WP3 - Market Analysis

Deliverable D3.1 - Feedback from RTOS users

WP3 - Market Analysis : Deliverable D3.1 - Feedback from RTOS users

by Adrian Matellanes, Agnes Lanusse, Francois-Xavier Russotto, and Stanislav Benes

Published October 2002

Table of Contents

List of Tables

Document Presentation

Table 1 Project Coordinator

Organisation	UPVLV
Responsible person	Alfons Crespo
Address	Camino Vera, 14 46022 Valencia, Spain
Phone:	+34 963877576
Fax:	+34 963877576
Email:	alfons@disca.upv.es

Table 2 Participant List

Role	Id.	Participant Name	Partcipant acronym
CO	1	Universidad Politecnica de Valencia	UPVLC
CR	2	Scuola Superiore Santa Anna	SSSA
CR	3	Czech Technical University in Prague	CTU
CR	4	CEA/DRT/LIST/DTSI	CEA
CR	5	Unicontrols	UC
CR	6	MNIS	MNIS
CR	7	Visual Tools S.A.	VT

Table 3 Document version

Release	Date	Reason of change
1_0	15/01/2003	First release
2_0	15/04/03	Second release

Chapter 1. Introduction

The goal of this document is to present some feedback from the customers and users of RTOSes.

We have made an overall analysis of the general requirements in the Real-Time and Embedded Systems (RTES) arena and have specially analyzed the needs from the Robotics, Multimedia and Process Control systems development points of view.

We find of special interest the fault-tolerance characteristics and worth a dedicated chapter, in it we present the most relevant fault-tolerance mechanisms that lack in Open RTOS.

The structure of the document is the following: First we will give a brief general analysis, then we will discuss in more detail the fault-tolerance requirements; in Chapter 4 we will present Process Control requirements, then Multimedia requirements and finally the Robotics requirements.

Chapter 2. General Analysis

2.1. Introduction

In this chapter we will give an overview of the general requirements of RTES development.

The main sources of information for the specification of what is going to be done in OCERA are, the RTOS analysis that we have done in WP1, the analysis of the new trends in RTOS research, and feedback from RTOS users.

On a general analysis, we found also a big segmentation in both, the proprietary and the open source communities, which can be observed in "D1.1. RTOSes Analysis". There is also a lot of information spread around that makes difficult for a user to choose a particular RTOS, apart, of course, from the technical issues that makes the analysis really difficult.

The number of users of RTOSes is increasing and the RTES marketplace keeps growing at, probably, the highest rate in the Information Technologies area. As the investments in embedded systems increase and the number of users multiplicate, the market keeps segmentating. The intrinsic complexity of the RTES and the variety of applications they are used for makes it difficult to evaluate and decide which RTOS fits a particular need.

When focusing on free software RTOS things get even worse, it is important to notice the lack of apropriate documentation, clear specifications and the fact that lots of efforts are being diversified between the different projects.

2.2. General Requirements

Here it is a list with requirements of a general character; we do not include the common functionality that is already available in almost all RTOS. For a list of currently available functionalities, refer to "D1.1 RTOSes Analysis":

- Standard API

  The use of a standard or close-to-standard API is most of the times mandatory. Currently POSIX is the approach preferred by customers, if available

- Common API between user space and kernel space
- Correct and up-to-date documentation and examples

  When creating development tools or environments, documentation is sometimes underestimated as a powerful reason why users make one choice or another. End users find very important to have a complete and up-to-date documentation and to have examples for every outstanding feature.

- Easy configuration and installation.

  The configuration and installation of OCERA should be easy and smooth. We should also provide an easy way to compile applications.

2.3. Embedded Systems Requirements

- Tools for configuration, generation and deployment

  We have found a lack for an integrated environment where users could develop, test, debug and deploy their RTES. We realized that there are several tools and environ-

ments for each task (develop, test, debug and deploy) but they are incomplete and difficult to integrate with each other. **To provide such a framework is one of the goals of OCERA.**

- Remote debugger
  The ability to debug through a serial port is an important feature we should consider. There is already support for that in some systems and sometimes there are embedded systems whose interface with the "exterior" is just a serial cable.

Chapter 3. Fault tolerance mechanisms analysis

3.1. Fault-tolerance in open RTOS

Fault-tolerance involves many different aspects that have been well studied in the literature. They concerned design issue, high level scheduling, specific replication techniques, communication protocols as well as specific hardware components. However, if generic approaches have been described and specific solutions implemented within proprietary systems, until recently very few things had been done towards openness and standardization under this topic.

Though the wonderful job done with the POSIX effort of standardization has been successful in the real-time, (major Operating Systems providers are now shifting to POSIX interface[1]), a lot of work has still to be done, particularly concerning fault-tolerant issues.

Some commercial Operating Systems providers targeted to safety-critical applications have developed other the years specific features to provide hard real-time performance as seen in WP1. Among them, LynxOS which is used for mission-critical applications has developed specific interrupt management and scheduling mechanisms in order to provide determinism and linear performance scalability. LynxOS also fully supports processor and MMU control functions thus insulating one faulty process from the others. Besides protecting memory MMU provides significant efficiency gains thanks to virtual addressing. In additional to these features available in the 4.0 version, a high-availability package can be purchased. It offers: a Fault Management Framework for managing fault-tolerant takeover operation, Redundant systems slot (SRSS) support and Compact PCI, Hot Swap support for I/O boards.

Other initiatives have been undertaken in specific domains such as automotive to provide support for real distributed systems and fault-tolerance (OSEK).

But things are rapidly changing mainly, thanks to the open source community and its impact on proprietary Operating Systems evolution strategy.

Recently, the telephone equipment manufacturers have considered using Linux for their demanding carrier applications. Though Linux was a good candidate for them, the characteristics of Public Switched Telephone Network (PSTN) applications required to operate reliably and guarantee accessibility in emergency situations and needed thus further developments towards reaching these high demanding requirements. They created the Open Source Development Lab in January 2000 to promote the development of a carrier grade Linux that would provide for high-availability. The OSDL has just issued Requirements definitions in June 2002.

Many of the directions promoted within this framework will contribute to provide the community with useful components that will serve also the goals of OCERA since many projects will be started

We can cite work towards hardened driver support, development of whatchhdog timers, ethernet link aggregation (bonding project under sourceforge) to provide for safe redundant networking, Raid support, data check-pointing , resource monitoring among many other directions

Open source community has also developed tools for large distributed clusters (beowulf using PVM), that may provide useful hints for OCERA. Other developments within the grid project may also provide material in the near future. All these developments will have to be constantly surveyed during the project.

Finally we may take advantage of academic work done in the domain of scheduling which is quite dynamic and can serve our objectives for advanced dynamic reconfiguration.

As a conclusion we can say that almost nothing can be reused from open-source community in order to implement fault-tolerance at present, however we can benefit from the great experience obtained in academic research and in industrial implementations to build rapidly basic mechanisms.

3.2. Requirements

- *error detection and signaling*
  mechanisms to detect errors and signal these errors are mandatory. It is necessary that they signal both code errors and temporal errors (deadline miss).

- *reconfiguration mechanisms*
  reconfiguration mechanisms are mandatory too to decide what to do when an error occurs. A fault-tolerance monitoring component has to decide if the application must be stopped or not, if a reconfiguration is possible and apply a new mode at the application level or at task level. This decision has as consequence to stop certain tasks, change priorities or behavior of others and possibly create new ones.
  This monitoring component must exploit information on tasks and tasks behaviors as well as on application modes conditions of change. This information must be given explicitly by programmers at design time. It will then has to propagate new constraints to a QoS component and to a dynamic scheduler at RT level.

- *dynamic scheduling and QoS Scheduling*
  mechanisms to take into account dynamically new scheduling constraints. A QoS scheduler could take into account alternative behaviors for tasks and importance criteria. It should be able to reschedule resources on change of tasks constraints. The RT-scheduler should be able to take into account orders to stop tasks and reschedule automatically tasks left.

- *error recovering mechanisms*
  error recovering mechanisms that permit to mask errors or to get back in a safe consistent state are required to implement real fault-tolerant systems and not only a system. These may be transactions systems and / or redundancy management mechanisms
  error recovering mechanisms implementation implies that redundancy can be effective in a distributed system. This requires safe end-to-end communications and specific mechanisms for replicas check-pointing synchronization.

- *fault-tolerant communication system*
  Communications must be atomic and a failure of a node must be detectable. Communication times must be predictable.
  Specific communication protocols must be developed to support redundancy management (distributed check-pointing and state transmission must be atomic)
  Global time management must be provided.

- *general debugging tools and tracing facilities*
  mechanisms to trace systems behavior are mandatory and measure times are mandatory to verify hypothesis on temporal behavior of applications.
  debugging tools must handle thread level events.

- *user specification and configuration tools*

The user must have a way to specify in a declarative way temporal constraints on tasks along with information related to fault-tolerance. Behaviors to be adapted when faults occur must be explicitly entered in the system.

Tools to exploit this information and configure adequately the RTOS components are also required.

Tools to check validity and feasibility of application given temporal constraints and architecture characteristics, would be very useful.

Notes

1. LynusWorks for instance has renamed a lot of their LynxOS system calls in order to be POSIX compliant

Chapter 4. Process Control Requirements

4.1. Introduction

Process control covers mainly applications in fields of power plants, the gas industry and the traffic.

One important common feature of these application fields is time criticality, because controlling actions must be performed by a stated time limit, which is ordered by technological characterictics of the controlled object. Therefore real-time systems are usually used in this region.

Second important common feature of these applications is that a fault of the controlling system can cause a demage of property, an injury or death of persons. In such a case these applications are generally called as critical applications.

4.2. Problems

To design controlling system in process control application field usually brings following typical problems:

- We have to accomplish required time limits for control actions performing. The limits are usually unrealizable in an environment of a standard operating system.
- We have to use a sufficiently reliable hardware and software according to the level of criticality of the controlled application.
- Development environments for real-time operating systems aren't usually so comfortable as it is common at other operating systems, so that SW development isn't effective enough.
- We have to use reliable and high performer communication means, which aren't available in standard operating systems.
- An usual problem in this application field is that we need to run some tasks with a guaranteed time period, because of precision of controlling actions in application algorithms. This condition doesn't perform even some real-time operating systems.
- In some special cases we need to ensure, that a process works in a protected memory space and no other process can rewrite its data. In other cases we need a shared memory as an system object for interprocess communication or multiprocessor communication in distributed systems.
- Proprietary operating systems are too expensive for a serial production.

Following requirements for real-time operating systems result from the above stated problems.

4.3. Requirements

- The real-time operating system should include a set of objects for an effective interproces communications to facilitate a dividing task into independent processes and their processing according to their priority. The system should be able to work with process switching period at least 1 ms.
- The OCERA components should be ported to a Motorola family processor, which are supposed most reliable in the process control application field. Using a FLASH and RAM filesystems because of hardware reliability is necessary too.

  During whole OCERA project is necessary to keep rules of a system quality management for software development to ensure sufficient software reliability. Exception handling and system redundancy as fault-tolerant means are required too.

- The development environment for OCERA components should include Gcc compilers and debugers for cross and native environment. OCERA components should keep POSIX API standard to facilitate a portability of applications.
- OCERA components should include real-time ethernet, TCP/IP, UDP/IP, CanOpen protocol and communication means for RS232 and RS485 interface.
- A scheduling algorithm with fixed priorities of threads is necessary to guarantee fixed period of some tasks. A scheduling algorithm with dynamicaly counting priorities is possible for processes.
- OCERA components should provide a possibility to work in a protected memory space for processes including cases of a dynamicaly allocated memory.

  Shared memory is required for interprocess and multiprocessor communication.

- The real-time operating system should be free.

Chapter 5. Multimedia Requirements

5.1. Introduction

We will try to discuss the real-time requirements for the development of multimedia applications. Why should I use real-time capabilities to develop multimedia applications if I normally use multimedia applications with my desktop OS?

Audio and video have some time constraints that should be managed. It depends on the nature of your applications whether you can permit yourself to respect those time constraints or not. Usually in current desktop OS you will miss those deadlines, the more loaded the machine the more deadlines you will miss.

The use of computers for multimedia purposes (professional or particular use) has been increasing in the last years thanks to the availability of more and more powerful machines. Current multimedia applications range from simple players, editing applications, A/V recorders through complex object recognition applications for digital video or speech recognition. Platforms also range from powerful SMP machines through PDAs and other embedded systems.

Applications that require live digital audio and video are unique concerning their real-time throughput and latency requirements.

One important question arise when considering multimedia applications from the RT and/or embedded point of view, though there are multimedia systems specially dedicated, most of the applications run in a dynamic enmvironment where resources have to be shared with a considerable range of applications varying, thus, the load of the machine.

Existing operating systems and network communications protocols are usually inadequate for multimedia systems such as video browsing of high-bandwith conferencing. This is due to the real-time processing requirements of digital audio and video, and in particular to the rigid throughput and latency requirements. Being normally not critical systems, we are used to see multimedia applications in current Desktop OS that usually have a lot of jitter, latencies and unpredictability. When high-quality multimedia is demanded or critical applications are integrated into multimedia systems, the use of Real-Time capabilities is obliged.

5.2. Problems

Most of the widely used multimedia applications are not critical. Missed deadlines in A/V applications do not led to catastrophic consequences. On average, digital video processing is not as much time critical as digital audio processing.

Some of the problems faced when working with multimedia systems are:

- Audio and Video jitter
- High Bandwidth

  Multimedia contents are often characterized by having a high bandwith. Capturing several A/V channels, for example, results in a high bandwidth being bursted through PCI or transmitted accros an ethernet network. Aside, from the data compression point of view, there is a trade-off between bandwidth and audio and video quality, usually one have to adapt quality to reduce bandwidth.

  Not having a mechanism to control deadline misses and time constraints forces developers to be conserative and reduce bandwidth as much as possible to be able to process the video signal when the machine is overloaded.

- Latency

Poor lip-synchronization between audio and video. Buffering is the usual mechanism to get rid of overload conditions. But this mechanism increases latency. On professional applications, latency should not exceed 150 ms. When minimal latency is required, deadline misses and time-related problems arise and developers begin thinking about RT capabilities.

- Slow responsiveness

  Problems about slow responsiveness come out when resources have to be shared with other applications or when dealing with high bandwith signals.

- Insatiability

  The more available resources there are, the more resources multimedia applications demand. Thus it is often difficult to know how and when to limit resource consumption

5.3. Requirements

Considering the problems presented in the previous section, it is natural to ask for certain real-time capabilities that permit the development of multimedia application having a deeper control of machine resources

- Interrupt latency and throughput.

  Systems with large interrupt latency do not support well multimedia applications due to the Continuous Media character of multimedia applications. Handling a MIDI stream, for instance, results in a high number of interrupts to the kernel. Systems with slow context switch cannot cope with those system demanding streams.

- Provide the user with a predictable control of resource allocation.

  Due to the insatiability of multimedia applications, systems can starve of shared or communications resources. The mechanisms to allocate these resources should allow dynamic change in allocation and some form of predictable availability of the specified resource.

- Mechanisms to adapt, dynamically, to the different workload situations.

  Due to the high bandwidth managed by multimedia systems, adapting to different worload conditions affects in a critical manner the use of resources. Droping an image or skipping a GOP (Group of Pictures) in a MPEG A/V sequence results in tens to hundreds of kilobytes of difference being written to disk, sent through the network, etc... Moreover, the interactive character of multimedia systems contribute to the system overload since, apart form the, already CPU-consuming, data processing of signals, systems usually have to send their content through the network or to the VGA increasing hardware interrupts, PCI bandwidth limitations, etc... for man-machine interaction.

  It would be also desirable that this adaption is done in a smooth manner, avoiding rapid accelerations, decelerations in multimedia output.

- Missed deadlines notification

  Though, usually, missed deadlines do not produce catastrophic results, it is important to have tools to know if a deadline has been missed so the application can choose the apropriate behavior in each case, by skipping the computation, perform just a part of it, etc... Some issues in this respect are deadline control, notification, recovery schemes, etc..

- Deliver expected behavior when not in overload conditions.
  The facts that we have mechanisms to manage resources, and eventually adapt the system to diverse workload conditions should not prevent the system to work as expected when in normal conditions. This can be regarded, in short, as a "new mechanisms should not affect scheduling and kernel performance" policy

- Mode-changes Management
  Mode-changes management results very useful when dealing with multimedia since developers can provide different scenarios for different resources availability states. We can think for example in shutting down a video motion detector when the systems enters in alarm mode, etc...

- Static and Dynamic QoS
  QoS level specified in task creation vs. level automatically changed

- Feedback on actual performance
  Support for some kind of feedback between the system and the application on the actual performance is desirable.

Chapter 6. Robotic Requirements

6.1. Introduction

Robotics systems using real-time software embedded may have several different architectures, depending on system requirements, objectives and constraints. However, whatever the architecture, problems encountered in robotics applications development are often the same ones.

The following sections present the most common problems encountered when developing real-time application for robotic systems and sets a list of requirements that a RTOS should satisfy to avoid or minimize such problems.

6.2. Problems

In this section, a set of common problems occurring in robotic application development is listed and detailed.

6.2.1. Scheduling

Most of the robotic applications are generally composed of two different kinds of tasks that can be either synchronous or asynchronous. Problems and requirements are different for these two kinds.

6.2.1.1. Synchronous tasks

Robotics systems requirements are always expressed in terms of deadlines. For instance: an acquisition phase must start at the latest before a specifiable delay after the beginning of the period; an actuation phase must finish its work at the latest before the end of the period or, if asynchronous work is needed each period, at the latest at a specifiable delay before the end of the period.

In mostly used RTOS for robotic application purposes (VxWorks, RTLinux, RTAI, ...) the only scheduling policy implemented is the Rate Monotonic Algorithm (RMA), in which every task is assigned a fixed priority. However, when application becomes more complex and counts many tasks running simultaneously, it may be difficult to distribute fixed priority to the different tasks in order to perform the correct job. Fixed priority is not well suited in this case because far away from the initial requirement.

One could think that the dynamic approach provided by the Earliest Deadline First algorithm (EDF) and available in many RTOS is the most suitable for robotics needs. However, one must be sure that some critical tasks in the system will always have a higher priority than others, independently of their deadline. So pure EDF algorithm is not as well suited as appearing.

6.2.1.2. Asynchronous tasks

For asynchronous tasks, problems are quite different. Deadline misses are allowed for asynchronous tasks by definition, or sometimes, such tasks have no deadline at all.

Asynchronous tasks generally have to perform either basic background jobs or event-driven non time-critical jobs. In both cases, it may be difficult to guaranty that CPU time given to each asynchronous task is optimal regarding the overall application CPU usage; e.g.: guaranty that a minimal average CPU time is given to the asynchronous tasks set and that each of the concurrent tasks of this set has fair access to the CPU resource, especially when some of them are CPU hungry.

6.2.2. Ressource managment

Access to resources shared by tasks belonging to different real-time classes can cause blockings leading to missed deadlines in the real-time cycle. Only a few RTOS provide efficient mechanisms to ensure integrity of a resource despite potential concurrent access and avoiding, or minimizing, priority inversion.

Moreover, the use of some resource protection mechanisms with some particular scheduling policy can lead to inopportune context switches introducing high overheads and, in worst case, deadlines miss (e.g.: synchronization semaphores and standard RMA scheduling policy under VxWorks).

6.2.3. Memory managment

Paging schemes or other mechanisms for dynamic memory allocation are generally not provided in RTOS because they may create unpredictable delay incompatible with required real-time behaviour. Although, statically reserved memory at system initialisation (which is generally performed) do not provide full flexibility to developers who sometimes must resort to complex strategies which do not improve efficiently memory management. For instance, it may sometimes be necessary to allocate more memory than needed by the application.

One other problem taking place in the memory management section is memory protection. Main requirement of robotic systems is reliability, which is highly disadvantaged when no memory protection is provided at runtime by the RTOS. Despite of that, most of the RTOS do not provide any memory protection mechanism at runtime.

6.2.4. Fault tolerance

As technology make progress, robotic applications and hardware become more and more complex and, as a result, it becomes more and more difficult to certify that a peace of software is 100% fault free. New generation mobile robots, for instance, implement very complex path planning algorithms that run in real time. Such algorithms have so many internal states that it is virtually impossible to exhaustively test any transition from one state to another that may occur. As a result, even after validation, bugs often remain in the source code.

Main requirements for robotic systems are reliability and robustness. To avoid such bugs to produce overall system lock, hardware breakdown or even physical injury, fault tolerance mechanisms are generally implemented in the core of the application. Although, as these mechanisms are implemented at the application level, efficiency of such algorithms is rarely optimal.

Robotic systems may be critical (e.g.: automatized transportation system, cooperative robots manipulated by humans such as master robot arms or exo-skeleton robots) or not. In critical robotic systems, fault tolerance is mandatory mainly to avoid physical injury. In this case, the main requirement is fault detection and confinement of the faulty task so that it cannot interfere with other tasks running in the system and finally the system can safely enter a secure state.

In some other applications, robotic systems have to intervene in hostile environment where man cannot go (e.g.: dismantling robot evolving in nuclear environment, submarine robot evolving at very high deep,...). Such robotic systems are not critical (in the way there is no risk of physical injury) but are often very expensive. In the mentioned applications, one has to be sure that robot will always be able to go back home even if a failure (software or hardware) occur; if not, many money (and time) may be lost. In this case, the main requirement is automated system reconfiguration so that it is able to perform some specific tasks in a degraded mode such as: motion using reduced actuators or sensors set.

Exceptions can be considered as "faults" and so lead to same problems and requirements.

6.2.5. Device drivers

Device drivers and interrupt handling often cause severe headaches to robotic developers who generally have no good training nor background of driver (and more generally low level) programming.

Despite of that, robotic developers always have to face up to new hardware (e.g.: sensors, actuators, communication adapters,...) which is constantly evolving and require up to date drivers to work properly.

When using commercial RTOS such as VxWorks, robotic developers can benefit for common devices from the very large device drivers library provided by WindRiver. But when using a free open source RTOS, device drivers are counted on fingers of one single hand and driver programming cannot be overlooked.

6.2.6. Debugging

Debugging real-time application could be a mess. Only a few RTOS provide adequate tools to make debugging easier to application developers.

6.3. Requirements

For all problems related in the previous section, an exhaustive list of requirements are reported in the following section.

6.3.1. Scheduling

6.3.1.1. Synchronous tasks

As seen in the previous section, standard RMA algorithm is not well suited to robotic applications, as much as pure EDF algorithm that cannot guaranty that critical tasks will always have the CPU resource when needed. Depending on the application, an algorithm could be more suitable than another. So the best feature a RTOS could provide to robotic application developers is a set of different scheduling policy (and even optionally user defined algorithms) that could be selected according to the specific needs of the application.

As for other time-critical applications, fast context switch time is mandatory for robotic applications especially to respond in a minimal time to an exceptional event that could occur in the system such as: obstacle detection, hardware failure detection,...

6.3.1.2. Asynchronous tasks

Regarding the problems described in the previous section, a mechanism providing predictable control of the CPU usage allocation to asynchronous tasks is required. This mechanism should guaranty minimal CPU resource to the asynchronous tasks set and a configurable fair CPU usage for each of the concurrent tasks of this set.

6.3.2. Ressource managment

The RTOS should provide efficient mechanisms to ensure integrity of a resource despite potential concurrent access and avoiding or dramatically minimizing priority inversion. A mechanism to avoid inopportune context switches when accessing or releasing a resource should also be provided.

6.3.3. Memory managment

A feature providing dynamic memory allocation at runtime in synchronous tasks would be of great help to developers. Nevertheless, such feature should not lead to unpredictable delays and should not cause unpredictable software faults.

As discussed in the previous section, the main requirement of robotic application is reliability. Following this requirement, a memory protection mechanism is mandatory. This mechanism should lead to an exception raising (that may be handled by a Fault Tolerance manager) if a task attempts to read or write data out of its reserved memory area.

6.3.4. Fault tolerance

The exhaustive characteristics of fault tolerance requirements are reported in the dedicated Fault Tolerance chapter (3), section 3.2.

6.3.5. Device drivers

Robotics applications involve many new devices such as conversion cards and constantly require development of new drivers. A tool and/or development rules to make new drivers suitable to RTOS would be welcomed. Also, a specific protocol providing an additional layer of abstraction allowing to build generic real-time compatible drivers would be useful.

6.3.6. Debugging

Debugging tools and tracing facilities are mandatory for real-time development as for any software development. As debugging in kernel space is a mess, adequate and/or standardized tools would be of great help.