Impact of Embedded Systems Evolution on RTOS Use and Design

David Andrews[*], Iain Bate[†], Thomas Nolte[‡], Clara M. Otero Pérez[§], Stefan M. Petters[¶]

Abstract

In this paper, we discuss how the evolution of embedded systems has impacted on the design and usage of Real-Time Operating Systems (RTOS). Specifically, we consider issues that result from the integration of complex requirements for embedded systems. Integration has been identified as a complex issue in various fields such as automotive, critical systems (aerospace, nuclear etc) and consumer electronics. In addition, the pressure on time-to-market, the emergence of multi-site development, and the ever-increasing size of software stacks are driving radical changes in the development approaches of modern applications. These complex requirements have placed greater requirements on Operating Systems with respect to how interfaces are defined and how resources are managed. These requirements are expanded and justified through the course of this paper. The requirements are then discussed in the context of emerging solutions from a number of domains.

1 Introduction

The demand for increased levels of functionality and dependability within small, lightweight embedded and real time systems has been steadily increasing for a number of years. While practitioners (academic and industrial) have been attempting to manage and deal with the complexity of modern embedded systems, the open issues are slowly becoming apparent to the average user. There have been numerous examples of projects not being fielded (e.g. Nimrod Early Warning aircraft) or fielded systems with problems arising associated with the interdependencies of complex requirements (e.g. Ariane 501, Mars Pathfinder). In the consumer electronics domain, integration problems have led to longer time-to-market and unresolved issues becoming visible for the end consumer (TV resetting, DVD recorders hanging). A common characteristic of all these examples are that they are the result of emergent properties resulting from integration and that are difficult to identify and verify under sterile laboratory conditions. An example of an emergent property related to real-time is deadlock and priority inversion when blocking on shared resources. This puts stringent requirements on the RTOS to provide better mechanisms for supporting integration in complex architectures and infrastructures using well defined abstractions and interfaces.

In this paper discuss open challenges for run time kernels (Section 2) and implications and limitations on Operating Systems (Section 3) and then discuss proposed solutions in the context of three applications domains: avionics, automotive, and consumer electronics (Section 4). Finally, the paper discusses some potential ways forward (Section 5).

2 Embedded systems development

Although embedded real time (RT) systems platforms and software are tailored for specific application domains such as consumer electronics, automotive and avionics they all share common problems. Problem examples include timing overruns due to effects such as blocking, unexpected time dependent calculations, and difficulties in understanding the implications of changes. These and other issues can be traced to conflicts in functional decomposition of high level requirements into the existing capabilities of desktop operating system semantics adopted for the embedded systems domain.

The existing open problems are a concern as greater and greater demands are being placed on precision and reliability in the growing breadth of application domains within systems that are becoming larger and more complex. Some of the challenging new trends in designing embedded systems are:

[*]David Andrews is at the Information and Telecommunication Technology Center, University of Kansas, Lawrence, KS, USA. email: dandrews@ittc.ku.edu

[†]Iain Bate is in the Real-time Systems Group, Department of Computer Science, University of York, York, YO10 5DD, UK. email: iain.bate@cs.york.ac.uk

[‡]Thomas Nolte is at MRTC, Department of Computer Science and Electronics, Mälardalen University, Västerås, Sweden. email: thomas.nolte@mdh.se

[§]Clara M. Otero Pérez is at the Philips Research Laboratories Eindhoven (PRLE), The Netherlands. email: clara.otero.perez@philips.com

[¶]Stefan M. Petters is at the National ICT Australia Ltd., Sydney, Australia. email: smp@cse.unsw.edu.au

- *Complexity* - Greater levels of functionality together with legacy code and lack of abstractions. The complexity of these issues is derived from:
  - Consumer electronics systems with the convergence of storage requirements, connectivity, and increased integration of functionality (camera, mp3, connectivity for consumer electronics).
  - Automotive systems that are integrating more functionality to decrease cabling and numbers of processors.
  - Avionics sector weight is a major issue. Size and weight issues are driving the movement away from federated systems to integrating functionality on fewer units.
- *Flexibility* - late changes, software download, reuse.
- *Dependability* - the level of integrity required in both failure and non-failure cases have increased. This has been brought about not just due to the fear of losing valuable sales (e.g., Intel adopted more formal approaches after their floating point unit problems on the early version of the Pentium processor) but also because of legislative pressure.
- *Connectivity* - on the systems level we have system integration where there is greater pressure on systems to work together, e.g., mobile phones to communicate with laptop computers etc..
- *Modularity* - needed to help provide maintainability (see below) but also to support concurrent and multi-site development of systems and subsystems. Concurrent and multi-site development is exacerbated as more projects are managed as partnerships and/or using global software development teams.
- *Maintainability* - there is a move away from monolithic development as it makes change difficult and does not support reuse strategies such as Product Line Architectures.
- *Upgradeability* - there is a need to be able to upgrade systems in the field. The upgrades need to be performed by both experts and naive users.
- *Size and power* - there is pressure towards smaller devices that can run over batteries for longer periods of times.

Early embedded systems were mostly uni-application, uni-processor systems point designs developed by teams co-located and targeted for systems with available power. In contrast, embedded systems are being developed to support more than one application domain and must support the upgrages and the addition of new applications in the field. This increases the need for standards and componentization within the solution requiring more abstract interfaces. Initially the need for greater flexibility implied additional functionality within software, e.g., engine control systems were converted from hydro-mechanical systems to computer-based systems. Now, with reconfigurable logic components, additional functionality is being specified at the hardware level.

At the same time there has been a great deal of technology improvements such as the availability of practical Real-Time Operating Systems (RTOS), ’novel’ devices such as Field Programmable Gate Arrays (FPGA) or hyper-threading processors, Systems on Chip (SoC), Network on Chip (NoC), middleware etc.. These trends lead to novel approaches for both hardware and software. These type of solutions support a number of processors, which are often not uniform (e.g., general purpose and signal processing processors). These multiprocessor SoCs are deployed to cope with the market demand for high performance, flexibility, and low cost. NoCs are similarly used. A comparably new trend is the use of asynchronous logic in FPGAs. This is mainly driven to speed up the operation. However, this requires very detailed models on timing behaviour.

To achieve a cost effective solution, expensive resources, such as memory and processor time, are shared among concurrent applications. In the consumer electronics domain, given the dynamic load fluctuations of these applications, worst case resource allocation becomes prohibitive. The allocation of resources below average needs implies that applications have to get by with occasional overloads, reducing system reliability. In the automotive domain, the number of Electronic Control Units (ECUs) is high, driving costs, power usage and integration complexity up, proposing a new era of sharing of ECUs between several subsystems. More powerful, but fewer ECUs allow for automotive subsystems to share an architecture of ECUs. Due to the safety-critical nature of many automotive and avionic applications resource allocation are still based on worst case scenarios. However, integration problems emerge that needs to be treated, i.e., subsystem integration issues.

Component based technology is considered a prime approach to address the problem of time to market and the perceived advantage of reusing code and hardware regarding cost and reliability. The call for increased functional integration on fewer units leads to RT and non RT parts working side by side on the same hardware. This adds complexity in the timely delivery of results, and the security and reliability of operation. The emerging of standards based on collaborations between competitors in the respective area is something, which is now commonly deployed in hardware and software. The standards are used to encourage competition between suppliers, or at least provide means for a second source and hence reduce cost. Instead of traditional top-down development, systems are built bottom-up from a collection of independently developed components and subsystems.

Industrial development has changed to address this complexity. Development happens not any more in a single office but is spread around the world to make effective use of capabilities within a company, multi-site development. This requires different means of development as this obviously has an impact on communication. In order to reuse existing developments legacy hardware and software are deployed. Thus the effort is shifted from the development of new subsystems into the integration and support of legacy subsystems. The use of Commercial Off The Shelf (COTS) components and the outsourcing of well defined components to subcontractors is an attractive means to reduce the in-house development effort. Recently some industries have moved to open source developments. The public scrutiny by enthusiasts is considered a good way of making software reliable.

3 Implications and limitations on Operating Systems

The recent developments on embedded systems introduce new requirements on the infrastructures and consequently on the RTOS. Current RTOS techniques suffer from a number of limitations that have to be addressed.

Developing and testing system components and subsystems is a complex task in itself. However the main challenge appears at integration time, where emergent properties arise as resource sharing causes unpredictable behaviour. A system could potentially consist of a wide diverse of subsystems where the system integrator has varying possibility of control of function, reliability, resource usage, performance and so on. However, there are a number of legal and policy issues. One example is the potential infection of in-house code with public licenses like the GNU Public License (GPL). However the added complexity has meant that the problem of understanding basic components has increased dramatically, never mind the problems of understanding how they might be integrated and the resulting emergent properties.

Scheduling techniques tend to only concentrate on the timing aspects of systems. Although it is acknowledged that in recent years there has been some work on expanding scheduling to deal with other properties such as power. The key problem though is the majority of systems have a large number of properties and objectives to be satisfied. Some of the interactions between properties and objectives can be quite subtle, which means they are often over looked. For instance, in the design of avionics systems there is a link between the variations in when tasks execute and mechanical stress. The reason being is variations in timing lead to errors in data, causing noise and instability on signals, which leads to the moving surfaces of the aircraft (e.g., flaps) being moved more than necessary and hence mechanical stress. To date, little work has been done on truly multidisciplinary design, which has lead to a lack of available analysis techniques. Even if appropriate techniques were available, it is questionable how flexible and scalable the analysis would be for larger, different or more complex systems. The need to support multiple properties suggests that techniques and need to be more aware of the overall system problem and the environment that it is operating in. At the same time there is still a need for the RTOS to have appropriate abstractions from the rest of the system.

Furthermore, in order to cope with maintanance, bug fixes, and system extensions during the life time of an embedded system, these systems provide interfaces for interoperation. These interfaces may be maintanance ports in a car or specific command sequences issued to a sattelite. Furthermore some of these interfaces are an essential part of the functionality of systems, like networking in mobile phones or sattelites. These interfaces can be misused either deliberate maliciously or accidental and thus raise issues in the area of security and possibly safety. One scenario in the mobile phone industry, for example, is a reprogramming of the radio modem of a phone. This could lead a mobile phone to be used as a cell jammer. Paired with a clever written virus to distribute the code, similar to the recent attacks via bluetooth, this can produce serious damage to the mobile phone infrastructure.

Many of the techniques are biased to the worst-case 'hard' real-time systems. However a great deal of systems only have a few hard real-time requirements. Therefore designing the system for the absolute worst case is often un-realistic and results in fragile solutions that are prone to change. A key issue is designing for the worst case wastes valuable resources most of the time, which with current market pressures is not practical. Again, components and techniques need to be designed with QoS in mind whilst not disregarding the importance of selective rigidity. Some kind of Quality of Service (QoS) support might be required.

Finally, there is a lack of first principles/guidelines to build embedded systems and how the functionality is mapped to software tasks or hardware blocks. There is no structural way, no rule of thumb and often, the reasons for certain mapping are not understood.

In the next section, we will consider what some of the key requirements are and which emerging techniques are suited to meeting the requirements.

4 Emerging solutions

The problems can be distilled into the following requirements placed on the way systems are developed and in particular the infrastructures:

- Provide appropriate well-defined abstractions and interfaces.
- Provide design and analysis techniques to account for the complex interactions.
- Support flexible and robust execution. This is from two perspectives; change and failure.
- Partitioning is an essential ingredient to support integrity in systems and fault containment.
- Reduce the trusted computing base, which is the amount of code, which needs to be trusted to keep the system operational.
- Appropriate means for deciding whether components are mapped onto either hardware, software or a mix of the two (i.e., IP cores hosted on FPGAs).

The above is now discussed in the context of three application domains, namely avionics, automotive, and consumer electronics.

4.1 Avionics

Significant work has been performed within the avionics domain to achieve the stated objectives. The main body of work has been performed under the banner Integrated Modular Avionics (IMA) [1, 6, 7]. This work has been driven by the need to support incremental certification and technology transparency. Figure 1, which is based on the civil IMA standard (ARINC 653 [1]), shows the typical structure of an IMA architecture.

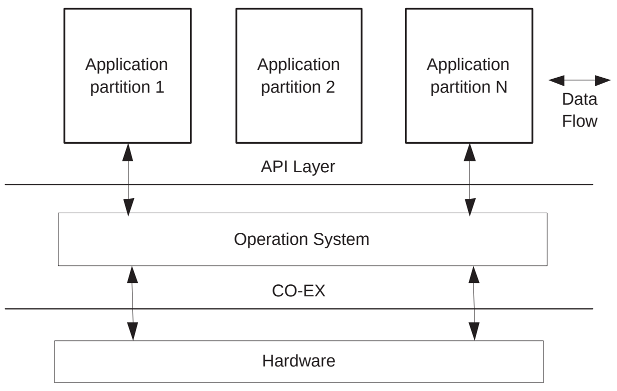

Figure 1. A typical structure of an IMA architecture.

The architecture features two key abstractions / interface layers, which are between the applications and the operating systems (APEX), and then the operating system and the hardware (COEX). Other key components of this architecture is that it represents a move away from federated systems (where a single computing device supports a single application) to modular systems where multiple applications may be supported on a single device. However more than that, the IMA architectures are being developed to support multiple criticality applications on a single device, which means there is a strong requirements for both temporal and spatial partitioning. Thie requirement is resolved through a mix of hardware support and the OS (by checking virtual memory look ups). Other complexities related to the operating system is the use of ”blueprints” that provide location transparency between communicating applications. The blueprints have to provide fast reliable resolution of references and be alterable to support reconfiguration. Other key initiatives related to IMA is the need for modular timing analysis to help support change. One solution proposed to this is the adoption of Reservation-Based Analysis (RBA) [9]. More recently work has commenced on assessing the parts of the hardware infrastructure that can be mapped onto Programmable Logic Devices, e.g., FPGAs. The aim of this work is to reduce chip counts and allow functional to be customised so that it can be made dependable. The IMA OS work represents a good example of work that fits with the requirements that have been identified during this paper. A number of IMA OS are in development and production but there are some key challenges still including making the OS calls more efficient and providing better support for dynamic reconfiguration.

4.2 Automotive

In the automotive domain, the embedded systems are distributed; hence the communications play a key role in the development process all the way from the design, to implementation and integration.

Traditionally, many OEMs have their own standard platforms for developing their embedded computer systems. This is not good from an integration point of view, when several subcontractors are required to adopt platform depending on which OEM that currently is its customer. The solution here is the effort towards standardization of non competitive elements by the initiation of several large consortia in order to agree on a common scalable electric / electronic architecture (e.g., AUTOSAR [2]) and a common scalable communication system (relying on FlexRay [8] together with existing standards such as CAN [10], LIN [11] and MOST [12]).

Looking at communications, automotive systems distribute data over fieldbuses. One way to do this is, e.g., based on specifications of how specific messages are to be used and what data and signals they are to contain (e.g., the J1939 [17] used in the truck and bus applications). This specification is then respected throughout the automotive system lifetime, resulting in a clear but somewhat inflexible networking interface.

Opposite to this early and static specification, the Volcano system [4], currently used by Volvo Car, provides tools for packaging data (signals) into message frames, both for CAN and other networks possibly interconnected with gateways. On top of this specification and signal packaging, it is possible to perform timing analysis of the system from a network point of view, and code can be generated for easy interfacing to data and signals. The Volcano approach allows for a greater degree of flexibility, compared to fixed specification of how data and signals are packed into message.

OSEK/VDX [13], which is a collection of widely used standards for automotive systems, specifies a scalable real-time operating system OSEK/VDX OS [16], communications with transparent communication services OSEK/VDX COM [14], and a network manager OSEK/VDX NM [15] allowing for easy integration of subsystems developed by different OEMs. OSEK/VDX provides reusability and portability of software by using abstract high level interfaces. OSEK/VDX COM allows for communications on a high level abstraction, without detailed knowledge on communication transmitters and recipients locations.

The latest automotive software standard is AUTOSAR, by the AUTOSAR consortia, scheduled to be complete in 2006. The goal of AUTOSAR is to create a global standard for basic software functions such as communications and diagnostics. From an integration point of view, AUTOSAR provides a Run-Time Environment (RTE) routing communications between software components regardless of their locations, both within a node and over networks. Tools allows for easy mapping of software onto the existing architecture of nodes (Electronic Control Units (ECUs)). This mapping is depicted in Figure 2. AUTOSAR is working towards integration of standardized tools relying on, e.g., operating system standards such as, e.g., OSEK/VDX OS, and various communication standards as, e.g., OSEK/VDX COM, FlexRay, CAN, LIN and MOST.

The function integration over the network is a less complex task compared to the integration at the application level. Looking at application level, while designing and specifying the automotive system, model based development is used by some OEMs. Component based development is not used systematically, however, possibly by subcontractors of specific subsystems. Also, the introduction of AUTOSAR will increase the usage of component based software development.

To further increase the flexibility of the development process, some OEMs use a physical model at an early stage for implementation, integration and testing of parts and subsystems. This physical model is used together with modelling tools, such as Statemate and MATLAB/Simulink, to simulate parts and subsystems, environments and specific run-time scenarios. Models of subsystems allows for integration at an early stage in the development process. However, an issue is the exchanging of models between subcontractors and OEMs since these models need to have proper abstractions, not revealing too little or too much information.

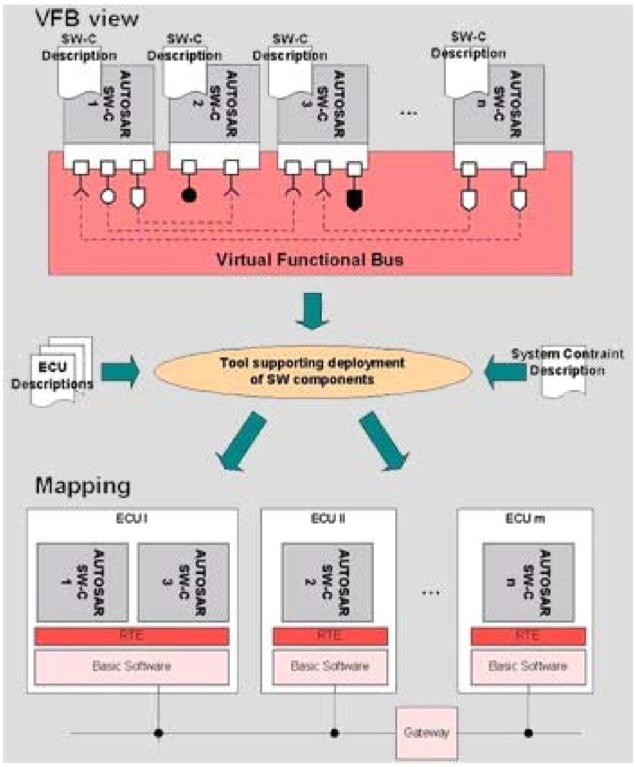

Figure 2. AUTOSAR Virtual Functional Bus and ECU mapping [3].

Furthermore, there are the issue of litigation, if subsystems of different subcontractors are integrated onto a single ECU. In the case of a major, but isolated fault it is important to clearly identify the faulty component. Compartementalisation of applications serves on one hand to isolate faults and on the other hand allows an easier identification of the faulty application.

4.3 Consumer electronics

The consumer electronics companies start to recognize the need for industrial standards to cope with the new trends in software and hardware. Mastering system complexity is not any more the task of a single engineer or a single company. A number of initiatives have been initiated that bring together various CE companies in an attempt to achieve industry-wide standards that benefit all. In this spirit, the Universal Home API (UHAPI) [18] is a hardware independent API that aims at developing and maintaining sustainable CE products. This API favourers the growth of the ecosystem around the products by enabling independent software vendors (ISVs) to create middleware and applications components that easily interact.

Another initiative that directly relates to the OS is the Consumer Electronics Linux Forum (CELF) [5], which advocates for a open source platform for consumer electronics (CE) devices. CELF intends to leverage the benefits of the open source community and process to maximize the re-use of common solutions to common problems.

On the other hand the use of commodity operating systems on embedded devices introduces the problem of millions of lines of code needed to be trusted not to be breakable via denial of service attacks or spreading viruses. This calls for removing any functionality, which does not need to be priviled from the kernel and moved into the user space, supported by proper partitioning.

5 Way forward

Surely, the current challenges facing real time operating systems within these and other embedded applications domains are challenging at best, and will only continue to grow. How should developers and designers of RTOS's proceed to meet the growing challenges? Several issues are clear and must be considered immediately for inclusion in next generation RTOS's.

First, RTOS designers should consider meeting the growing requirements provided in technology growth by exploiting and not fighting Moore's. Hardware/software co-design of RTOS's have historically provided increased scheduling precision. As Moore's law provides a doubling of transistor capabilities every three years, this can be exploited to offer a scaled increase in RTOS performance and capability that cannot be equaled in pure software solutions. With current software based RTOS's, increased functionality requires more lines of sequential code exacerbating already difficult maintenance of critical sections and additional timing overhead in context switching and operating system processing. By migrating portions of the operating system into hardware, Moore's law enables a migration from the temporal to the spatial domain, and enables functionality to increase concurrenty within the transistors.

Second, appropriate abstract interfaces must be formalized to support the rapid seamless insertion of additional hardware and software application components within a system centric framework. This capability is foundational to many of the existing issues, including dealing with increased complexity through higher level abstractions and supporting component reuse to increase times to market for hardware as well as software components. A higher level abstract interface also brings the benefits of abstract type checking into the hardware/software co-design domain, which provides additional dependability, modularity, and maintainability.

Third, security must be elevated to a first class design constraint for RTOS's. Fundamental issues of atomic sequencing between secure states should be considered as both a hardware and software issue in order to eliminate classic time of check to time of access breeches. Current monolithic operating system organizations have also shown the vulnerability of single supervisor mode, unlimited access to global state information. Thus next generation RTOS development should consider built in compartmentalization of operating system functionality, and provide a framework for the development of both soft and hard secure IP within systems that support unsecure components. However, this can not be solved by the RTOS alone. Restricting access to global states inevitably means memory and device access protection. This requires processors to be equipped with memory management units.

Fourth, scheduling should be expanded to include system resource utilization in meeting application timing deadlines. To support expanded schedulability, RTOS's will be required to perform fast non-invasive resource monitoring and scheduling of dynamically time varying reconfigurable resources to meet more complex and interdependent functional requirements.

6 Summary

In this paper we have collected some of the main development trends in embedded systems for the automtive, avionics and consumer electronics domains. Increasing complexity require new approaches to system composition for both hardware and software. In the hardware side, flexibility is enabled by the use of heterogeneaous Systems on Chip, Networks on Chip and FPGAs. For the software, components are beeing developed multi-site and multi vendor. For cost-efficiency reasons, the system resources are being shared introducing unpredictability in the integrated system. To still maintain the traditional "-ilities" some of the limitations on current RTOS have to be addressed. Some of this limitations include

- Lack of first principles
- Interference due to resource sharing not explicitly considered by analisys techniques
- Security

- Lack of QoS support

Finally, emerging solutions for the application domains were discussed.

References

[1] ARINC. *ARINC 653: Avionics Application Software Standard Interface (Draft 15)*. Airlines Electronic Engineering Committee (AEEC), June 17th, 1996.

[2] AUTOSAR. Homepage of Automotive Open System Architecture (AUTOSAR). http://www.autosar.org/.

[3] AUTOSAR Web Content, V22.4. Available 2005-06-06 from: http://www.autosar.org/.

[4] L. Casparsson, A. Rajnak, K. Tindell, and P. Malmberg. Volcano - a revolution in on-board communication. *Volvo Technology Report 98-12-10*, 1998.

[5] Consumer Electronics Linux Forum. http://www.celinuxforum.org/.

[6] R. A. Edwards. ASAAC phase I harmonized concept summary. In *Proceedings ERA Avionics Conference and Exhibition*, London, UK, 1994.

[7] D. Field and J. Kemp. The ASAAC Programme, NATO HQ, Brussels, July 20th, 2000. Available from: http://www.safsec.com/safsec_files/resources/nato_p 1.ppt.

[8] FlexRay Consortium. http://www.flexray.com/.

[9] A. Grigg and N. C. Audsley. Towards the timing analysis of integrated modular avionics systems. *In Proceedings ERA Avionics Conference and Exhibition*, pages 3.1.1–3.1.12, 1997.

[10] ISO 11898. Road Vehicles - Interchange of Digital Information - Controller Area Network (CAN) for High-Speed Communication. *International Standards Organisation (ISO)*, ISO Standard-11898, Nov 1993.

[11] LIN Consortium. LIN - Local Interconnect Network. http://-www.lin-subbus.org/.

[12] MOST Cooperation. MOST - Media Oriented Systems Transport. http://www.mostcooperation.com/.

[13] OSEK/VDX. Open Systems and the Corresponding Interfaces for Automotive Electronics. http://www.osek-vdx.org/.

[14] OSEK/VDX-Communication. Version 3.0.3, July 2004. http://www.osek-vdx.org/mirror/OSEKCOM303.pdf.

[15] OSEK/VDX-Network Management. Version 2.5.3, July 2004. http://www.osek-vdx.org/mirror/nm253.pdf.

[16] OSEK/VDX-Operating System. Version 2.2.2, July 2004. http://www.osek-vdx.org/mirror/os222.pdf.

[17] SAE J1939 Standard. The Society of Automotive Engineers (SAE) Truck and Bus Control and Communications Subcommittee. *SAE J1939 Standards Collection*, 2004.

[18] Universal Home API (UHAPI) Home Page. http://www.uhapi.org/home.