to gain the lowest possible latencies even though we ran into some difficulties here. They were associated with the more complicated way to communicate with the rest of the system, in our case the absence of certain functions in the `rtai_comedi` driver which can otherwise be found in the standard version.

4.1 RTAI Kernel Mode
In the Kernel Mode we work with the kernel modules. There are two basic functions `init_module()` and `cleanup_module()` in each kernel module. The `init_module()` function represents an entry point to the module and is called each time our module is inserted into running kernel. The purpose of it is to prepare our module for running and it is also a perfect place for allocation of resources and starting RT tasks. RT task RT_TASK and its handling function must be created to use the RT features of RTAI.
The `cleanup_module()` function is the exact opposite, meaning that everything we started with `init_module()` is stopped and discarded here. This function is called each time we remove the module from the kernel.
After writing our code we have to generate a kernel object `name_of_module.ko`. The best way to do this is to create a `Makefile` with a specific structure. The kernel module is put to function by the command `insmod name_of_module.ko` and removed by the command `rmmod name_of_module.ko`.

4.2 RTAI User Mode
RTAI User Mode is generally more popular than Kernel Mode because working with it is much simpler. It is just a simple GNU\Linux task with a `main()` function, where the RT tasks are started. To compile programs written in User Mode we can use GCC compiler, but it requires linking of special `flags` [20]. To run this programs, we usually need to be root, unless we have not specified otherwise.

4.3 RTAI Modules and Schedulers
Schedulers are the center of RTAI. There are two types of them - `rtai_lxrt` and `rtai_sched` and both can be used either in the Kernel Mode or in the

User mode. The difference between them is in the relation to schedulable objects. The `rtai_lxrt` sheduler supports hard RT and optional soft RT behavior for Linux schedulable objects and `rtai_sched` provides hard RT behavior not only to Linux schedulable objects but also for its own RTAI kernel tasks. [20] Even though there is a vast functionality associated with these schedulers, when we require more possibilities we can also use other modules such as [19]:

- `rtai_fifos` - a module which implements fifo services. It is used to handle and display data between kernel space and Linux space
- `rtai_shm` - a module which allows memory sharing between multiple RT tasks and linux processes synchronously
- `rtai_pqueue` - a module providing kernel-safe message queues
- `rtai_pthread` - a module providing hard RT threads, where each thread is a task
- `rtai_comedi` - a module representing functionality of comedi project
- `rtai_mbx` - a module providing functions of a message box.
- `rtai_msg` - a module providing functions for message manipulation
- `rtai_sem` - a module providing semaphores for fifo synchronisation
- `rtai_hal` - a module providing functions used by RT tasks to handle interruptions and communication with Linux processes

5 Control System

To test the RT qualities of the aforementioned platforms, we had to choose an appropriate object to control. We have chosen an RC-RC filter with quick response. A DAQ card and other necessary software tools have been used to communicate with the filter.

5.1 Used Hardware and Software

We used a dual core Intel processor PC with 2GB of RAM. We tested all three mentioned RT OS's and ultimately we chose SUSE Linux Enterprise Server 11 SP1 with Real Time Extension to be the one we run our JRTS experiments on. The RT kernel was 2.6.33.7. Even if we observed that Solaris is better for running JRTS as long as it provides more additional options like Thread Scheduling Visualizer, we faced a huge problem with hardware support here. For the RTAI experiments, we used version Magma under Linux kernel 2.6.38.8.

5.2 Data Acquisition Card

To communicate with the controlled system we have chosen the Advantech PCI-1711-U DAQ card. It is a common card with analog and digital inputs and outputs, FIFO memory, programmable counter, programmable gain and Automatic channel/gain scanning [21]. This card was more than sufficient for our experiments. However, we ran into problems here because the drivers were modified for the last time in 2006, so they didn't work under Linux kernel used by us. At that moment we had three options: modify the provided drivers, create our own drivers or use project Comedi. We chose the last option.

Project Comedi is an open source project providing drivers and tools for DAQ. It is very convenient to learn to work with this set of functions, because it supports hundreds of DAQ cards and therefore we don't have to learn a new set of functions every time we have some different DAQ at our disposal. What is more, it has RT support for most of the hardware [22].

The only problem we met at this stage was, that Comedi is C-based. So we had to use JNI (Java Native Interface) to communicate between JRTS and the DAQ card.

5.3 Controlled System

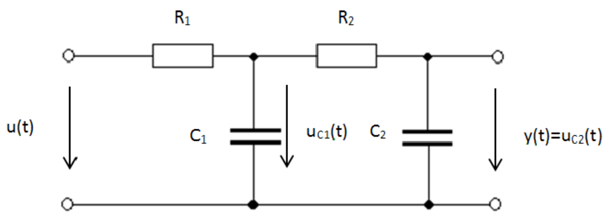

Fig.1: Scheme of RC-RC filter

Table 1: Values of RC-RC components

R_1	100 kΩ
R_2	1 MΩ
C_1	1 nF
C_2	1 nF

We chose an RC-RC filter in Fig.1 designed with component values stated in Table 1 to be the controlled plant suitable for our experiments. It is a linear second order system with the following differential equation (1):

$$u = R_1C_1R_2C_2\ddot{y} + [R_1C_1 + R_1C_2 + R_2C_2]\dot{y} + y$$
$$y = u_{C2} \qquad (1)$$

The aforementioned plant was controlled by a PI controller which parameters were set by means of the Optimal Module method (2).

$$G_r(s) = r_0 + \frac{r_{-1}}{s}$$

$$r_o = \frac{0.5T_2^2}{T1} - 1 \qquad r_{-1} = \frac{0.5+r_0}{T_2} \qquad (2)$$

We then converted the PI controller into a discrete PS controller. Its algorithm is in equation (3) while the sampling period for the conversion was set to 140 μs:

$$u(k) = u(k-1) + q_0 e(k) + q_1 e(k-1) \qquad (3)$$

6 Results of the experiments

We carried out two experiments according to which we wanted to compare the RT capabilities of the previously described software frameworks. The period of our discrete controller was set to 140μs in all cases. This period was chosen on behalf of the recommendation stated in [23] and also because it is suitable for comparison of RTT and NHRT when we take into consideration their guaranteed latencies. The first experiment was carried out without additional CPU load and the second was carried out with additional CPU load. The comparison of the performance of the individual software tasks helped us to choose the ultimate RT framework for our future work.

We ran an infinite cycle in which for the first 100 samples the desired system output was 1V, for the next 100 samples 2V, and then we gave the RC-RC filter time to get back to zero for the last 200 samples. This was repeating infinitely. We measured the first and the n-th cycle (in our case third) several times and then statistically evaluated. This procedure is pictured in Fig. 2.

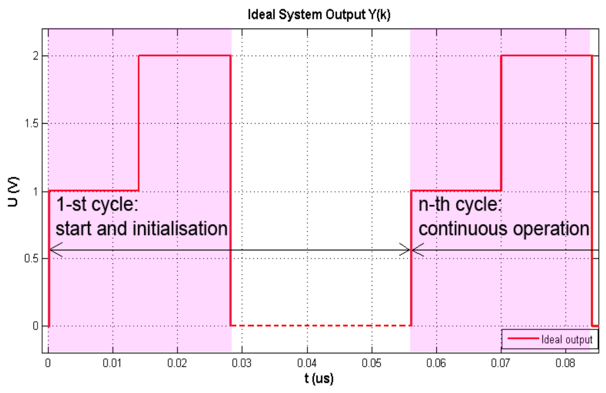

Fig.2: Illustration of the measuring procedure

At first we measured the system behavior by implementing the control algorithm with JRTS RTT, JRTS NHRT, Java SE thread and RTAI RT_TASK in n-th cycle without additional PC load. Meaning that no other applications or programs were running on the computer. The system output is in Fig.3, the corresponding controller output while measuring with RT systems in Fig.4, and the corresponding error rate in Fig. 5. We can see that the output in case of controlling with JRTS RTT is different compared to the other tasks. This is caused by relatively big latency of the RTT tasks (the tasks are executed sooner then required). In Fig.6 the plant output while controlling with Java SE is shown. Obviously in this case the output performs poorly and the answer for its behavior can be found in figure Fig.7. With Java SE each period was exceeded several times and the controller was therefore only jumping between its maximal and minimal output values (Fig.7). The corresponding error rate is in Fig. 8.

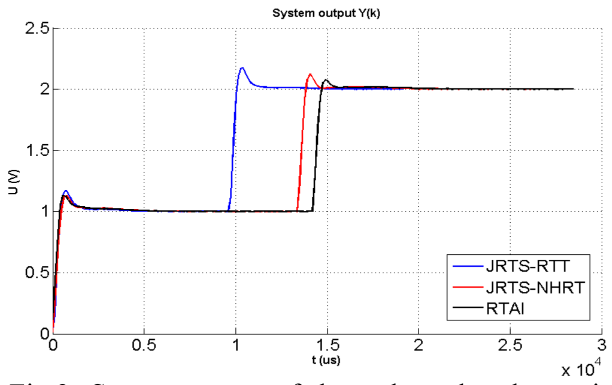

Fig.3: System output of the n-th cycle when using RTAI and JRTS without additional computer load

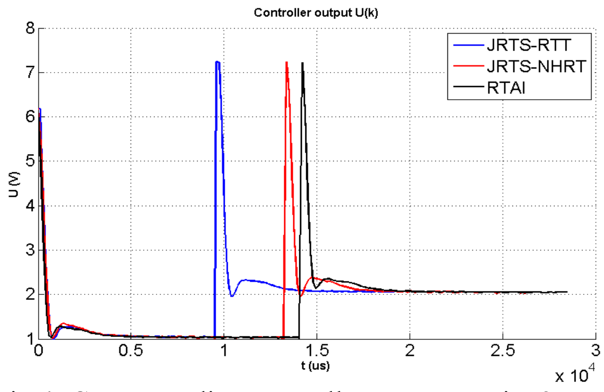

Fig.4: Corresponding controller output to Fig. 3.

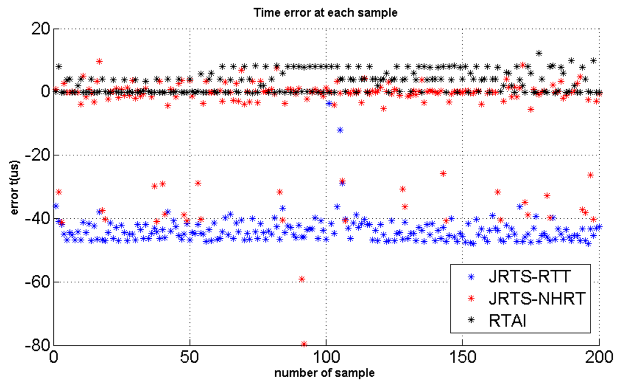

Fig.5: Corresponding error rate to Fig. 3

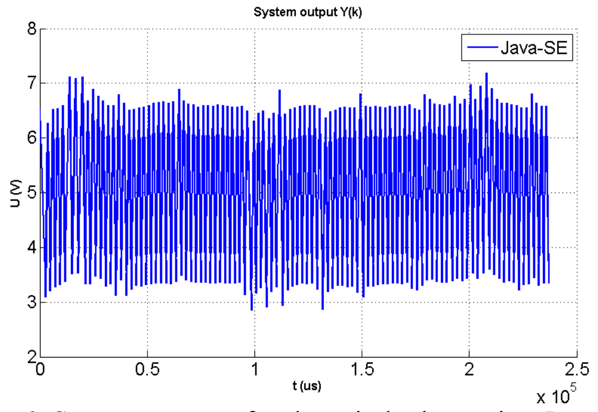

Fig.6: System output of n-th period when using Java SE without additional computer load

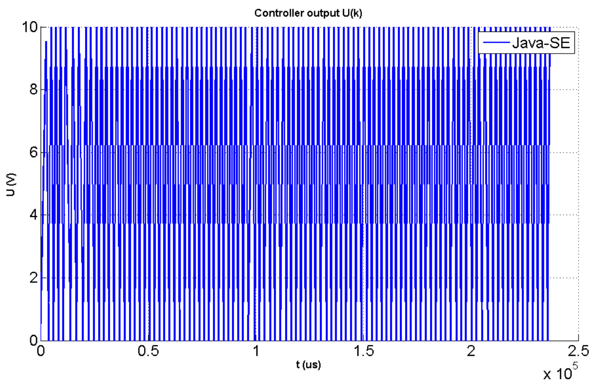

Fig.7: Corresponding controller output to Fig. 6

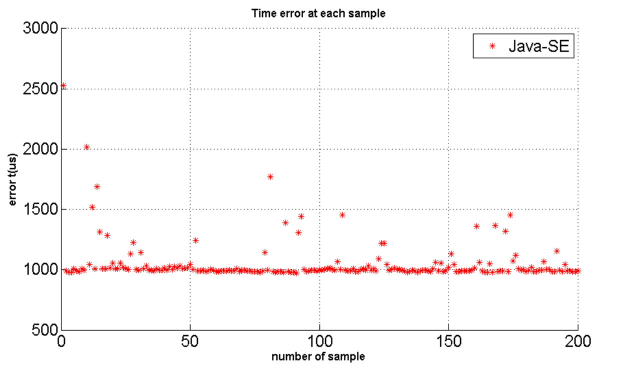

Fig.8: Corresponding error rate in each sample to Fig. 6.

In the second experiment we decided to not use Java SE anymore, since it showed unacceptable performance for RT applications. This time we added load to the system by running two Matlab benchmarks at the same time. The benchmarks were causing over 70% CPU usage. We also took in our focus the first period of the cycle. Let us call it “the initialization period”.

According to Fig.9 and Fig.10 we compare the system output in the initialization period and the n-th period under the load. In Fig.11 and Fig.12 we can see corresponding errors in each sample. In the initialization period, the most interesting part are the several samples of JRTS NHRT at the beginning. They are significantly shorter and it causes a big overregulation. However, in the n-th period, the behavior of NHRT is even better than when controlling without additional load. This is something we ran into under all supported operating systems when using JRTS. Therefore we think it is a property of the x86 architecture we were using. It has processor saving modes and generally it is not deterministic by its design at all. When under constant load, the processor does not execute its saving state, hence its performance is better compared to the case when the saving state is executed frequently. We didn’t want to explore RT behavior on expensive industrial machines but on regular computers. Our major tool was supposed to be software.

In the case of JRTS RTT we can’t see any significant change in sampling behavior. But since this kind of RT treads is designed for soft RT behavior, we can still say that is performing acceptably even in short periods. RTAI is performing comparably in both of the experiments. However, since we are working directly in Linux kernel it is not so surprising because it isn’t influenced by Linux processes and can access hardware much faster.

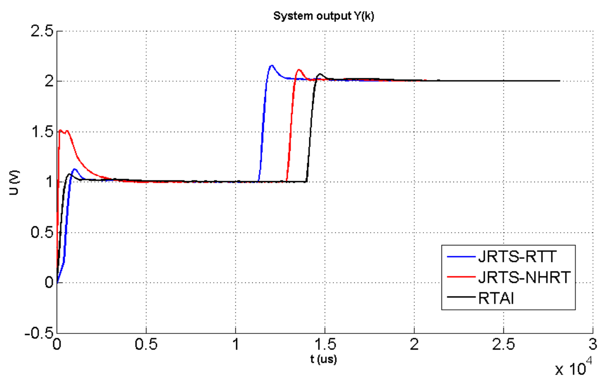

Fig.9: System output of the 1-st cycle when using RTAI and JRTS with additional computer load

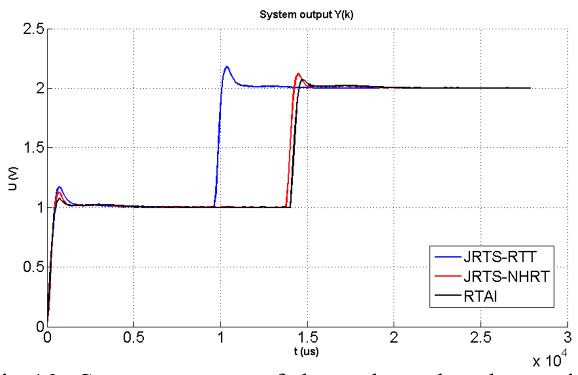

Fig.10: System output of the n-th cycle when using RTAI and JRTS with additional computer load

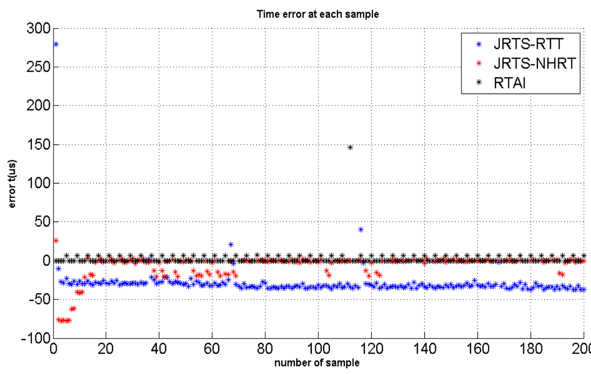

Fig.11: Corresponding error rate of each sample to Fig. 9

Now we present all the previous data in column charts and tables to make a better comparison of the qualities of the frameworks with the help of exact values (Fig. 13-15, Table 2-4).

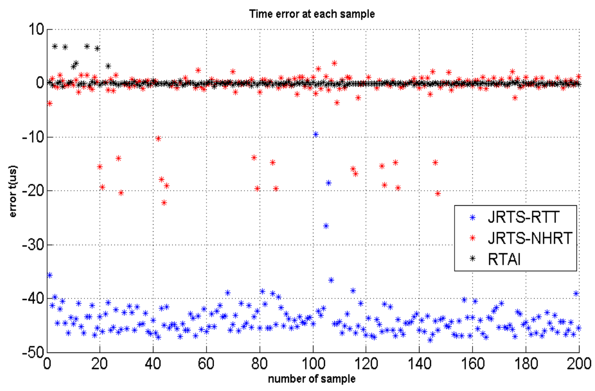

Fig.12: Corresponding error rate of each sample to Fig. 10

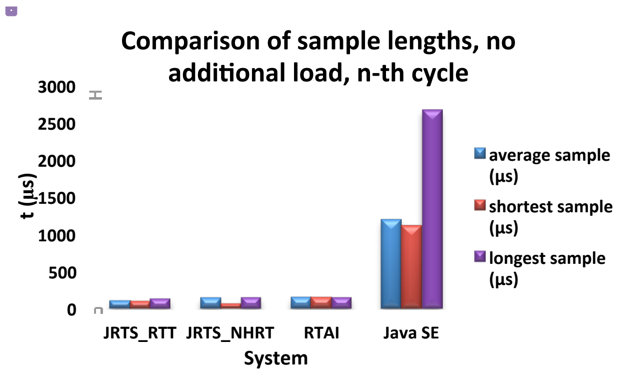

Fig.13: Comparison of sample lengths while measuring without additional PC load in n-th cycle

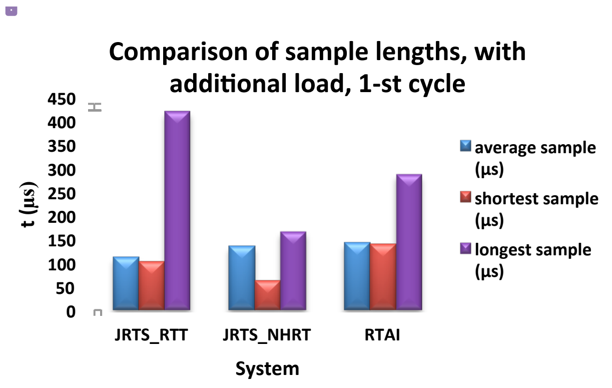

Fig.14: Comparison of sample lengths while measuring with additional PC load in 1-st cycle.

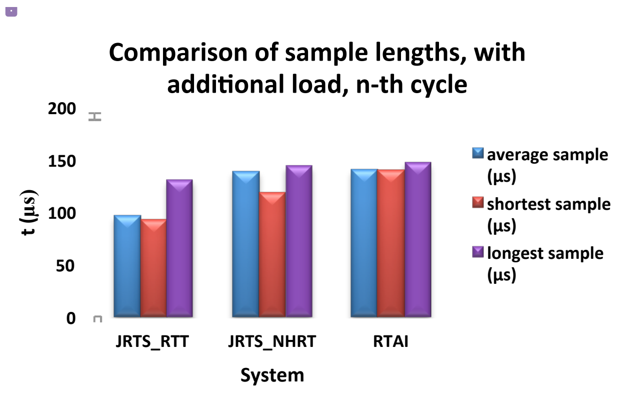

Fig.15: Comparison of sample lengths while measuring with additional PC load in n-th cycle

Table 2: statistical data related to measuring without additional PC load in n-th cycle

System	JRTS RTT	JRTS NHRT	RTAI	Java SE
Average sample (µs)	96.279	134.047	143.029	1 190
Shortest sample (µs)	92.006	60.416	139.693	1 113
Longest sample (µs)	136.294	149.632	152.091	2 665.5
Standard deviation	4.571	14.626	3.302	172.953
Overall time (µs)	19 256	26 809	28 606	238 010

Table 3: statistical data related to measuring with additional PC load in 1-st cycle

System	JRTS RTT	JRTS NHRT	RTAI
Average sample (µs)	111.206	134.415	142.271
Shortest sample (µs)	102.554	61.952	139.376
Longest sample (µs)	419.277	166.272	286.390
Standard deviation	23.364	15.141	10.626
Overall time (µs)	22 241	26 883	28 454

Table 4: Statistical data related to measuring with additional PC load in n-th cycle

System	JRTS RTT	JRTS NHRT	RTAI
Average sample (µs)	96.219	138.231	140.070
Shortest sample (µs)	92.314	117.760	139.299
Longest sample (µs)	130.458	143.680	146.851
Standard deviation	3.897	5.302	1.047
Overall time (µs)	19 244	27 646	28 014

In the last experiment we tried to calibrate RTAI. The main point of this experiment was, that the performance of RTAI can be influenced by measuring latencies and then including these measurements in RTAI settings. We measured that the latency of our computer is 6841ns. We then included this information in the RTAI configuration, recompiled it, reinstalled it and measured the latencies again. This time we gained a value of 8747ns and we did the procedure again. The results are shown on figures Fig.16 and Fig.17.

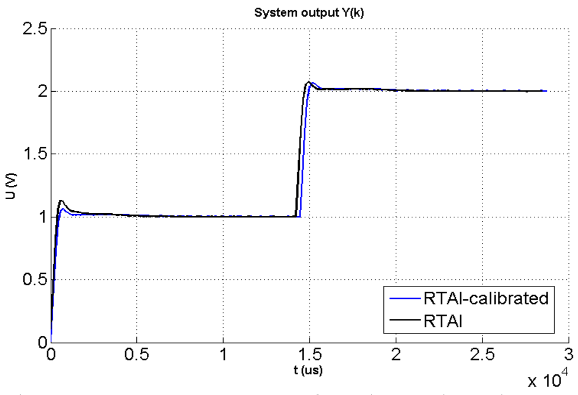

Fig.16: System output of n-th cycle when using RTAI before and after calibration

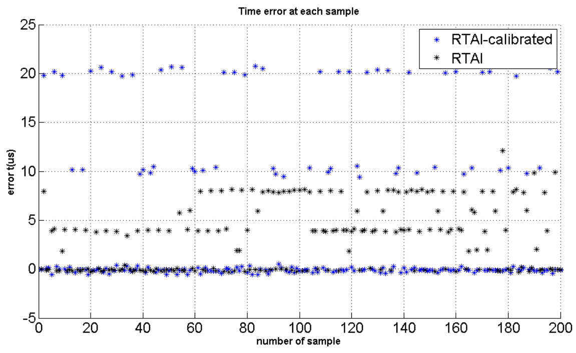

Fig.17: Corresponding error rate of each sample to Fig.16

7 Existing RTAI based control systems

After evaluating the data from the tables above, we concluded that the framework with the most predictable behavior is RTAI. However, the use of Java is not excluded.

Our solution is the use of RTAI extension of the standard Linux kernel combined with a Java based graphical user interface (GUI). The use of RTAI combined with an upper level interaction is also described in [8, 24].

Because the modification of the Linux operating system for real-time control does not change the functionality of the original system, we can use all of the software features that the OS offers in the user space. The limitation for the selection of the programming environment is that it has to provide resources for using interprocess communication and all of the enhanced functionality the programmer wants to implement in the GUI (image processing, sound processing, database, etc.) [25].

The control system runs on a standard PC and it is divided into two parts:

- lower control level (real-time, kernel space)
- upper control level (non real-time, user space)

The lower control level (procedural level) consists of sensors, actuators and the control PC interconnected by a fieldbus. The software controller is a set of RTAI kernel modules that communicate with the user space by the tools of Linux interprocess communication:

- RT FIFO pipes
- shared memory
- mailboxes

The upper level is responsible for communication with the user and for the tasks, that do not have to run in real-time. It can perform visualization and complex tasks over the collected data (database, image processing, sound processing, statistical evaluation).

We decided to create the upper control level in the Java programming language because it is a platform independent, object oriented and very popular programming language with a vast number of libraries that satisfy all needs for the user interface. The communication with the lower level is handled by the Java Native Interface (JNI) and the aforementioned methods of interprocess communication.

By using the Standard Widget Toolkit we were able to achieve the native look and feel of the underlying OS.

Examples of our existing control systems based on the described software model are a plasma metal cutting machine Fig. 18, 19 and a computer controlled water fountain Fig. 20, 21.

Both of the existing control systems utilize the combination of the best features of both RTAI and Java in terms of RT performance and user experience. A feature that contributes to the security of the systems is the protection of the computer by the Java interpreter from breaking the OS. It is very important because even if the GUI crashes, the RT task can be safely ended, thus bringing the machine to a safe state [25].

Both applications show that even if JRTS is not used as the RT software framework, Java SE can be used in control in combination with RTAI.

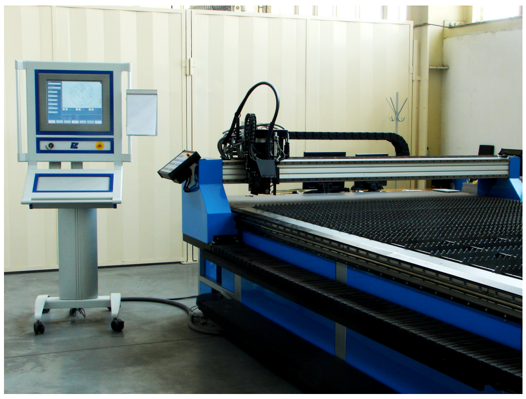
Fig.18: Plasma cutting machine.

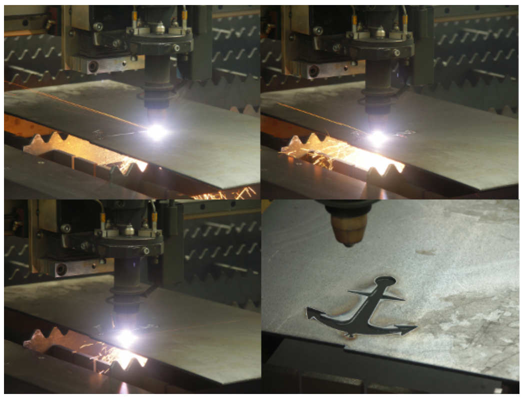
Fig.19: Plasma cutting machine - detail.

Fig.20: Water fountain

Fig.21: Water fountain - detail.

7 Conclusion

This paper has presented and examined the software run-time environments that support real-time capabilities. Two software frameworks have been presented. The first of them was the Java Real-Time System. We examined its garbage collector, scheduling and RT threads, memory management, synchronization and event handling and the high resolution clock.

The second software framework was the RTAI modification of the Linux kernel. This modification brings the support for RT kernel modules that run deterministically and independently on the user interface. However, it also offers support for RTAI user mode.

In order to choose the better of the presented software frameworks for our future work, we had to put togehter a control system and compare their performance. The chosen plant was an RC-RC filter with a quick response. We programmed a PI controller in each of the frameworks and ran them with the same parameters. The comparison shows that the performance of RTAI is more deterministic, thus it is better suitable for our applications.

However, the use of Java in control was not excluded by that statement, because two examples of networked control systems based on two-level software structure have been presented. The first of them was a plasma cutting machine and the second was a computer operated water fountain with sound and light effects. Both of the examples show the advantages of the proposed architecture by running time-critical RT threads in the kernel and the UI which is not time-critical in Java. The Java environment provides an easy way of creating rich and intuitive user interfaces and the ability of sound and graphics processing with the use of additional libraries. The communication between the two levels is managed by means of RT fifo pipes and shared memory.

8 Acknowledgments

This work was supported by the scientific grant agency of the Ministry of Education of the Slovak Republic and of Slovak Academy of Sciences under contract No. VEGA-1/1256/12. It was supported by the Slovak Research and Development Agency under contract No. APVV-0211-10 as well.

References:

[1] T. Murgaš, P. Fodrek, Ľ. Farkas, Networked Control System Using Linux Real Time Application Interface, *Recent Researches in Engineering and Automatic Control* : ECC´11: 2nd European Conference of Control, ECME´11: 2nd European Conference of Mechanical Engineering, ECCIE´11: 2nd European Conference of Civil Engineering, ECCE´11: 2nd European Conference of Chemical Engineering; Puerto De La Cruz, Tenerife, Spain, 10.-12. December 2011. - : WSEAS Press, 2011. - ISBN 978-1-61804-057-2. - pp. 53-58

[2] R. Bachnak, C. Steidley, M. Mendez, J. Esparza, D. Davis, Real-Time Control of a Remotely Operated Vessel, *Proceedings of the 5th WSEAS Int. Conf. on Signal Processing, Robotics and Automation*, Madrid, Spain, February 15-17, 2006 (pp188-194), ISBN: 960-8457-41-6

[3] B. Tabakova, Design and Implementation of Real-Time Fuzzy Control for Thermodynamic Plant, *Proceedings of the 9th WSEAS International Conference on FUZZY SYSTEMS (FS'08)*, Sofia, Bulgaria, May 2-4, 2008, ISBN: 978-960-6766-57-2

[4] M. Blaho, S. Bielko, P. Fodrek, T. Murgaš, Deterministic platforms for real-time control systems, *Latest Trends in Circuits, Automatic Control and Signal Processing : Proceedings of the 3rd International Conference on Circuits, Systems, Control, Signals (CSCS'12)*, Barcelona, Spain, 17-19 October, 2012, ISSN 1790-5117, pp. 249-253

[5] Y. Zhu, Z. Huang, G. Zhang, Modeling and Analysis of Real-Time Software based on Resource Communicating Sequential Process, *Proceedings of the International Conference on Information Engineering and Computer Science (ICIECS 2009)*, 19-20 December, 2009.

[6] P. Fodrek, Ľ. Farkas, T. Murgaš, Realtime Scheduling Using GPUs - Proof of Feasibility, *Recent Researches in Circuits, Systems, Communications & Computers : ECS'11: 2nd European Conference of Systems; ECCTD'11: 2nd European Conference of Circuits Technology and Devices; ECCOM'11: 2nd European Conference of Communications; ECCS'11: 2nd European Conference of Computer Science*; Puerto De La Cruz, Tenerife, Spain, 10 - 12 December, 2011, ISBN 978-1-61804-056-5, pp. 285-289.

[7] P. Fodrek, Ľ. Farkas, M. Blaho, M. Foltin, J. Hnát, T. Murgaš, Real-Time Scheduling Using GPUs-Advanced and More Accurate Proof of Feasibility, *Journal of Communication and*

Computer, Vol. 9, No. 8 (2012), ISSN 1548-7709, pp. 863-871

[8] L. García, M. J. López, J. Lorenzo, Hardware/Software Environment for Process Identification, Robust Controller Design and Hard Real Time Implementation, *Proceedings of the 5th WSEAS International Conference on Telecommunications and Informatics*, Istanbul, Turkey, May 27-29, 2006, pp. 503-508.

[9] M. Chiandone, S. Cleva, G. Sulligoi, PC-based feedback acceleration control using Linux RTAI, *13th European Conference on Power Electronix and Applications (EPE '09)*, 8-10 September 2009.

[10] Z. Janík, K. Žáková, Real-time experiments in remote laboratories based on RTAI, *15th International Conference on Interactive Collaborative Learning (ICL 2012)*, 26-28 September, 2012.

[11] M. Mrosko, E. Miklovičová, Real-Time Model Predictive Control, *Mathematical Methods and Techniques in Engineering and Environmental Science : 4th WSEAS International Conference on Visualization, Imaging and Simulation*, Catania, Sicily, November 3-5, 2011, pp. 138-143.

[12] F. M. Raimondi, L. S. Ciancimino, M. Melluso, Real-Time Remote Control of a Robot Manipulator using Java and Client-Server Architecture, *Proceedings of the 7th WSEAS international conference on Automatic control, modeling and simulation (ACMOS '05)*, Prague, 2005, pp. 122-126.

[13] M. T. Higuera-Toledano, Towards an Analysis of Garbage Collection Techniques for Embedded Real-Time Java Systems, *Proceedings of the 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, 2006, pp. 97 - 100.

[14] D. de Niz, R. Rajkumar, Chocolate: a reservation-based real-time Java environment on Windows/NT, *6th IEEE Real-Time Technology and Applications Symposium (RTAS 2000)*, 2000, pp. 266 - 275.

[15] M. Pfeffer, S. Uhrig, Th. Ungerer, U. Brinkschulte, A real-time Java system on a multithreaded Java microcontroller, *Proceedings of the 5th International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC 2002)*, 2002, pp. 34 - 41.

[16] M. H. Dawson, Challenges in Implementing the Real-Time Specification for Java (RTSJ) in a Commercial Real-Time Java Virtual Machine, *Proceedings of the 11th International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC 2008)*, 2008, pp. 241 - 274.

[17] G. Bollella, E. J. Bruno, *Real-Time Java Programming with Java RTS*, USA Crawfordsville: R.R. Donelley, 2009. 409 p, ISBN 978-0-13-714298-9 .

[18] ORACLE, *Java Help Center* [online], [quoted on 2013-03-26], Available online at: http://www.java.com/en/download/faq/index_general.xml

[19] DIPARTIMENTO DI INGEGNERIA AEROSPAZILE, *DIAPM RTAI - Beginner's Guide* [online]. Milano: Politecnico di Milano, 2006, [quoted on 2013-03-26]. Available for download at: https://www.rtai.org/index.php?module=documents&JAS_DocumentManager_op=downloadFile&JAS_File_id=32

[20] G. Racciu, P. Mantegezza, *RTAI 3.4 User Manual* [online], 2006 [quoted on 2013-03-26]. Available for download at: https://www.rtai.org/index.php?module=documents&JAS_DocumentManager_op=viewDocument&JAS_Document_id=44

[21] ADVANTECH, *Datasheet PCI-1711* [online], [quoted on 2013-03-26], Available for download at: http://downloadt.advantech.com/ProductFile/Downloadfile3/1-32A92F/PCI-1711L_DS.pdf

[22] COMEDI.org, Comedi documentation [online], [quoted on 2013-03-26], Available online at: http://comedi.org/doc/index.html

[23] L. Harsányi, et al., *Teória automatického riadenia*, Bratislava: Slovenská technická univerzita v Bratislave, 1998, 216 p, ISBN 80-227-1098-9.

[24] L. García, M. J. López, J. Lorenzo, Hardware-in-the-loop Environment for Control Systems evaluation under Linux/RTAI, *Proceedings of the 6th WSEAS International Conference on Applied Computer Science*, Tenerife, Canary Islands, Spain, December 16-18, 2006.

[25] T. Murgaš, P. Fodrek, Ľ. Farkas, Using Open Source Technology in Building of Control Systems, *Latest Trends in Information Technology : Proc. of the 1st WSEAS International Conference on Information Technology and Computer Networks (ITCN'12)*, Vienna, Austria, 10-12 November 2012, WSEAS Press, 2012, ISBN 978-1-61804-134-0, pp. 55-60.