OVERSEE - a generic FLOSS communication and application platform for vehicles.

Nicolas McGuire
der.herr@hofr.at
Andreas Platschek
andreas.platschek@opentech.at
Georg Schiesser
georg.schiesser@opentech.at

Opentech EDV Research GmbH
Augasse 21, 2193 Bullendorf - AUSTRIA

Abstract

Following the trend already set by the avionics industry, the target of the OVERSEE project is to create a platform that allows to integrate multiple ECUs into one hardware node, while protecting them from each other. This protection in time and memory ensures the independence of residing applications and their real-time requirements. This approach does not only allow the reuse of (legacy) software modules based on the temporal and spacial isolation, which ensures preserving the dependability, safety and security properties of the individual legacy modules, but also lends it self to modular validation. It potentially also reduces the number of ECUs in the car, leading to a decrease of power consumption, a decrease in weight and allows a higher utilization of the hardware nodes. Reduction of the number of ECUs is not only a cost issue but also a reliability issue. The more systems deployed independently in a car the higher the complexity - the current level of up to 80 ECUs [13] is most likely close to the limit that is economically tolerable.

In addition, OVERSEE will provide a secure interface from the outside world into the vehicle, allowing the connection of the car to the Internet in a secure and safe manner. This connection can be used by the driver to download new applications and data (e.g. maps, software updates,...) into the OVERSEE ECU, providing data communication to the vehicle vendor and/or authorities as well as allow the migration of data from off-car locations while fully preserving privacy, integrity and authentication.

OVERSEE is committed to the usage of open-source technologies and anticipates the creation of an environment that will allow automotive enthusiasts to bring new applications - beyond the creativity of traditional automotive vendors - to the road.

In this article we will outline the goals of OVERSEE, the current state of assessment efforts and the preliminary design of the OVERSEE platform. Though OVERSEE is in an early design stage feedback from the open-source community is essential to ensure that this platform can satisfy the needs of industrial users as well as the open-source community.

It is an expressed goal of the OVERSEE project consortium to provide the necessary community infrastructure to allow creation of an active open-source community in the automotive context and thus not only enhance the capabilities of on-board software but lead to entirely new capabilities at the concept level.

1 Introduction

As the current state of security in automotive systems is less than satisfying [2], the upcoming trend of connecting vehicles to the Internet will make the situation even worse. For this reason, the OVERSEE[1] project is tackling this problem, in order to allow a secure connection of the vehicle to the Internet, while maintaining the automotive system's safety, and therefore the safety of it's passengers. Although OVERSEE's approach does not take care of the problems described in [2], it makes sure, that things do not get worse even with the vehicle's computer systems connected to the outside world.

The second goal of OVERSEE is to research the IMA (Integrated Modular Avionics) approach that is already in wide use in the Avionics Industry[11, 12] for it's use in an automotive system. In comparison to an federated system, where each software component is contained in it's own hardware node, a integrated system allows to house several software components of different criticality on the same hardware node. Of course special precautions to maintain the independence of the software components have to made. The advantage of such an integrated approach lies in the better utilization of the hardware nodes, and - due to reduction of the number of nodes - in savings in power, weight and cost.

There are several standards for operating systems in use in the automotive industry, one of the most popular is the OSEK [3] standard. It's popularity and simplicity are the main reasons, why we at OpenTech think, that it would be a good idea to implement this standard, in order to allow potential users of the OVERSEE platform to run their legacy OSEK compliant applications on top of the OVERSEE platform with no or only minor migration efforts.

Another important step attempted in OVERSEE is to build and establish a FLOSS platform for the automotive industry. So far, the automotive industry traditionally has been building on proprietary OS, in fact it was centered around a per-brand system to a large extent making interoperability of software between vehicle manufacturers almost impossible. Even with the introduction of OSEK and the wide spread endorsement, the situation did only mildly change due to the specifics of bus systems and of course vendor specific algorithms at almost all levels. OVERSEE does not expect that this can be changed easily but at the same time sees a large potential if the automotive applications could gain a common basis to allow non-vendor specific applications to be readily exchanged. With this premises the automotive industry would have the potential to open up to a broader creative community of developers similar to what has happened in the mobile phone market in the past 5 years.

The stakes are higher in the automotive case though as there are not only security related challenges but clearly safety related issues that do not directly arise in the mobile phone market. Learning from the mobile device and consumer electronics though is vital if this process is to be successful - thus OVERSEE has a clear focus on security aspects at this point.

Building a viable community is not a simple task and it is not to be expected to happen quickly - but clearly building on FLOSS technologies, eliminating vendor locking and software dependencies, is essential to achieve a broad automotive vendor acceptance and at the same time a community acceptance. In this sense OVERSEE is anticipating to be an enabling technology - enabling the creative potential of the open-source community to leap on one of the most common computing platforms available world wide - the car.

In the following we give a short introduction to FLOSS implementations of the OSEK specification, present the approach that OVERSEE will take and have a look at the current state of the OVERSEE project. In the last section we will conclude the OVERSEE platform and give a sneak preview on how OVERSEE could develop in our opinion.

Numerous recent publications have indicated that the usage of modern communication technologies in cars pose potential dangers - while not too surprising it is impressive how naively wireless technologies have actually been deployed at this point [14]. OVERSEE is explicitly targeting a secure point of access to automotive environments in a sufficiently generic manner to allow utilizing all mainstream communication technologies.

2 Mapping ARINC653 to OSEK

Since the trend towards an integrated approach is already in use in the avionics industry [11, 12] (IMA - Integrated Modular Avionics), and XtratuM has been developed following the avionics standard ARINC653, we find it necessary to show that the IMA approach, and therefore XtratuM are suitable for the automotive industry. To do so, we made a mapping from ARINC653 to the most commonly standard used in the automotive industry - OSEK/VDX - in order to show that XtratuM (or any ARINC653 compliant OS for that matter) is also suitable for the use in an automotive environment.

This mapping between ARINC653 and OSEK/VDX has been split up into three parts:

Dictionary First a dictionary to map equivalent expressions has been made (i.e. Process / Task).

This part has been done more or less during the other two, while mapping the parts of the standards against each other, you realize that an expression used in one is equivalent to another expression in the second one. This dictionary is meant to give the reader who knows one standard well but does not know the other one a quick start.

Parts directly mappable Next, those parts of the two standards that can be mapped directly have been mapped to each other, using requirements that have been derived from OSEK OS. The assumption here is, that if the requirements derived from one standard (OSEK OS) can be mapped to the second one (ARINC653), these part are compliant. Fortunately, the largest part of the mapping can be done here as a 1:1 mapping.

Parts not directly mappable Last the missing parts (those which are only part of one of the two standards) have been discussed, and it is shown that they are no contradiction to the other standard. For most parts it can even be shown, that though they are not explicitly mentioned in the standard, they are implicitly in use in some part of it (e.g. semaphores are explicitly defined in ARINC653, but are only implicitly in use in OSEK OS). This way a complete mapping between those two standards has been accomplished.

Following to this mapping, a ARINC653 compliant OS following the IMA approach is also suitable for the automotive industry. From a safety perspective this mapping should give us a good basis in case the OVERSEE platform is certified by the authorities in the future. Although this is not planned to be done in the context of the OVERSEE project itself, it is vital to provide this possibility, in order to make the OVERSEE platform more attractive to the OEMs.

3 Available FLOSS implementations of OSEK

During the assessment phase, we identified two FLOSS projects, which implement the OSEK/VDX [3] and seem to be mature enough to be used in the context of OVERSEE, FreeOSEK[4] and Trampoline[5]. There were a couple of others, which seemed a little bit to immature to consider them, therefore we focused on these two.

This section intends to summarize the most important features of these two projects, as well as their (current) shortcomings. Both of these two projects are developed following the MISRA-C coding guidelines, seem to have documented deviations from the MISRA-C guidelines well. Furthermore, both provide a full implementation of OSEK OS, and at least big parts of the rest of the OSEK/VDX.

3.1 FreeOSEK

FreeOSEK[4] is a OSEK implementation started by Mariano Cerdeiro. It currently runs on ARM and on POSIX compliant platforms, so you can test it on your Linux desktop machine.

FreeOSEK is licensed under the GPLv3 with link exception. This means, that you can link your code into FreeOSEK and can still license your code under whatever license you want (free or proprietary).

According to the FreeOSEK homepage, they currently run about 80% of the OSEK conformance tests, and of those about 95% pass. In addition, FreeOSEK is tested, using the static code checking tool splint.

Another nice feature we found in the context of FreeOSEK is GOB[9], a GUI based OSEK configuration builder.

3.2 Trampoline

Trampoline[5] is developed at the Real-Time Systems group of IRCCyN (Jean-Luc Béchennec, Mikaël Briday, Sébastien Faucou and Yvon Trinquet) in Nantes.

In contrast to FreeOSEK, Trampoline supports more hardware platforms, among them ARM, PPC, AVR, c166, cortex-m3, ...

Trampoline is licensed under the LGPL, also allowing you to link your code with whatever license you prefer into Trampoline.

3.3 Assessment Result

Although the community around FreeOSEK is merely not existent, the code base is solid and the most important parts of OSEK/VDX are available, and for those parts implemented, not deviations from the standard could be found.

In contrast, Trampoline has full OSEK support, and they even started to implement parts of AU-

TOSAR. But during the assessment phase, small deviations from the standard could be found. In example additional process states are defined for no obvious reason. This raises the question which additional deviations are in the code which we did not spot.

4 A OSEK Layer for Linux

Although OVERSEE targets to run an OSEK OS on XtratuM, we are going to implement the OSEK layer on a standard Linux environment as well. This gives us three big advantages. First of all we gain a better development environment (i.e. more and better tools for debugging and testing). Furthermore this secondary implementation demonstrates the high portability of the chosen OSEK OS and allows the reuse of FLOSS based components.

The procedure we are going to use in porting an OSEK OS for the OVERSEE platform is, to design and implement an OSEK Layer on top of Linux (in example as a guest OS of lguest and/or kvm). In this environment, implementing, debugging and testing should be a lot more comfortable than in XtratuM.

The port of such an functional layer to XtratuM should be not that hard, since by then we already know, that we have a functional OSEK layer and can focus on the port itself.

4.1 Linux Virtualization Capabilities

Mainline Linux provides a wide variety of virtualization solutions. At the moment all those solutions focus on the use in server applications, and do not target embedded systems at all. Although there have been attempts to use Linux as an hypervisor [8] there is no project - as far as we know - using Linux as a hypervisor in a real-world project.

For the implementation in the context of OVERSEE we are going to assess two candidates, we see as most suitable: KVM [7] and LGUEST[6].

Others like XEN are not seen as very suitable due to their size and complexity. The same might be true for KVM (TODO: find nr. of lines of code) which is huge compared to lguest with roughly 6000 lines of code. Apart from size, a second argument speaking for lguest is it's outstanding documentation - the source code of lguest can be considered as on of the best documented software projects ever.

5 OVERSEE

OVERSEE targets a hypervisor approach, where multiple independent applications, share the same computational platform. As you can see in figure 2, one main aspect of a hypervisor is that it provides a virtualization layer, taking control over the hardware, and controlling the resource usage of the guest operating systems. These resources include CPU time, Memory, Communication Interfaces, ...

Apart from virtualization, the most important features of the hypervisor are a static configuration, known at compile time, the memory protection of the guests, the time partitioning it provides for them and a health monitoring system.

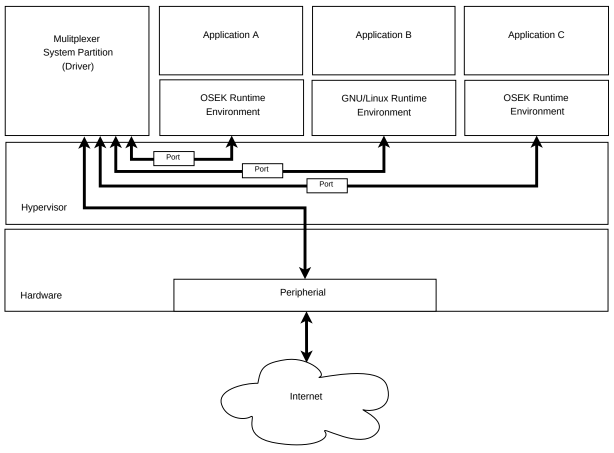

FIGURE 1: *IMA approach - multiplexing secure communication channels*

Static Configuration The configuration of the hypervisor is done via a XML configuration file. This file defines everything from the amount of memory needed by the application and the CPU time it needs to the communication channels it is allowed to write to / read from. The XML configuration file is parsed at compile time and all the configuration data of the system is statically contained in the binary and cannot be changed during runtime. To change the configuration, the XML file has to be edited, the binary has to be rebuilt and flashed into the target.

Static Scheduling The most important property, the hypervisor has to guarantee, is the independence of the applications. We distinguish between different kinds of independence. The first one is the independence in time. This means, that a faulty application must not be able to block the CPU and therefore lead to the starvation of another application.

Memory Protection The second kind of independence is the independence in memory. This means, that an application must not be able to alter the memory space of any other application.

Independent Communication The next kind of independence regards the communication system: a faulty application (e.g. babbling idiot) must not be able to influence another application via the communication channel (e.g. by keeping it from work by sending too many messages).

Health Monitoring System fail - there is no way we can prevent this, but systems rarely drop dead, they actually issue multiple indications of an approaching failure - just that we generally tend to ignore these [15]. Operating systems intended for the use in safety related systems like ARINC 653 or OSEK based solutions, try to utilize this fact by providing appropriate monitoring interfaces, exported to the application domains and thus allowing to detect the deteriorating health of the system before a potentially hazardous situation occurs. In OVERSEE the monitoring infrastructure of the underlying hypervisor modeled along the lines of ARINC 653 is made available to the application and system partitions. This monitoring is also to include security aspects, as in a accessible automotive system like OVERSEE, security breaches could directly impact safety of the system and the environment[14].

6 Composability Issues

The goal of providing platforms for industrial sectors is not new. One of the critical problems for any such platform concept is the composability of systems. Composability has traditionally focused on performance issues, most notably temporal behavior of applications - basically this was the main issue of ARINC 653 and in a lesser part of OSEK.

Security has not been in the focus of automotive industry at the operating system level - simply because cars were seen as "closed" networks and thus there is little potential to attack the system. With the dramatic change introduced by the utilization of wireless communication within cars and to the outside word - obviously this has changed. This implies also that composability demands are changing - OVERSEE addresses the composability demands of automotive industry by providing a core security capabilities in a way that allows to place a non-security related application in a partition (lets say some infotainment application) while placing a highly sensitive application in an adjacent partition (lets say a digital blackbox) and guarantee security properties at the OS level. That is to say the applications are truly independent of each other with respect to security design (in fact the infotainment application is assumed to potentially be malicious). To allow this novel security level composability, OVERSEE has to not only provide core security mechanisms at the OS level but also has to be able to ensure that there are no side-channels in the system - with other words: only explicit interpartition data exchange may be possible. This is enforce by the communication subsystem of the core OS.

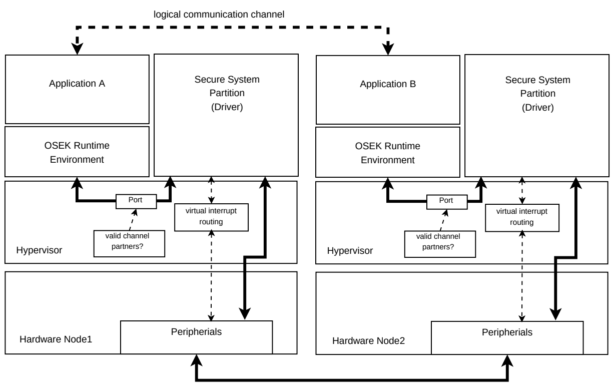

FIGURE 2: *IMA approach - Scalability*

The capabilities that ensure secure composability in OVERSEE are

- spatial partitioning
- partition isolation (temporal and spatial)
- explicit communication (through the core only)
- system level security services (i.e. encryption, TPM, etc.)

7 The state of OVERSEE

Currently (September 2010) the OVERSEE project is in the first third of the design phase, and many vital decisions have not been made yet. So far, we have identified a huge number of possible use-cases[16] ranging from parking sensor systems over eCall to V2V applications. After the consortium has decided on a smaller subset of those use-cases, they were used to derive requirements for the OVERSEE platform, and to be possible candidates for the proof of concept and demonstration phases.

7.1 Next Steps

Based on some identified use-cases and rough requirements identified in the last few month, the design phase has started - this design phase is not a typical bespoke design phase as it is more of an integrative nature - building on pre-existing software components that need to be integrated. This not only raises a lot of interface issues - it is easy to end up with more "glue-logic" than actual code if one is not careful - but also at the same time requires some consolidation of capabilities to fit the specifics of automotive needs.

As with any FLOSS based project, first an extensive assessment phase to identify FLOSS components that can be reused is mandatory, the interfaces between the reused FLOSS building-blocks and the building-blocks to be developed during the OVERSEE project must be designed. This process is highly iterative as it is neither possible to arbitrarily modify FLOSS components without breaking main-line compliance, nor is it reasonable to impose new paradigms on an industry that don't fit current practice. In this sense OVERSEE is a highly integrative project.

In parallel to these activities, the stringent security demands, mandate development of thorough security requirements in the form of a protection profile in the context of the common criteria (IEC 15408). Any such protection profile will not only impact the design but also the development process - this in the context of pre-existing components is a major challenge, from our perspective, for the OVERSEE project.

8 Conclusions

The primary show stoper to enabling community driven automotive apps have been identified as:

- closed APIs and proprietary interfaces
- closed networks as the current paradigm in automotive industry
- and complexity of application integration

OVERSEE targets to resolve these shortcommings by providing a secure access point to in car and internet resources, while providing standadrd APIs.

OVERSEE has the potential to enable car apps to be developed in the opensource community and at the same time ensure security and integrity of automotive platforms by providing critical services as core functionalities of an open platform.

Following the trend already shown by the avionics industry, OVERSEE has the potential to revolutionize the automotive industry and introduce an integrated approach replacing the feterated in-vehicle systems which have already been in use for too long and are by any means behind the state of art.

Acknowledgements

This paper has been produced in the context of the OVERSEE project (FP7-ICT-2009-4, Project ID 248333).

References

[1] *OVERSEE - Open Vehicular Platform*, OVERSEE Homepage, *http://www.oversee-project.com*

[2] *Experimental Security Analysis of a Modern Automobile*, Karl Koscher, Alexei Czeskis et. al, Center for Automotive Embedded Systems Security (CAESS), *http://www.autosec.org/pubs/cars-oakland2010.pdf*

[3] *Open Systems and the Corresponding Interfaces for Automotive Electronics*, OSEK Group, *http://osek-vdx.org*

[4] *FreeOSEK - Scalable RTOS for embedded systems...*, FreeOSEK Homepage, *http://opensek.sourceforge.net/*

[5] *Trampoline - OpenSource RTOS project* , Trampoline Homepage, *http://trampoline.rts-software.org/*

[6] *Lguest: The Simple x86 Hypervisor*, Lguest Homepage, *http://lguest.ozlabs.org/*

[7] *KVM - Kernel based Virtual Machine*, KVM Homepage, *http://www.linux-kvm.org*

[8] *Towards Linux as a Real-Time Hypervisor*, Jan Kiszka, *http://www.linux-kvm.org*

[9] *GOB - The free OSEK configuration builder*, GOB Homepage, *http://gobx.sourceforge.net*

[10] *MISRA-C*, MISRA - The Motor Industry Software Reliability Association, *http://www.misra.org.uk/*

[11] *Perspectives: Reusable Software in Integrated Avionics*, Cary Spitzer, Avionics Magazine, *http://www.aviationtoday.com/av/categories/commercial/Perspectives-Reusable-Software-in-Integrated-Avionics_838.html*, 2005

[12] *Integrated Modular Avionics: Less is More*, James W. Ramsey, *http://www.aviationtoday.com/av/categories/commercial/8420.html*, 2007

[13] *This Car Runs on Code*, Robert N. Charette, *http://spectrum.ieee.org/green-tech/advanced-cars/this-car-runs-on-code/0*, 2009

[14] *Security and Privacy Vulnerabilities of In-Car Wireless Networks: A Tire Pressure Monitoring System Case Study*, Ishtiaq Roufa, Rob Miller, et al. *http://www.usenix.org/events/sec10/tech/full_papers/Rouf.pdf*, 2010

[15] *Normal Accidents: Living with High Risk Technologies*, Charles Perrow, September 1999

[16] *T1.1 Use Case Identification*, OVERSEE, *https://www.oversee-project.com/fileadmin/oversee/deliverables/OVERSEE_D1_1_Use_Case_Identification.pdf*, 2010