# System Design for Embedded Automotive Systems

S.Vergata[1,2], J.Wietzke[1], A.Schütte[1] and P.S.Dowland[2]

[1]Faculty of Computer Science, University of Applied Sciences Darmstadt, Germany
[2]Centre for Security, Communications and Network Research,
University of Plymouth, United Kingdom
{s.vergata, j.wietzke, a.schuette}@fbi.h-da.de, p.dowland@plymouth.ac.uk

## Abstract

These days, a modern car will be a complete office, lounge and chauffeur system, composed from different systems, that incorporate themselves into an on board automotive cluster system. This will behave like one component and offer a customized user interface. To facilitate the system development of such a cluster system, different system interfaces like memory and inter process communication mechanisms need to be provided through all the cooperating systems. Embedded systems are now facing the same problems, which occurred years ago in the server world. One of the major automotive original equipment manufacturers chose Intel's Atom CPU as the target platform for their latest development. This step shows that a modern x86 based architecture could provide the basis for a reliable automotive system. There is now an opportunity to introduce virtual machine (VM) technology combined with new security and reliability methods in the embedded automotive sector.

## Keywords

Embedded systems, Hypervisor architecture, Automotive systems, Scheduler, Security compartments

## 1. Introduction

In the automotive business, the software development is domain oriented. Telephone, speech recognition and speech output, for example, were running on their own processor in the last generation. Only because of cost reasons and higher HW-integration, many domains nowadays get combined on the same processor. We count between 8 and 16 domains, some of them supplied by third party vendors, often directly contracted by the OEM.

It is no surprise, that they follow their own priority scheme and scheduling according to their own history. At the same time ideas like domain based binary delivery, get promoted which represent a domain each, contributors are requested to deliver their code as a binary, ready to run, with APIs specified and followed, but without priority and scheduling coordinated.

Caused by the chosen Round Robin scheduling in the OS for the moment, this approach forces all provided binaries to use one common priority (10), which is not a valid concept for a product, causing a high complexity in integrating all components

together in one single system. A new type of system architecture has to be introduced which separates system critical and resource hungry applications case by case.

A recent found automotive alliance named GENIVI selected the OpenSource project MeeGo (Hoffmann 2010) for their base system opening up the automotive sector for freely developed application, with all different types of system usage and requirements. The kernel 2.6.33 used in MeeGo uses a Completely Fair Scheduler (CFS) designed for an interactive desktop system. This scheduler evenly distributes available calculation time to all available threads (Jones 2009).

## 2. Problem Statement

One of the major operating systems for embedded automotive systems QNX, (Turley 2005), developed a new scheduler called Advanced Partition Scheduler (APS) to face the problem of partitioning CPU and resources in a complex software system for multiple application providers.

The current APS (Danko 2007) is tailored to a use-case perspective, in which high priority threads are collected into one partition, less important threads into another partition and so forth.

### 2.1. Budgeting

In this approach, budget inheritance is mandatory through use cases, which is not followed in all aspects.

There is an upcoming need for domain virtualization, in which each domain has its own sandbox with independent priorities and schemes. On this level, there is no need for a full virtualisation with different operating systems in different partitions. There is also no need for budgeting.

### 2.2. Concurrency of priorities

APS doesn't allow independent, non-coordinated priorities and scheduling schemes in different partitions. A somehow special and at the same time typical example of the concurrency of priorities and budgets in APS can be observed in the telephone environment.

Handsfree telephony collects audio samples and processes them one sample at a time. This processing includes echo compensation and has to run on maximum load (100% CPU usage) and on high priority, since crackling, robot voices and mutes should be avoided.

The typical average CPU usage behaviour for this is 50 %. So a characteristic CPU load looks like:


**Figure 1 Typical CPU usage**

In the same domain a so-called background thread is running which is allowed to use all remaining CPU power on a very low priority. If a mobile is connected the first time, all the SIM-card data will be downloaded to the automotive system. In our case, this thread fills up the idle gaps of this partition with its 50% load. So the averaged load in a given time slice will be at 75 %. The available APS Budget will be empty just before the high priority thread restart working. In a time window of 100 ms, perfect hands free will be achieved for 75 ms, the remaining 25 ms, the audio buffers will be empty, causing signal crackling or mutes.

### 2.3. Scheduling in critical mode

In this domain concepts, budgeting and priorities don't cope with each other. We will always face the combination of high and low priority threads in the same partition/domain, sometimes active, sometimes sleeping. Overspending budgets, like proposed and implemented in APS by declaring critical budgets is not a solution, since a thread in critical mode does not follow Round Robin anymore (Danko 2007).

## 3. Steps to a system design

The above described Problems show the need for a new type of system design in the embedded automotive sector. In the following sections two possible designs will be proposed.

### 3.1. Reduced clock scheduling

One solution could be a scheduling scheme, which distributes the CPU power in a fixed and fine granular scheme, so that it looks like we have e.g. 8 parallel processors, each hosting a partition. By that, they do not run on budgets but on reduced CPU clocks, each according to a given static configuration, predefined during system configuration time.

If a partition has threads neither running nor ready waiting, it can allow the scheduler to skip to the next partition. But this is the only dynamic aspect allowed.

A simple non-invasive implementation to gain experience could be the following: A 'Super'-thread on a high priority level next to kernel priority keeps a list of threads per partition. After booting, it starts to increase all threads of the first domain by a fixed value, so that they come first. For that the overall handled priority queues should be expanded by 64 up to 128, so that the addend will be 64 for raising up a domain to the according run level. The available add-on priority levels higher then 64 shouldn't be set by user threads but only by the 'Super'-thread.

Then the 'Super'-thread blocks on a timer with a sleeping time according to our granularity value (e.g. 1 ms). After waking up again, it sets the priorities of the first partition back to their normal value and increases all threads of the next partition.

The scheme specifying which domain follows which, is stored in a look up table. To ensure the configured timings for each domain, it is necessary to use a distributed pre-calculation based on the configured percentage each domain needs. Such a pre-calculation could be done by using a calculation scheme called Sainte-Laguë (Mueller 2003). This type of calculation is used for "seat distribution" in a government parliament.

A small example:

Let us assume a system specified by the following facts:

- 20 % CPU time reserved for base system
- 80% CPU time available slot time for distribution by Pre-Calc

| Domain | D1 | D2 | D3 | D4 | D5 |
|---|---|---|---|---|---|
| Percentage | 35% | 30% | 20% | 10% | 5% |
| Slots | 28 | 24 | 16 | 8 | 4 |

**Table 1: Slot distribution per Domain**

Table 1 shows that the overall available CPU slot time is distributed by the given percentage (division by 5 domains). D1 is the Domain with the highest configured percentage and with the highest calculated CPU-time-slots (28).

The smallest percentage step allowed should be 5%, so with a granularity of 5 each domain can allocate CPU time at configuration time. Smaller steps are possible but not reliable enough. The maximum number of domains should be 20, so it will be possible to distribute in a fair way and the overall system overhead for every scheduling step is small.

The scheduling algorithm could be similar to a Fair Share (Ferrer 2010) scheduling used in an OpenSource implementation called OpenVZ. This implementation also provides memory isolation, which could be useful for an implementation that can separate single domains and prevent them from interacting illegally with other domains.

By having separated domains with assured timeslots, everyone can run independently and use their assured maximum CPU time without interfering with other domain runtimes. On the overall system performance monitor a total system usage of Tu = (100 – Ts) * Td should be seen.

### 3.2. Virtualization

Virtualization began in the 1960s with IBM CP-40/67 and Cambridge Monitor System (CMS) (Parmelee 1972) as first type of hypervisor. The Patent granted to VMWare Inc. US006397242 (Devine 1998) opened up the x86 architecture for server virtualization. Virtualization on modern server systems does not only provide the possibility to use 100% computing power but also creates small containments easier to develop, maintain and rollout. To use the benefits achieved in x86- server virtualization an important step had to be taken, the switch over to x86 CPU.

One of the major automotive original equipment manufacturers chose Intel's Atom CPU as the target platform for their latest development (Otellini 2009). This step shows that a modern x86 based architecture could provide the basis for a reliable automotive system and in a second step opens up all the possibilities known from the server market to the automotive sector.

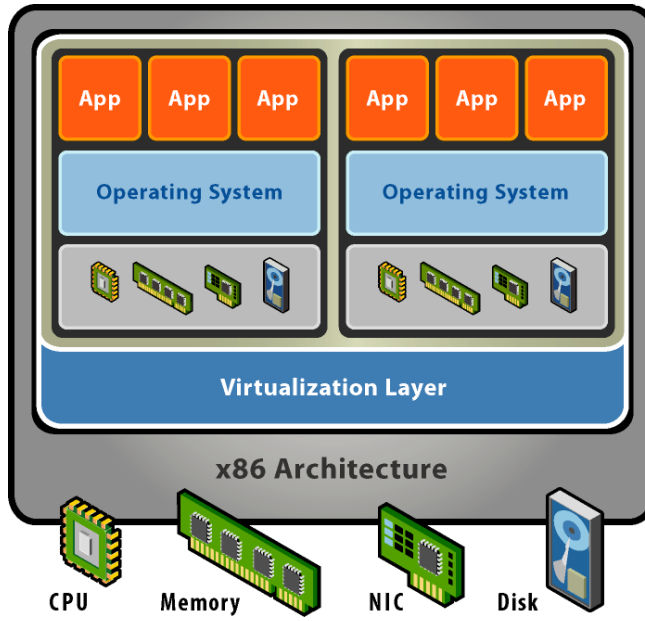


**Figure 2: Virtualization Layer (VMware 2007)**

As shown in Figure 2 the virtualization technique introduces an abstraction layer between the available hardware and the virtualized operating system. The CPU instructions get binary translated, replaced with hypercalls or mapped by the Virtualization Machine Monitor to a physical CPU. The chosen type of virtualization technique is dependent on the hardware. The figure below shows the different levels and the way a command travels through the layers.

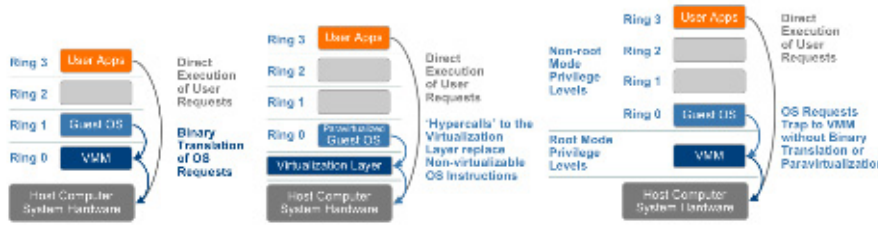


**Figure 3: Different Types of Virtualization (VMware 2007)**

Some work has been done taking virtualisation to the embedded market (VirtualLogix 2006). In the market existing virtualisation solutions separate into two virtualization types:

- System / Full virtualization
- Software / Container virtualization

In a full virtualization each running client has its own operating system and all the needed libraries. This creates a well-known system for the client and provides freedom for the developer. On the other hand it requires more CPU and memory, caused by multiple running OS-Kernels, an overhead on the system stack, and the separation of hardware resources.

To provide the possibility to run under the primary system kernel and by that to reduce the system overhead, a software virtualisation is needed. This has to provide secured execution compartments, memory abstraction and runtime control. Applications and used libraries run in this container and will be provided by the application programmer.

Both types of virtualizations have the possibility to provide a central debug instance to monitor and intercept applications, trace system calls or analyse memory allocation without modification in the running applications. With that centralized monitor, the integrator for the primary system has all methods available needs to check and control the single clients from outside without interfering into the provided system.

First evaluations and tests showed that not only a single virtualisation method is needed, moreover a multilevel virtualisation should be considered

The different types of virtualisation should be combined in one overall virtualisation solution to support all software components that may be required in a future system. The system virtualization requires hardware capable of providing the hardware-based virtualization extension. More and more CPUs introduced into the embedded market like the ARM Cortex A9 MPCore, Hitachi SH7789 or Intel Atom Z5xx are equipped with multiple independent core or virtualization extensions like Intels VT-x or AMD-V, providing the capabilities of a good performing virtualization technologies.

## 4. Conclusion and Outlook

Caused by the increasing amount of applications wanted in the automotive environment many problems will have to be faced in the future. Each application could have a varying way of development strategy, including scheduling, memory usage and much more. In this paper some insufficiencies could be identified, caused by integrating different complex software system together in one automotive system. Our proposals combine applications of different types in a non-interfering way together into one embedded system.

In the next steps of research, both proposals will be integrated into an overall system to provide full virtualization, software virtualization and reduced clock scheduling in secured compartments. With these three different types of systems containment it should be possible to integrate a variety of applications in one embedded system without interfering with each other. A complete system image could be loaded in a full virtualization environment. Reduced software systems without Kernel could be run in a container virtualization and hardware drivers or base applications could be run in a reduced clock scheduling.

The approach described in this paper separates all systems and assures the non-interfering between the systems. The design and realisation of a good performing communication method for the inter-system communications has to be considered in a next research milestone. In future research a new system design for embedded automotive systems will be developed considering security, efficiency and communication of different interacting software components.

## 5. References

Broy, M. (2006), "*Challenges in automotive software engineering",* In: Proceedings of the 28th International Conference on Software Engineering, p. 33–42, ACM.

Chaudhary V., Cha M., Walters J., Guercio S., and Gallo S.. (2008) *"A comparison of virtualization technologies for hpc".* In Advanced Information Networking and Applications,. AINA 2008. 22nd International Conference on, pages 861 –868

Danko A. (2007), "*Adaptive Partitioning - Scheduler",* http://community.qnx.com/sf/wiki/do/viewPage/projects.core_os/wiki/Adaptive_Partitioning_Scheduler (last accessed 06-Apr-2010)

Devine S., Bugnion E., and Rosenblum M. (1998), "*Virtualization system including a virtual machine monitor for a computer with a segmented architecture"*. US Patent #6,397,242.

Elkaduwe D. and Derrin P. and Elphinstone K. (2008), "*Kernel Design for Isolation and Assurance of Physical Memory"*, In: 1st Workshop on Isolation and Integration in Embedded Systems p. 35-40, ACM.

Ferrer R. (2010) *"Fair-share scheduling",* http://wiki.openvz.org/Fair-share_scheduling (last accessed 10-Aug-2010)

Hoffmann J. (2010), *"What does GENIVI's selection of MeeGo mean?"*, http://meego.com/community/blogs/jahoffmann/2010/what-does-genivi's-selection-meego-mean, (last accessed 16-Aug-2010)

Jones M. T. (2009), *"Inside the linux 2.6 completely fair scheduler"*, http://www.ibm.com/developerworks/linux/library/l-completely-fair-scheduler/, (last accessed 20-Dec-2010).

Matthews J. N., Hu W., Hapuarachchi M., Deshane T., Dimatos D., Hamilton G., McCabe M., and Owens J.. "*Quantifying the performance isolation properties of virtualization systems"* In ExpCS '07: Proceedings of the 2007 workshop on Experimental computer science, page 6, New York, NY, USA, 2007. ACM.

Mueller D. C. (2003), "*Public Choice III"*, Cambridge University Press, pages 267-274

OpenICM (2010), Webpage of the OpenICM Framework, h-da - University of Applied Sciences Darmstadt, Germany, http://openicm.fbi.h-da.de (last accessed 20-Aug-2010).

Otellini P. (2009), *"Intel Developer Forum San Francisco Opening Keynote"*, http://download.intel.com/pressroom/kits/events/idffall_2009/pdfs/Otellini_IDF_transcript.pdf , (last accessed 01-Aug-2010)

Parmelee R. P., Peterson T. I., Tillman C. C., and Hatfield. D. J. *"Virtual storage and virtual machine concepts"*. IBM Systems Journal, 11(2):99 –130, 1972.

Pretschner, A., Broy, M., Kruger, IH. and Thomas S. (2007), "*Software engineering for automotive systems: A roadmap",* In: International Conference on Software Engineering 2007, p. 55-71, IEEE.

Seelam S. R., Teller P.J. (2007), "*Virtual I/O scheduler: a scheduler of schedulers for performance virtualization"*, In: Proceedings of the 3rd international conference on Virtual execution environments, p. 105-115, ACM

Sangiovanni-Vincentelli, A. and Di Natale, M. (2007), *"Embedded System Design for Automotive Applications"*, In: Computer, volume 40, issue 10, p. 42–51, IEEE.

Turley J.(2005), "*Embedded systems survey: Operating systems up for grabs",* http://www.eetimes.com/discussion/other/4025539/Embedded-systems-survey-Operating-systems-up-for-grabs (last accessed 18-Aug-2010)

VirtualLogix (2006), "*Meeting the Challenges of Connected Device Design Through Real-Time Virtualization",* http://www.virtuallogix.com

VMware (2007), *"Understanding Full Virtualization, Paravirtualization, and Hardware Assist"*, http://www.vmware.com/resources/techresources/1008 (last accessed 10-Aug-2010)

Wietzke, J. and Tran, MT. (2005), "*Automotive Embedded Systeme"*, Xpert.press, Springer.