By Neale Foster,
Automotive Market Development Manager,
Wind River Systems

Designing an OSEK Development Environment for Automotive Applications

Today's automotive manufacturers are challenged with replacing traditional electromechanical systems with complex embedded systems. At present, a variety of software environments are used by suppliers, both off the shelf and bespoke, making it very difficult to change CPU suppliers or reuse the software in a new generation of products. The OSEK/VDX standard is emerging as a possible answer, with standardised interfaces and functionality, the OS supports re-usability and portability of application software modules and therefore reduces development cost.
Developing complex OSEK-based systems requires additional OS aware tools to minimize important development time. OS awareness is not a trivial issue and requires tight integration between the development environment of OSEK, the debugging tools, and means of communicating with such tools. One solution is a totally Integrated Development Environment (IDE) incorporating tools and OS from the same vendor delivering OSEK awareness and connections to high level case tools.

Until recently, the development of automotive systems was largely concerned with the design and integration of mechanical and electrical assemblies. In the last few years, however, automotive manufacturers have increasingly faced demands to deliver large amounts of complex embedded software to support the electronic control units replacing the traditional electromechanical systems and point-to-point wiring harnesses.

The task of successfully managing the transition to this increasingly complex software environment has now become a crucial competitive factor for automotive manufacturers because of the stringent quality requirements associated with critical safety systems and the enormous cost of car recall programs. This has led to a situation where many manufacturers have developed their own software design environment, so as to maintain full control over of a key part of their technological investment.

This model of software design creates two costly supplier-dependencies in the production chain. First, the ECU supplier will have invested a serious amount of money in an embedded software product that is very tightly integrated with the CPU architecture, making it very difficult to change CPU suppliers or reuse the software in a new generation of products. Second, the car manufacture will also have invested a lot in application software that is integrated into a proprietary layer from the ECU supplier, making the move to another supplier extremely expensive.

Another factor contributing to the increasing complexity of embedded automotive software can be attributed to the demand for ever more powerful and extensive features from the electronics systems in the car. Not only are individual ECUs becoming more powerful, but the links between them are also becoming more important. For example, the driver display system needs to receive messages from any malfunctioning component within the electronics systems. As another example, it can be desirable to spread the computational load so that a security lock controller CPU can perform another function apart from its very irregular initial use. Furthermore, as cars become complex distributed systems, car manufacturers need to ensure that ECUs from multiple suppliers use the same networking and communication standards. Without a common standard, manufacturers bear the cost of maintaining and enforcing a proprietary communications and networking system.

One possible answer to these problems is to adopt commonly available commercial software and development tools, providing a standardised environment and a simple upgrade path to new processors as they become available. However, given the complexity of the embedded automotive environment, the current generation of off-the-shelf commercial operating systems fall short of meeting user demands. These RTOSs tend to be too large; have insufficient scalability; have inappropriate networking technology and may not available on the required target architectures. Some car manufacturers have defined their specific software requirements and taken delivery of custom operating systems from third party RTOS suppliers. For example, GM PowerTrain has standardised on the WindStream operating system, designed to their specific requirements by Wind River Systems. However, such customised solutions are too narrow in their application to be become the standard for other automotive manufacturers.

Copyright 1999 by Real-Time Magazine 99-3 (http://www.realtime-info.com)

OSEK – AN INDUSTRY STANDARD SOLUTION

The OSEK/VDX standard is aimed at representing a uniform functioning environment which supports efficient utilisation of resources for application software. The Application Programmers Interface (API) of the OS is defined in the specification, and the API is identical for all implementations of the OS on various processor families. Standardised interfaces and functionality of the OS supports re-usability and portability of application software modules and therefore reduces development cost and time.

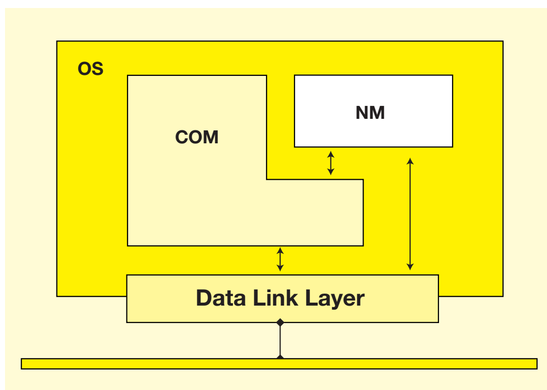

Figure 1. OSEK/VDX Architecture

The OSEK/VDX architecture comprises the following components:

- The operating system (OS): real time executive for ECU software and the basis for the other OSEK/VDX modules
- Communication: data exchange within and between Control Units
- Network Management: configuration determination and monitoring

In order to be a viable standard in the vehicle, the OSEK operating system must be applicable to a wide range of processors – from the simplest 8-bit window lifter ECU to the most complex, high-end 32-bit engine management system. It should be noted that most vendors describe their commercial operating systems as "scalable". This actually means that different components of the OS can be included or excluded, such as networking stacks, but the kernel is always present and unchanged. OSEK takes the scalable concept to a new level in that it allows the actual kernel structure to be optimized for each application.

An example of how the OSEK/VDX OS is scaled is illustrated by the categorization of the scheduling methods into:

- Pre-emptive: a task can be interrupted any time by a task of a higher priority
- Non-pre-emptive: a task can't be interrupted by a task of a higher priority
- Mixed: the scheduling policy can be set for every task

As the operating system is intended for use in any type of control units, it must support time-critical applications on a wide range of hardware. A high degree of modularity and ability for flexible configuration are prerequisites to make the operating system suitable for low-end microprocessors and complex control units alike. These requirements have been fulfilled by definition of "conformance classes" and a certain capability for application specific adaptations.

Conformance classes were created in OSEK to support the following objectives:

- To provide convenient groups of operating system features for easier understanding and discussion of the OSEK operating system.
- To allow partial implementations along pre-defined lines. These partial implementations may be certified as OSEK compliant.
- To create an upgrade path from classes of lesser functionality to classes of higher functionality with no changes to the application using OSEK related features.

The OSEK operating system is built with a tool called a configurator. The user defines the structure of the OS with an intuitive graphical interface and then allows the configurator to build the source code of the OSEK operating system. During the design cycle, the user can choose one of several scheduling policies and select from different "Conformance Classes" offering different levels of functionality. For example, a choice can be made between pre-emptive and non pre-emptive scheduling, balancing kernel size and RAM memory requirements against the real-time constraints of the system. The bottom-line is that a highly optimized

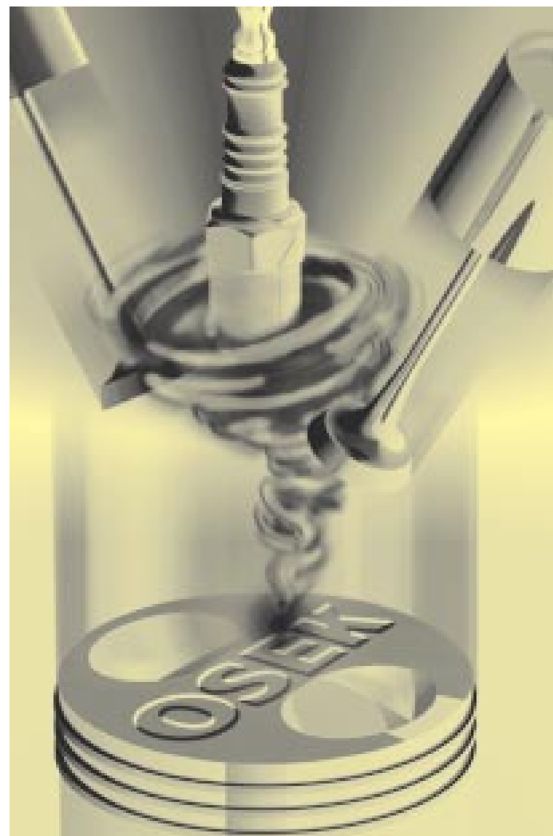

Figure 2.

Copyright 1999 by Real-Time Magazine 99-3 (http://www.realtime-info.com)

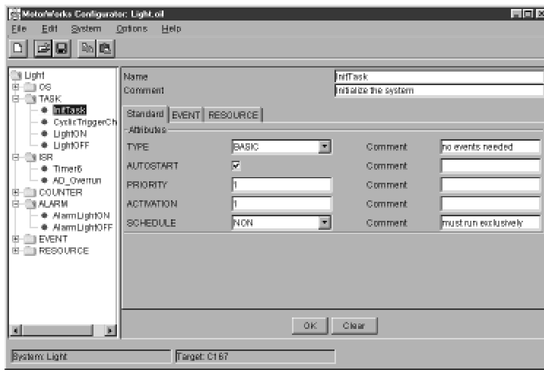

Figure 3. Graphical Configurator

OSEK-compliant operating system can be built for a wide variety of applications with the optimum speed, ROM and RAM footprint. Memory usage and size profiles can be evaluated and optimized with the graphical configurator (see Figure 3).

The OIL file is a key part of an OSEK system design. This describes each instance of an OSEK operating system in an ASCII file, and delivers a standard description of the structure of an RTOS. CASE tools can interrogate the specific OSEK instance and generate appropriate code or even generate the RTOS description and send it as an OIL file to be built in the configurator tool.

DEVELOPING OSEK APPLICATIONS

Most embedded developers are familiar with standard tools such as C compilers, source-level debuggers and emulators. However in order to develop complex, OSEK-based systems, additional tools are required to minimize development time. Without OS aware tools it is extremely hard to examine what is going on with OS-level and examine OS objects. For example if the debugger hits a breakpoint within a function its difficult to work out in which task execution thread this occurred. Another example maybe to check the status of an alarm. Without OSEK aware tools, the user must go back to trawling source code and disassembly displays to check the memory location for clues such as time to alarm expiry.

For more established commercial operating systems such as VxWorks, it is quite common for some kind of OS awareness to be added to third party debuggers. For example Lauterbach provide OS awareness on their in-circuit emulator range. Although adding OS awareness for these companies is not trivial, they are assisted by the OS having a similar architecture for different target processors and the structure of the core kernel remaining constant. OSEK, however, introduces two key problems. The first is that OSEK is a configurable kernel with 24 variations. For example the user can choose to have preemptive or non-preemptive scheduling. This provides the third party debugger company with a kernel that exists in as many as 24 versions with correspondingly different data structures.

Second, OSEK is targeted at the very cost sensitive automotive market, which results in very memory efficient implementations. This means that the data structures of OSEK can be very fragmented so as to use the minimum memory space, and may also be completely different from one CPU architecture to the next. So for a third party debugger, it means significant extra work to port OS awareness from one processor to another. Compounding these factors is that although OSEK is a standard, the specific implementations are proprietary and will differ between vendors.

INTEGRATING DEVELOPMENT TOOLS WITH OSEK

A typical design and development process for an OSEK/VDX-compliant ECU might include the following elements: CASE tool design, application programming (potentially automatic code generation), OS configuration, OIL file generation, debugging and target level development.

An important requirement for OSEK-aware system debugging is to capture the history or trace of task execution or get a dynamic update at the task-level of the system. In order to do this some minimal instrumentation has to be added at the same time as the system is built. Emulators, background debug tools and monitor-based debuggers cannot do this without assistance. Any effective debugging tool must be fully aware of the nature of this instrumentation.

Therefore to get an effective suite of OSEK tools, the development environment of the OSEK system must be very tightly integrated with the debugging tools, and support a means of communicating with such tools. A framework is needed, allowing these disparate elements to work together and communicate, whatever their origin – RTOS vendor, third party tools vendor or the application developer using in-house tools. Wind River has adapted its existing Tornado architecture to support such an open Integrated Development Environment (IDE) targeted on its new OSEK-compliant RTOS, MotorWorks.

INTEGRATION WITH CASE TOOLS

High-level CASE Tools are being increasingly used as natural high-level graphical descriptions to capture concepts and implementation. MathWorks's MATLAB and I-Logix Statemate are heavily used in the automotive industry. Historically the connection from the output of these tools to the embedded software development team was tenuous at best, often not more than a specification. This leads to problems where changes in the software structure made by an embedded programmer are not reflected back onto the original design in the CASE tool.

Companies are looking for ways to solve this by automatic code generation, and by the real-time integration of CASE tools and embedded software development tools. OSEK facilitates this trend as it allows code to be generated for a common OS. For example MATLAB can now generate an OIL compatible file to allow Tornado for Automotive to generate an OSEK OS. OSEK compatible code can then be generated to run on the OSEK OS. Similarly connections are now being made so that the system visualization tools in the CASE environment are connected to lower level tools.

For example in state based tools, states can be single stepped and the corresponding execution at the code

level examined. However, these CASE tools often require small run-time agents to be present on the target to interact. These need to be inserted at build time, requiring a tight integration of the tool that builds the OSEK OS with the tools interface or environment connected to the CASE tool.

The interface between the debugging facilities in the CASE tools and the IDE is particularly important with this approach. The ability of the CASE tools and the IDE to support a common set of primitives such as continue, debug, suspend, single step, set breakpoint, delete breakpoint and get list of breakpoints, is vital. Gopher support is also very useful. This can be used by the CASE tool to request that target data structures of arbitrary complexity be retrieved from the target.

AN INTEGRATED ENVIRONMENT FOR OSEK DEVELOPMENT

A complete development architecture is shown below in Figure 4, and consists of the target ECU running the OSEK OS, and the host PC running the IDE.

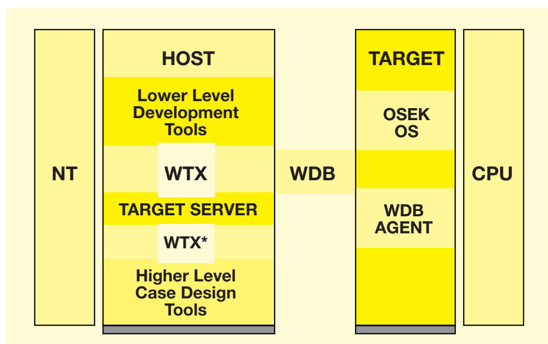

Figure 4. Integrated Environment for OSEK Development

Such an integrated OSEK development environment is already being used in a number of commercial automotive projects across Europe, and a major European collaboration project DRIVE, funded under the European Commission's ESPRIT program, is utilizing Wind River's Tornado for Automotive environment to produce a fully integrated software tool chain for developing automotive ECUs.

REFERENCES

1. The Hansen Report on Automotive Electronics, A Business and Technology Newsletter, Vol 11, No 2, March 1998.
2. Proceedings of the 2nd International Workshop on Open Systems in Automotive Networks, University of Karlsruhe, Germany, October 1997.
3. OSEK/VDX Operating System, Communication, Network Management, OSEK/VDX Implementation Language, http://www-iiit.etec.uni-karlsruhe.de/~osek/.
4. Future Development Trends in the Automotive industry, International Automotive Conference, Congresscentrum A der Messe Stuttgart, July 1998.
5. Tornado API Guide, Wind River Systems Ltd http://www.wrs.com/products/html/trnapift.html.

Neale Foster is manager of the automotive market development for Wind River Systems. His world wide responsibilities include defining market and customer requirements, driving product positioning, managing related third party relationships, and representing Wind River Systems on industry and standards technical committees. Prior to Wind River Systems he worked for a specialist software and engineering consultancy in a number of project management and business development positions. He also has a strong software development and control system background from his years working at Matra-British Aerospace Dynamics. His PhD research involved designing Variable Structure & H-Infinity fly-by-wire control systems in conjunction with the Defence Research Agency, Bedford.

Copyright 1999 by Real-Time Magazine 99-3 (http://www.realtime-info.com)