WP3 - Market Analysis

Deliverable D3.4 - Set of Recommendations

WP3 - Market Analysis: Deliverable D3.4 - Set of Recommendations
by Adrian Matellanes

Table of Contents

1. Introduction **1**

2. Set of Recommendations **2**

2.1. System Recommendations 2

2.1.1. Standalone RT-Linux 2

2.1.2. Hard-Soft Real-Time Communications 2

2.2. Resource Reservation Recommendations 2

2.2.1. Clusters of Threads 3

2.3. Development and Debugging Recommendations 3

2.3.1. Remote Debugging 3

2.3.2. User Space API 3

2.4. Memory Management Recommendations 3

2.4.1. Dynamic Memory 3

2.4.2. Memory Protection 4

Chapter 1. Introduction

This document refers to the work done in the second, and final, period of Workpackage 3, **Market Analysis**.

The second working period of Workpackage 3 had, as main objective, to make a revision of the decisions made in the first period. Those decisions made in the first period were the guidelines for the definition and implementation of the OCERA components.

Now, after some of the OCERA components have been developed, we have made an analysis of the possible needs, features and changes that could improve the architecture, design and range of application of the OCERA components and that have not been considered in the first phase of development in OCERA.

In the next chapter we present the conclusions of this revision. We have to notice that this document provides a set of recomendations and that does not imply any development compromise. Morevover, the work done in this Workpackage has not addressed technical difficulties, resources needed for the development or incompatibilities with already existing software; all these questions must be studied by the responsible of the appropriate workpackage.

Chapter 2. Set of Recommendations

In this chapter we present the set of recommendations that resulted from the analysis of the first phase.

We have structured this chapter in sections corresponding to different areas of interest: System generalities, Resource Reservation recommendations, Development and Debugging Recommendations, Memory Management recommendations, etc...

2.1. System Recommendations

2.1.1. Standalone RT-Linux

It was explained in the Architecture specification deliverable that a minimal Linux kernel was needed whatever system configuration we were developing.

That is to say that, even if our system consisted of a few critical RT-Tasks, we needed a whole Linux kernel.

This is undesirable for several reasons:

- Larger footprint

  When the system is very complex, includes a lot of libraries, services, etc... the extra footprint of a Linux kernel is, usually, negligible. But when we are dealing with very a simple, yet critical, system, composed of several real-time tasks, the extra footprint added by a whole Linux kernel (even if its footprint is taken to a minimum) can signify an important percentage of the whole system footprint.

- Longer Boot Time

  Some embedded systems need a boot time as short as possible. Cellular handsets and other mass market embedded systems require very short boot times that cannot be satisfied by the standard Linux boot sequence.

- Increased System Complexity

  Aside of development difficulties, when designing and developing an embbeded real-time system, it is desirable to minimize complexity as much as possible. This complexity reduction results in thorough control of the system and easier debugging. If the services provided by the Linux kernel are no longer needed, then, this system complexity should be avoided.

It would be, then, very convenient to be able to develop embedded hard real-time systems using the OCERA components without the need for a Linux kernel, that is, a standalone system.

2.1.2. Hard-Soft Real-Time Communications

Within the OCERA framework, it is clear how to develop hard and soft real-time systems.

Even if, there exists methods for communicating the hard and soft real-time spaces, we find there is a need for an improvement in this area.

Providing new mechanisms for better sychronization and IPC between these two spaces would ease the development of these hybrid (hard and soft) real-time systems.

2.2. Resource Reservation Recommendations

2.2.1. Clusters of Threads

As Liesbeth Steffens *et al.* point out in their paper *"Resource Reservation in Real-Time Operating Systems - a joint industrial and academic approach"*, recent developments show that many times resource reservation is needed, not only for a single thread, but also for a cluster of threads.

In fact, this is usually the case, an application is composed of several threads and the developer wants her application to have the specified resources. Having the possibility to assign resources to clusters of threads, the developer do not have to bother about calculating the amount of resources each thread of the application needs for working correctly.

2.3. Development and Debugging Recommendations

2.3.1. Remote Debugging

As far as the development of small, embedded real-time systems concerns, the need for debugging tools is obvious.

One debugging method, present in several RTOS and development environments is the remote debugging through a serial port; that could be a possibility.

Remote debugging allows developers to debug their applications when they are already deployed in the embedded system, requires minimum CPU and is less intrusive than other debugging methods.

The existence of a remote debugging method within OCERA would ease development and invite programmers and designers to get into the OCERA environment.

2.3.2. User Space API

We developing hard real-time systems, the development slows down due to the fact that we develop in kernel space, have just one memory space, no memory protection, etc... A bad pointer usually means a system hangs.

It would accelerate and facilitate the development process to have a safe environment where developers would begin coding and testing. Once the functionality would be implemented they could *move* the code into their real environment, kernel and rt-linux spaces, and compile and test the application there. Such a *safe environment* could be user space, having the same API in user space would permit to develop, debug and test the application saving a lot of time.

2.4. Memory Management Recommendations

Memory is, obviously, one of the most important resources an application manages, being it real-time or not.

As noted by the project reviewers, an apropriate memory management should include Dynamic Memory and Memory Protection.

All considerations that follows, apply only to Hard Real-Time applications, which, according to the OCERA Architecture, are executed in kernel space. Soft Real-Time applications execute under the umbrella of the Linux kernel and, thus, rely on the Linux kernel memory management.

2.4.1. Dynamic Memory

Although most of the hard real-time applications are designed and implemented assuming there is no dynamic memory services present, the complexity of hard real-time applications is growing and having no dynamic memory imposes limitations that have to be solved wasting memory (by allocating statically an amount enough) or reducing functionality,

It is desirable then to have mechanisms (though might be somewhat limited and not 'full-equiped') for dynamic memory management

2.4.2. Memory Protection

Almost the same reasoning applies to memory protection.

Most of the developers are used with this constraint and design and implement their systems accordingly. Anyhow, having memory protection, though does not guarantee system integrity, it does permit the inclusion of fault-tolerance mechanisms that can 'repair' or keep the system in a degraded mode if a memory fault occurs.

Without the presence of memory protection we risk data corruption, system hangs, etc...