MOPED: A Mobile Open Platform for Experimental Design of Cyber-Physical Systems

Jakob Axelsson, Avenir Kobetski, Ze Ni, Shuzhou Zhang, Eilert Johansson
Software and Systems Engineering Laboratory
Swedish Institute of Computer Science (SICS)
Kista, Sweden
E-mail: jakob.axelsson@sics.se

***Abstract*—Due to the increasing importance of cyber-physical and embedded systems in industry, there is a strong demand for engineers with an updated knowledge on contemporary technology and methods in the area. This is a challenge for educators, in particular when it comes to creating hands-on experiences of real systems, due to their complexity and the fact that they are usually proprietary. Therefore, a laboratory environment that is representative of the industrial solutions is needed, with a focus on software and systems engineering issues. This paper describes such an environment, called the Mobile Open Platform for Experimental Design (MOPED). It consists of a model car chassis, equipped with a network of three control units based on standard hardware, and running the automotive software standard AUTOSAR, which consists of operating system, middleware, and application software structures. It is equipped with various sensors and actuators, and is open to extensions both in hardware and software. It also contains elements of future systems, since it allows connectivity to cloud services, development of federated embedded systems, and continuous deployment of new functionality. In this way, the platform provides a very relevant learning environment for cyber-physical systems, today and in the future.**

Keywords—cyber-physical systems; federated embedded systems; automotive; AUTOSAR; software engineering; education.

I. INTRODUCTION

Cyber-physical systems (CPS), i.e., systems containing interacting elements of mechanics, electronics, and software, are of ever-increasing importance in our society. They are part of a large range of industrial products, in domains such as automotive, aerospace, and industrial automation. Due to this importance, it is essential to ensure that companies have access to engineers and researchers with a broad and deep knowledge of the workings of contemporary CPS technology, and education and academic research in the area must ensure that they deal with the challenges industry face today and tomorrow. Those challenges are increasingly on the software side, and on how to deal with complexity as the CPS becomes connected to others and forming federated embedded systems, where the embedded systems in different products connect to each other. CPS engineers do not only need a theoretical understanding of concepts, but due to the applied nature of the field, experience gained from practical work is also essential.

The latter is a challenging situation for educators, since it is difficult to provide students with a laboratory environment which is representative of current industrial systems. Real industrial systems could be an option, but they are proprietary, and thus hard to get unlimited access to. They are also often very expensive, and tend be cumbersome to work with in practice, meaning that students need to spend excessive time on parts that are not key to their learning. Of course, similar problems also face researchers, who want to advance the current state-of-the-art in the area.

The *purpose* of this work is therefore to develop an effective and efficient platform for research and education on contemporary and future industrial CPS, with a focus on software and systems issues. Effectiveness means that it should allow students and researchers to reach their learning objectives to gain practically useful knowledge, and efficiency means that they should be able to do so with a minimal waste of resources, both in terms of time and money.

More specifically, such a platform needs to reach the following *goals*:

- *Representative*: It should reflect key elements of the current state-of-the-art in industry.
- *Accessible*: It should be possible for any researcher or educator to use it, without limitations and within a reasonable budget.
- *Extensible*: It should be possible to easily add new functionality and interfaces.
- *Future oriented*: It should contain elements of technology that can be expected to become common in future CPS, such as connectivity, and continuous deployment.
- *Useable*: It should be possible to operate the systems built on the platform without the need for excessive special equipment, and in a normal office environment.

It should be noted that most real industrial systems fail to meet these goals, except the first one.

Elements that are important for a good learning experience in CPS include a good visual feedback of the physical systems in operation; practical experience of the interaction between software, hardware, and mechanics; and experience of dealing with the complexity of distributed, heterogeneous systems.

The *approach* used to meet these needs is to create and iteratively evolve a model car, which contains a hardware and software environment representative of the complexity and technology found in a real car, but built out of cheap standard components. The platform is called Mobile Open Platform for Experimental Design, or MOPED for short. Using a car as a platform was a deliberate choice, both due to the opportunities for visual feedback, and for the importance of the automotive industry, which is both large in comparison to many other sectors, and also in many respects leading when it comes to new technologies and new ways of working in software and systems engineering. Experiences gained using automotive technology are surely valuable also in other industries.

The rest of the paper is structured as follows. In the next section, some background information on automotive systems is presented, to introduce some of the concepts used in the platform. Then, in Section III, the architecture of MOPED is described, and in Section IV, some experiences of the work are discussed. Section V presents related work, and the last section summarizes the conclusions and gives directions for the future evolution.

II. AUTOMOTIVE SYSTEMS BACKGROUND

In this section, the nature of automotive systems will be described, and some key concepts are introduced that will be used later in the paper. The section will also later be used to substantiate the fact that the MOPED platform is meeting the representativeness criteria described in the introduction, by replicating the essential characteristics of real vehicular systems.

A. Vehicle Electronic Systems

Over the last few decades, automotive embedded systems have expanded rapidly in functionality and complexity. Today, a car typically contains several dozen electronic control units (ECUs) that are connected through communication networks. Each ECU runs control functionality using sensors and actuators, but there are also control functions that are distributed over several ECUs. The most common network technology is CAN, and several such networks usually exist in each vehicle. Also, there are sub-networks connecting devices to an ECU using simpler serial protocols such as LIN. Increasingly, more high-performance technology such as Flexray and even Ethernet are coming into use for the backbone communication infrastructure.

B. AUTOSAR

The increase in complexity is even more visible in the in-vehicle software, and to cope with this, the automotive industry has for the last decade been developing the standard Automotive Open System Architecture (AUTOSAR) [1]. It is, according to the AUTOSAR consortium, already in use in 25 million electronic control units (ECUs) in 2011, a figure expected to rise to 300 million in 2016.

AUTOSAR is structured around a layered software architecture that decouples the basic software (BSW) that needs to exist in all ECUs and can be standardized, from the application software (ASW). In between is a middleware called the runtime environment (RTE). AUTOSAR also provides a component model that eases reuse of parts of the ASW, and allows it to be reallocated if the underlying distributed hardware architecture is changed, thereby improving flexibility and scalability. The BSW consists of:

- Operating system (derived from the OSEK standard);
- System services for, e.g., memory management;
- Communication concepts;
- ECU and microcontroller hardware abstractions; and
- Complex device drivers for direct access to hardware.

The ASW consists of a number of *software components* (SW-C). Each component declares a number of *ports*, which can be either *required* ports (where the component is expecting input) or *provided* ports (that the component uses for its output). The ports can implement different interaction schemes, including sender-receiver or client-server. Ports of different components are connected using configuration tools to form an application. The internal functionality, or the *runnable*, of the component only accesses its ports, and not any other components. The runnables are mapped to OS tasks. SW-Cs can also be composite, i.e., containing other SW-Cs inside.

Between the ASW and BSW is the RTE. It manages all communication between ASW SW-Cs, as well as their access to the lower layers. To make the SW-Cs independent of their physical allocation to different nodes, a concept called the virtual functional bus (VFB) is used, which allows SW-Cs to communicate between each other as if they were all allocated to the same ECU. If they are in fact on different ECUs in a particular implementation, the communication between them has to be mapped to network messages, and this is taken care of by the VFB. The actual implementation of the RTE is done by generating ECU specific software from a description of how the constituent SW-Cs are allocated to ECUs and what links between the components exist. The result is thus a C program that provides an API to the ASW, and that in turn calls the API of the BSW. Apart from communication, the RTE also handles other functionality, such as events, critical sections, etc.

In addition to technical concepts, AUTOSAR also provides a development methodology which heavily relies on different tools for software configuration. Since the intended use is software for resource-constrained embedded systems, the approach is to do all configuring statically at design time instead of dynamically at run-time. This is achieved through a number of description files (using primarily XML format) that are processed by different tools. These description files include, among many other things, information about how the ports of different SW-Cs are connected to each other to form a system, and how SW-Cs are allocated to ECUs and tasks. From the description files, executable software is generated that implement the BSW, RTE, and ASW for a particular ECU.

C. Federated Embedded Systems

Although AUTOSAR provides a lot of flexibility in reconfiguring a system, it is important to notice that it occurs at design time. It does not offer any possibility to make dynamic

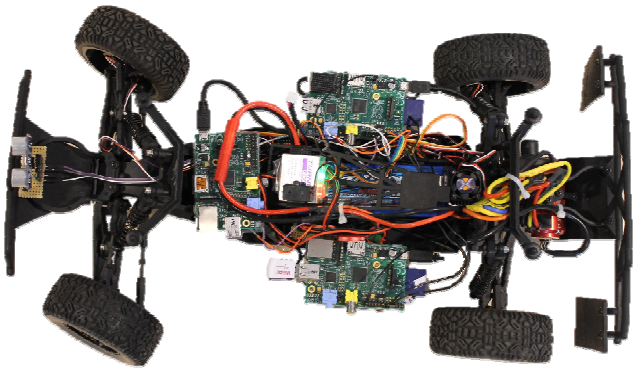

Fig. 1. The MOPED platform.

additions, but any changes require the software to be rebuilt and the ECU to be reprogrammed. In the future, it is likely that the need will arise to dynamically add plug-in software functionality to the embedded systems, in a way similar to how apps can be added to a modern mobile phone.

Such a dynamic model would have several benefits. Firstly, it would drastically decrease the time to market since software can be added or modified very late in the development process, allowing for continuous deployment of new functionality. Secondly, in combination with external wireless communication, it gives the possibility for creating *federated embedded systems* [2], i.e., embedded systems in different products that cooperate with each other. Thirdly, it would create a foundation for open innovation where an ecosystem of third party developers can develop new services that add to the value of the products. The benefits of federated embedded systems, and also the challenges it entails on business and product life-cycle, are further discussed in [3].

III. MOPED Architecture Overview

After having described some key characteristics of automotive CPS, the discussion will now focus on how a platform that is representative for those characteristics can be created. The presentation below is structured into a number of technology areas, and for each of them, the choices made in the MOPED platform are presented, together with a rationale and in some cases also a comparison to other alternatives. A bill of material containing the major components is provided in Table 1, and the physical implementation is shown in Fig. 1.

A. Mechanics

The mechanical platform consists of an off-the-shelf radio controlled car in scale 1/10. It is equipped with a brushless electrical DC motor for driving the rear wheels, and a servo motor for steering the front wheels. The main trade-off here was the choice of scale, and the *rationale* for the scale 1/10 was to have a sufficiently large car to allow easy packaging of electronics, and still a sufficiently small scale to allow driving indoors. A small size also makes it easy to transport the car, which is often needed. The particular car chosen is 53 cm long and has a turning diameter of 1.2 m, which suits those requirements. However, more or less any model car of a similar size and capabilities could be used. In order to use the car indoors, low speed control is more important than high speed, and therefore the original motor had to be replaced with a weaker one rated at 190W, and then the new motor was further recalibrated for slower driving.

B. Electronics

A key decision in the project was which hardware to use for the ECUs, and main factors here was accessibility and extensibility. The hardware should be readily available on the open market at a low cost, but still be fairly powerful to not limit future extensions. Two alternatives emerged in the evaluation: Arduino and Raspberry Pi model B, and the latter was selected. The *rationale* for this was that at the time of the decision, it was more capable than the corresponding Arduino alternative. However, current versions of the Arduino, e.g. Arduino Due, would also be feasible. The decision on ECU hardware is quite fundamental though, since a lot of embedded software is closely related to hardware, in particular OS and device drivers.

The Raspberry Pi has many powerful features, and only some of them are used in this project. It has an ARM11 based processor running at 700 MHz, 512 MB RAM, and contains an SD card for storing software. Peripherals include Ethernet connection, USB ports, 8 general-purpose I/O pins, UART, I^2C bus, and SPI bus.

To make the architecture realistic, it was decided to build a distributed system with three ECUs, connected through a network (Fig. 2). The ECUs are named Vehicle Control Unit (VCU), Sensor Control Unit (SCU), and Telematics Control Unit (TCU), to indicate their principle responsibilities. The *rationale* for using three ECUs was to allow a certain complexity in distributed control functionality, while at the same time keep a reasonable package volume and cost.

TABLE I. Bill of Material for Main Components

Component Type	Actual Component Used
Car chassis	Turnigy SCT 2WD short course truck, scale 1/10 (includes steering servo and original motor)
Motor speed controller	Hobbyking X-Car 45A Brushless Car ESC (sensored/sensorless)
Motor (replacement)	Turnigy TrackStar 17.5T Sensored Brushless Motor 2270KV
ECU Hardware (3 units)	Raspberry Pi, Model B
Ethernet switch	DLINK DES-105
Battery	ZIPPY Flightmax 4000mAh 2S1P 30C hardcase pack
Voltage regulator	TURNIGY 8-15A UBEC for Lipoly
Wheel speed sensor (reflective opto switch; 2 units)	OPTEK OPB715Z
Battery voltage sensor	MCP3008
Distance sensor (ultrasonic)	HC-SR04
Inertial measurement unit	MPU-9150EVB
Wireless connection	Element14 WiPi dongle
Basic software, VCU and SCU	ArcCore AUTOSAR v. 4
Operating system, TCU	Raspbian

The ECUs are connected through a network, and it was decided to use Ethernet for this. The *rationale* was that the Raspberry Pi has integrated Ethernet controllers, using 100Base-TX over RJ45 as a physical layer. Ethernet is also an upcoming standard in the automotive industry for backbone in-vehicle networks, which makes it relevant and future oriented (although the industry is likely to use the physical layer BroadR-Reach instead). The main alternative was to instead use CAN, and this has also been tested, by connecting an external CAN driver over SPI. The main drawback of Ethernet has been the complexity of the device drivers, which required substantial amounts of work. To be able to connect three ECUs, a simple Ethernet switch also has to be included.

Since the model car is intended for indoor use, no particular protection has been used against environmental factors, such as temperature, dust, moisture, etc.

C. *Electrical Power*

The electrical power system consists of a lithium-polymer battery of 4000 mAh, of a standard type used for radio control models, and the *rationale* for this choice was simply that the mechanical platform was adapted for it. The battery has 7.2V nominal voltage, which is fed to the motor and steering servo. For the motor, there is in addition a 45A speed controller. To feed the ECUs, 5V power is required, so a voltage regulator is needed between the battery and the ECUs.

D. *Sensors and Actuators*

The platform is equipped with various sensors and actuators, using the general-purpose I/O pins available on the ECUs. On the VCU, two optical wheel speed sensors are used to determine vehicle speed, measuring both a front and a rear wheel to be able to detect wheel slip. It also has sensors for measuring battery voltage. Actuators include control of the motor and steering servo, and these are based on PWM signals. There is also a set of light-emitting diodes, that can be turned on or off individually, to mimic the different lights of a real car.

The SCU contains more advanced sensor systems, and includes connection to an inertial measurement unit with nine degrees of freedom, which contains accelerometer, gyro, and magnetometer, each in three axes. This sensor is connected using an I^2C protocol. It also contains a forward-looking sensor for measurement of distance to obstacles. Currently, a simple ultrasonic sensor with a range of approximately 4 m is used, but future plans include an investigation of more advanced sensor solution, and most likely also sensor fusion to give an improved picture of the outer world.

The TCU currently contains a WiFi interface through a USB dongle. Previously, a Bluetooth serial connection has also been implemented. There are plans to later include an indoor positioning functionality in this, to provide something similar to GPS navigation in real vehicles.

The *rationale* for selecting this particular set of sensors and actuators has been to include sensors that are representative of functionality in real vehicles, and using similar technology. However, the MOPED platform is also used for experimenting with different sensor solutions, and therefore it is expected that many other interfaces, complementing or replacing the ones mentioned above, will be introduced over time.

E. *ECU Software*

A motivation for the whole platform was to apply software technology used in real embedded systems, and for a vehicular platform, it was a given choice to base ECU software on the AUTOSAR standard. From an educational perspective, it also provides a platform for learning modern software engineering principles based on component-based development for CPS, since this is the foundation for AUTOSAR.

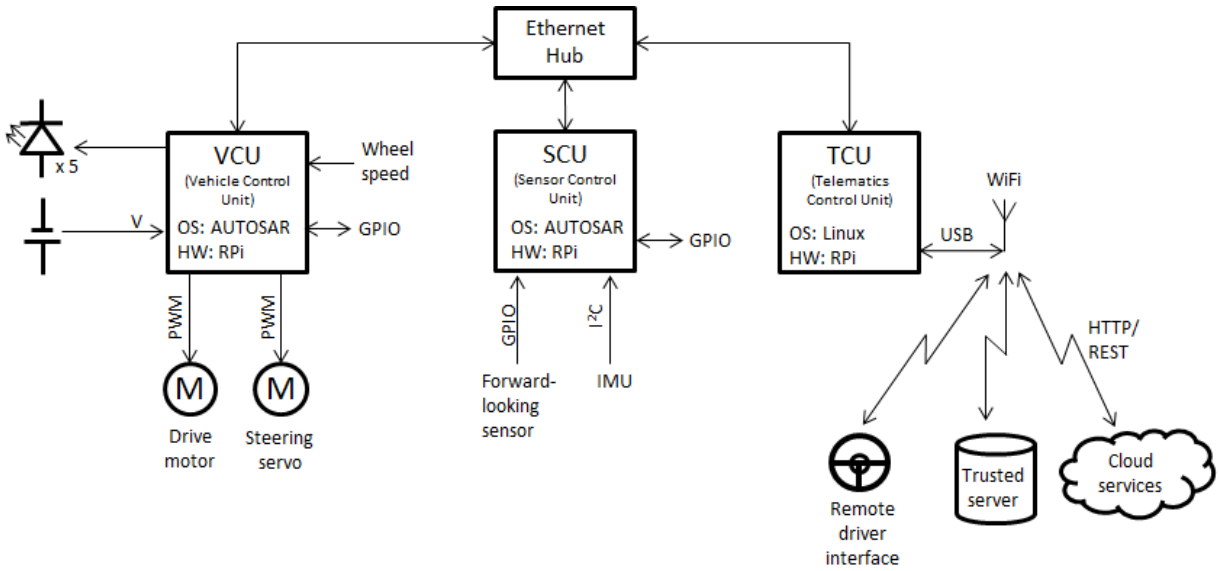

Fig. 2. Physical architecture view, showing ECUs and networks.

The implementation chosen was Arctic Core, and the *rationale* for this is that it is the only open source implementation available with full AUTOSAR support. Other alternatives, such as Trampoline [4], focus on the BSW part only. The implementation used corresponds to version 4 of the AUTOSAR standard, and the project has ported Arctic Core to the Raspberry Pi hardware [5]. The porting involved modifications of OS functionality such as boot loader, initialization, memory model, context switching, and exception handling. It also included writing device drivers for various I/O types, namely the general-purpose I/O, UART, SPI, I^2C, Ethernet, and CAN.

AUTOSAR is used on the VCU and SCU, but the TCU is using Linux. The *rationale* for this is that the TCU is more similar to the infotainment and telematics units used in vehicles, and Linux is increasingly used for that domain. The MOPED platform thus represents the additional complexity present in real systems due to the heterogeneous software architecture. A further benefit of this approach is that device drivers already exist under Linux for complex interfaces such as USB, which reduced the effort of providing wireless connectivity.

The application software consists of a number of SW-Cs, as shown in Fig. 3, including their allocation to ECUs. In principle, there is a sensor or actuator SW-C for each external connection, and individual SW-Cs implementing higher level functionality, such as a simple cruise control, automatic braking based on obstacle detection, basic positioning, etc. In the AUTOSAR based ECUs, the C language is used for writing the software, and on the Linux based TCU, the language is Java.

The built-in software of the ECUs can be reprogrammed either by removing the SD card and copying the new software to it directly, or by letting the boot loader download software over a serial connection.

F. *Driver Interface*

One aspect where a model car differs from a real car is the lack of a driver on-board, and to be able to run the car, a remote interface is needed. In the MOPED platform, a simple protocol has been included that runs on top of WiFi or Bluetooth. An application has been developed that runs under Android, e.g. on a mobile phone, and it consists of two sliders for controlling motor speed (including forward and reverse) and steering direction, respectively. An SW-C is included in the TCU for handling these commands, which are then passed over the network to the SW-C's in the VCU that actually control the motor and steering servo.

G. *Plug-in Software*

A novel part of the MOPED platform, compared to contemporary industrial solutions, is the possibility to download plug-in software into the embedded systems, as mentioned in Section II.C above. The technical solution for this is to embed a virtual machine (VM) into an AUTOSAR SW-C,

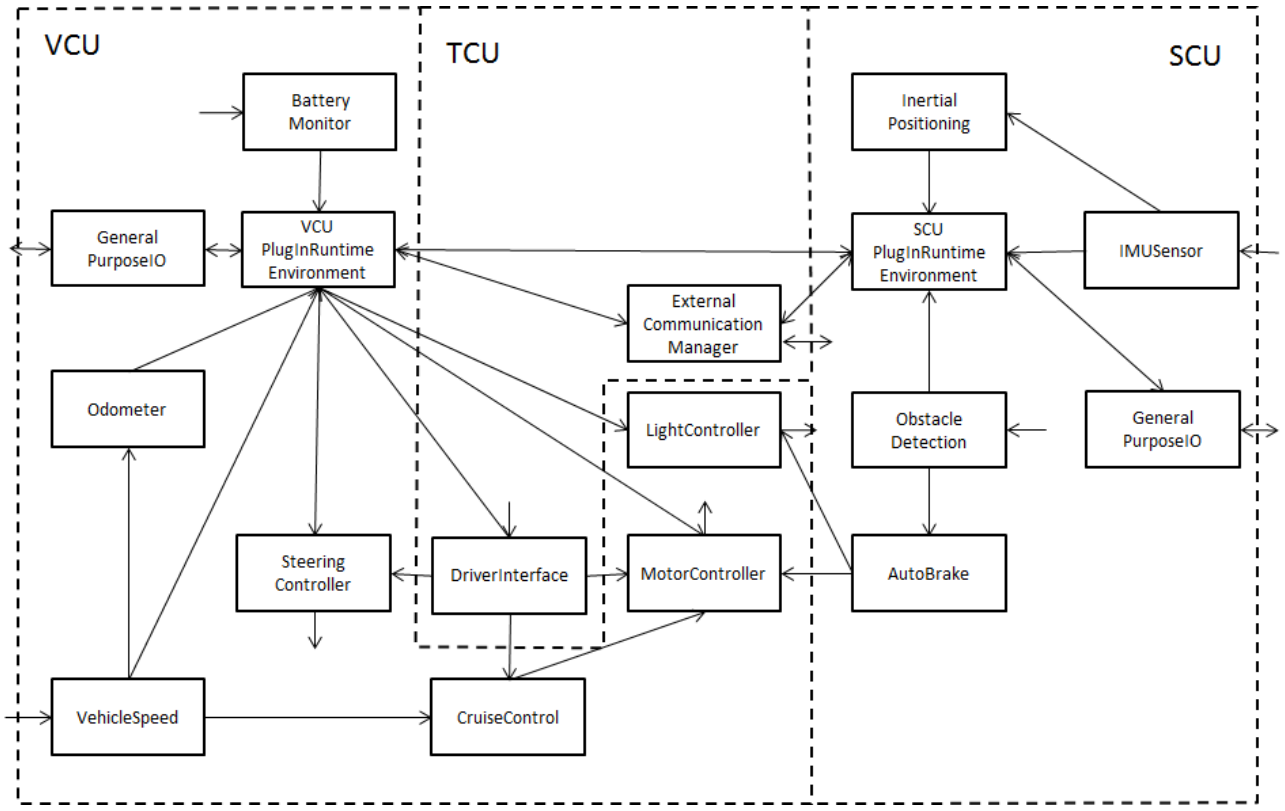

Fig. 3. Logical architecture view, showing AUTOSAR SW-Cs and their allocation to ECUs.

and use that as a sandbox for plug-in execution [2]. In this way, the exact interface between plug-ins and the built-in software can be controlled in detail, as can the resource usage of plug-ins.

It was decided early to use a Java VM, since this is a wide-spread technology, with good development tools available. The next question was then which Java VM to use, and there are many available. After a deep evaluation of alternatives, it was concluded that the Oracle KVM (kilobyte VM) was most suitable [6]. The *rationale* for this was that the KVM is available as source code, which was necessary since some modifications had to be made to integrate it into an AUTOSAR SW-C. It is also comparatively lean in terms of memory usage (50-70 kB), which makes it one of the more realistic alternatives for resource-constraint CPS. A further benefit is its modular and customizable structure, and the clean implementation which makes it highly portable. A potential drawback of this choice is that it uses Java version 1.4, which is somewhat dated, and also the limited support for libraries in its basic version, although additional libraries can be easily included thanks to the modularity of the KVM code. As an alternative, the more recent Squawk VM [7], which is a highly portable Java VM intended for bare metal execution on limited hardware, has also been investigated.

VMs are included in the VCU and SCU, and are given controlled access to the sensors and actuators of those units, as well as to the unused general-purpose I/O pins, to give maximum flexibility when it comes to writing plug-in software. The SW-Cs PlugInRunTimeEnvironment in the VCU and SCU implement the VMs. They differ only in the set of ports they provide and expect, but otherwise the internal functionality is identical.

H. Connectivity

In the concept of Federated Embedded Systems lies also the possibility to connect to systems outside the vehicle. One part of this is to connect to a trusted server, where plug-in software applications are stored. The concept does only allow installation from this predefined server for security reasons. A prototype trusted server has been implemented using Apache and Wordpress. One design decision in relation to this was whether the vehicle or the server should contain the intelligence for configuring plug-ins for a particular environment, and for keeping track of downloading and fault handling. It was decided to push as much of this intelligence onto the server, and the *rationale* was to minimize the load on the embedded system, and also to make the solution more robust when it comes to recovering from failures in the embedded system. The trusted server is thus more than just a file repository, it is a system for life-cycle management and configuration management of plug-ins.

The other reason for adding connectivity is to let plug-in software communicate with other systems, and for this a REST based protocol over HTTP is implemented. In this way, it becomes possible for plug-ins in the vehicle to send status information, such as sensor values, to cloud based servers or plug-ins in other systems, and also to receive commands.

The SW-C ExternalCommunicationManager in the TCM deals with both types of connectivity.

IV. Discussion

After having presented the technical details of the MOPED platform, we will now turn to a discussion of some of the experiences made while developing it.

A. Implementation Status

In the previous section, one reincarnation of the MOPED platform was described, and it reflects the current status of what we have in our lab at the time of writing. The major components have been developed and integrated, and have been subject to various degrees of verification and validation.

It should however be pointed out that we view MOPED more as a concept, than as one specific implementation. Key features of the concept are that it should capture the essential complexity of real software development for CPS, with focus on the automotive domain, and provide a representative learning environment by using similar technology as is used in industry, e.g. AUTOSAR. Things like what mechanical platform, electronics hardware, and sensors are used are less fundamental, and we expect to see many variants of this evolve over time. Therefore, it is never possible to say that the MOPED implementation is complete.

B. Development Tools

To work with the platform, a certain range of tools is needed. This includes basic electronics tools, like soldering equipment, multi-meter, battery charger, and possibly an oscilloscope.

For the software part, there is in principle a need also for an AUTOSAR tool chain, if one wants to modify the structure of the application software in any way. The reason is that the AUTOSAR RTE is generated by tools depending on how SW-C's are connected, and how they are allocated to ECU's, so if any of this needs to change, the RTE must be regenerated for each ECU. This constitutes an obstacle for wider applicability of the platform, since there is no open source or freeware version of the tools available. In this project, an open source version of the basic software from ArcCore was used, and they have also generously provided us with the needed commercial tools, but this cannot be guaranteed for others. The only known effort in developing open source versions of the tools is ARTop project, but it only releases its software to members of the AUTOSAR consortium. Many other commercial tool suits also exist, which could be used.

As for other software development tools, standard compilers, IDEs, etc. have been employed. In addition, for the close-to-hardware development such as device drivers, a JTAG debugger is useful.

To make the development of plug-in software as easy as possible, a simulator has also been developed, that allows early testing of the plug-ins before integrating them into the real car. This includes downloading applications from the trusted server, executing them to see how the car responds, and how it interacts with other plug-in software installed.

C. Difficulties Encountered

In the development of the MOPED platform, several difficulties were encountered. Some were expected, but proved even worse, and others were not foreseen. The porting of the AUTOSAR operating system required a considerable effort, and this was partly due to unfamiliarity with the implementation, and partly to the many details of the hardware that need to be understood. Device drivers proved to be at least as difficult as expected, and here we relied as much as possibly on existing solutions from other projects. However, the more complex drivers, like Ethernet and USB, are very difficult to integrate in a real-time operating system like AUTOSAR. In particular, the Ethernet implementation on the Raspberry Pi contained some surprises, since it actually uses USB to connect between the processor and the Ethernet hardware, and therefore elements of the USB driver had also to be understood to make Ethernet work, which was not anticipated. To some extent, device drivers are an important part of embedded systems, so it is a fine line to draw whether students should be exposed to them or not. We have strived to create an environment where the education and research focus can be more on the higher-level layers of software and systems engineering.

The Java VM also included many practical difficulties. Among the obstacles encountered was the lack of a file system in the underlying AUTOSAR operating system, which is needed to store the downloaded plug-in modules, so a rudimentary file system had to be implemented in RAM. At start up, the plug-in files with Java byte code are transferred from a permanent storage in the TCU under Linux, and then stored during execution in the other ECUs. Also, the interface between the plug-ins implemented in Java, and the AUTOSAR SW-Cs implemented in C posed certain challenges, since KVM does not provide a full Java Native Interface.

V. Related Work

In this section, an overview of related work is presented. It starts with general discussions about the contents of curricula for embedded systems, and then provides a number of examples of other experimental platforms for embedded systems.

In [8], the contents of a graduate curriculum for embedded software and systems is described. It covers many of the basic competencies needed for an engineer in the field, but also emphasizes the aspects covered by the MOPED platform, namely systems issues and architecting. The authors point out the importance of including laboratories on real hardware in the education, as well as providing insight into real industrial technology. However, their concrete suggestion for lab equipment, namely LEGO Mindstorms, does not appear to combine these two aspects as well as MOPED.

Ricks and Jackson [9] also stress the importance of the system level in the curriculum, and point at the lack of a suitable platform for this. They continue to outline such a platform, but are limited to the embedded system, whereas MOPED to a greater extent looks at the whole CPS product. Several other authors present platforms of similar kinds, e.g. in [10], [11] who present heterogeneous platforms, but with a focus on implementation inside the embedded system. [12] is a further example which focuses on hardware/software co-design aspects, and [13] which introduces certain higher-level software concepts such as middleware. All in all, there appears to be a gap between the desired curriculum, and what is actually supported by educational platforms. MOPED can serve to fill this gap with a tool for education on both low-level and system-level issues, integrated using real industrial technology.

The idea to use a model car is not novel in itself, and several other approaches have been presented. In [14], a model car is used as a basis for mechatronics projects, but the focus is on mechanical aspects such as four-wheel steering and driving, and even though the implementation uses some industrial technology, such as CAN, the software does not adopt state-of-the-art standards. In [15], several examples of using model cars as learning platforms for large student team projects are described. However, the purpose is not to build a platform for further use, but the students focus on building a working product from scratch. Although this is certainly a valuable experience in many ways, it requires very large courses, and MOPED differs in that it provides a base platform that can be reused also in courses or research with a more limited focus. In [16], a further example is given of using model cars, but here the focus is on research in cooperative driving, so the platform is not representative of real in-vehicle software.

There are also plenty of examples of educational platforms outside the automotive domain. For instance, [17] discusses the use of the TekBot mobile robot as a laboratory platform for computer engineering students, that can be used in many courses, in a similar way as could the MOPED car. However, there is no ambition to make that robot similar to real industrial systems, or to study system-level issues. A further example of a mobile robot is e-puck [18], but it suffers from similar issues when it comes to representativeness as the previous reference. However, the authors acknowledge the need for a simulator, similarly to what was found in the development of MOPED. Also, the paper discusses a large number of other available robotic platforms. Finally, [19] presents a quadrotor platform, which is possibly the one platform that comes closest to the goals of MOPED, with a focus on software and systems issues. However, despite the fact that it is more spectacular than a car and thus raises much positive attention, the use of a flying platform is in many ways delimiting compared to a vehicle. It requires more space to operate; the operating time is much shorter; the payload is limited leading to lower extensibility; and the risk of damaging equipment through crashes is higher.

In summary, there is strong support in literature for the need of a platform similar to MOPED, and it appears that the approach presented in this paper provides several unique characteristics: it does not limit itself to teaching of low-level embedded systems development, but can also expand into the full realm of CPS, allowing even the study of embedded systems that connect to cloud-based services, and the creation of federated embedded systems. It is also unique in the way that it builds on real industrial standards, making them accessible to students to broaden their experience and strengthen the practical value of their education. The MOPED platform thus provides a much broader spectrum of learning opportunities for students as well as researchers.

VI. Conclusions and Future Work

In this paper, an open platform called MOPED has been presented, that is designed to be highly representative of real automotive systems when it comes to software, while simplifying other aspects, and thus creating a highly accessible and useable learning environment at a small fraction of the cost of a real car. The platform is also extensible, allowing students and researchers to experiment with added functionality and interfaces, and contains elements of future federated embedded systems, such as connectivity to cloud services and continuous deployment of functionality.

When looking just at the functionality provided, MOPED is a very complicated way of building something quite simple, namely a radio controlled model car, but the complexity is in this case something desirable, since it reflects a complex reality that students need to experience in a learning situation. It can also serve as a source of inspiration for future cost-efficient architectures in real vehicles.

The core parts of the platform have been implemented and this will continue to be refined in different ways in the future, including adding more and improved sensor solutions and algorithms. In particular, extending with a high-precision in-door navigation solution would provide information that could be interesting for many experiments. To come even closer to a real vehicle, it would also be good to include more support for diagnostics and calibration, which often constitute large parts of the functionality in a real vehicle ECU.

To further increase the value of open platforms like MOPED, it would be very beneficial to let students and researchers have access to an open source, free version of the AUTOSAR tools needed, at least with some basic capabilities. It would also be interesting to look into alternative ways of providing a VM, other than Java, to see how far the resource requirements can be minimized.

So far, the experiences of using the platform in teaching are limited, so further evaluations are needed, in particular regarding the design of suitable courses and exercises to achieve a maximum benefit in the learning situation.

More information about the MOPED platform, and resources, are available at moped.sics.se.

Acknowledgment

The work presented in this paper is supported by VINNOVA (grant no. 2012-02004), Volvo Cars, and the Volvo Group. ArcCore has generously supported the project with AUTOSAR tools.

References

[1] AUTOSAR consortium. www.autosar.org.

[2] J. Axelsson and A. Kobetski. On the Design of a Dynamic Component Model for Reconfigurable AUTOSAR Systems. In Proc. 5th Workshop on Adaptive and Reconfigurable Embedded Systems, April, 2013.

[3] J. Axelsson, E. Papatheocharous, and J. Andersson. Characteristics of Software Ecosystems for Federated Embedded Systems: A Case Study. J. Information and Software Technology, 2014.

[4] J-L. Béchennec, M. Briday, S. Faucou, and Y. Trinquet. Trampoline – an open source implementation of the OSEK/VDX RTOS specification. In Proc. 11th Intl. Conf. on Emerging Technologies and Factory Automation, 2006.

[5] S. Zhang, A. Kobetski, E. Johansson, J. Axelsson, and H. Wang. Porting an AUTOSAR-Compliant Operating System to a High Performance Embedded Platform. In Proc. 3rd Embedded Operating Systems Workshop, August, 2013.

[6] Sun Microsystems. KVM Porting Guide. March, 2003.

[7] D. Simon, C. Cifuentes, D. Cleal, J. Daniels, and D. White. Java™ on the Bare Metal of Wireless Sensor Devices: the Squawk Java Virtual Machine. In Proc. 2nd Intl. Conf. on Virtual Execution Environments, pp. 78-88. June, 2006.

[8] P. Caspi, *et al.* Guidelines for a Graudate Curriculum on Embedded Software and Systems. ACM Trans. on Embedded Computing Systems, Vol. 4, No. 3, pp. 587-611, Aug. 2005.

[9] K. G. Ricks and D. J. Jackson. Incorporating System-Level Concepts into Undergraduate Embedded Systems Curricula. ACM SIGBED Review, Vol. 6, 2009.

[10] F. Salewski., D. Wilking, and S. Kowalewski. Diverse Hardware Platforms in Embedded Systems Lab Courses: A Way to Teach the Differences. ACM SIGBED Review, Vol. 4, pp. 70-74, 2005.

[11] S. Nooshabadiand and J. Garside. Modernization of Teaching in Embedded Systems Design – an International Collaborative Project. IEEE Transactions on Education, Vol. 49, No. 2, pp. 254-262, 2006.

[12] W. Wolf and J. Madsen. Embedded Systems Education for the Future. Proc. of the IEEE, Vol. 88, No. 1, pp. 23-30, 2000.

[13] J. Sztipanovits, *et al.* Introducing Embedded Software and Systems Education and Advanced Learning Technology in an Engineering Curriculum. ACM Transactions on Embedded Computing Systems, Vol. 4, No. 3, pp. 549-568, Aug. 2005.

[14] R. Grepl, *et al.* Development of 4WS/4WD Experimental Vehicle: Platform for Research and Education in Mechatronics. In Proc. IEEE Intl. Conf. on Mechatronics, pp. 893-898, April, 2011.

[15] M. Törngren, M. Grimheden, and N. Adamsson. Experiences from large embedded systems development projects in education, involving industry and research. ACM SIGBED Review, Vol. 4, pp. 55-63, 2007.

[16] D. Cruz, *et al.* Decentralized Cooperative Control – a Multivehicle Platform for Research in Networked Embedded Systems. IEEE Control Systems Magazine, Vol. 27, No. 3, pp. 58-78, June, 2007.

[17] R. Traylor, D. Heer, and T. S. Fiez. Using an Integrated Platform for Learning to Reinvent Engineering Education. IEEE Trans. on Education, Vol. 46, No. 4, pp. 409-419, Nov. 2003.

[18] F. Mondada, *et al.* The e-puck, a Robot Designed for Education in Engineering. In Proc. 9th Conf. on Autonomous Robot Systems and Competitions, pp. 59-65, 2009.

[19] P. Ulbrich, *et al.* I4Copter: An Adaptable and Modular Quadrotor Platform. Proc. ACM Symposium on Applied Computing, pp. 380-386, 2011.