Impact of Platform Abstractions on the Development Workflow

Johannes Pletzer and Wolfgang Pree

C. Doppler Laboratory Embedded Software Systems, University of Salzburg,
Jakob-Haringer-Str. 2, 5020 Salzburg, Austria.
firstname.lastname@cs.uni-salzburg.at

Abstract. The development workflow for distributed embedded system components in the automotive industry is typically characterized by a strict separation of concerns between the Original Equipment Manufacturer (OEM) and its suppliers. It is based on hardware components or Electronic Control Units (ECUs), where the OEM specifies the network layout and communication system properties before suppliers develop individual ECUs implementing the required functionality. We argue that platform abstractions such as envisioned by AUTOSAR or the Logical Execution Time (LET) abstraction would allow a fundamental overhaul of the development workflow, eventually leading to a significant gain in productivity and flexibility. We analyze the typical workflow and two standard development tools which are commonly used and compare both to the development workflow employed by tools based on the Timing Definition Language (TDL) which represents a LET-based language.

Keywords: Automotive development workflow, Timing Definition Language, Logical Execution Time, OEM - supplier relationship.

1 Introduction

So far, the principal means for structuring the growing amount of software in a car is the splitting of functionality into separate Electronic Control Units (ECUs). An ECU corresponds to a software module. This affects the division of work between an Original Equipment Manufacturer (OEM) and its suppliers and thus the overall development workflow. The OEM specifies all signals sent between the ECUs in the overall electronic system and the complete communication infrastructure which carries them. These signals and the topology information, together with a detailed functional specification, are the basis for the development work of the suppliers, which eventually provide one or multiple ECUs to the OEM who is then responsible for the final integration and testing of the overall system.

This approach requires quite a detailed knowledge of the electronic system from the beginning, as the ECUs depend on the communication parameters and signals and vice-versa. Especially when using the FlexRay protocol [3] there are numerous parameters, such as the division into a so-called static (time-triggered) and dynamic (event-triggered) part, the communication cycle length and static slot size, that need to be agreed on in an early phase of the development process as otherwise the ECUs are not able to communicate. Consequently, changes in a later phase are expensive, as they require adaptations in all ECUs of potentially different suppliers.

The original vision of AUTOSAR [8] was to abstract from platform details to allow developing a software component once and then be able to deploy it automatically on any hardware platform. This would have held the potential to also change the rigid development process. The Giotto language [1] and one of its successors, the Timing Definition Language (TDL) [5] share this vision with AUTOSAR. One consequence of an adequate platform abstraction would be that the communication schedule is not a requirement which suppliers need to obey, but which can be generated automatically as a last step when the OEM integrates all components.

In the following we will outline and compare a) the non-AUTOSAR workflow based on Elekrobit's EB Designer Pro b) an AUTOSAR-workflow based on Vector's DaVinci Tool Suite and c) a TDL workflow based on preeTEC's TDL tools integrated in MATLAB/Simulink [9]. We argue that b) is not sufficient to significantly simplify the development workflow in comparison to a) and that only abstractions such as LET that allow the automatic generation of platform-specific code will do so.

2 Current Workflow and Tools in the Automotive Industry

The tools available for developing automotive distributed systems reflect the workflow which is commonly employed in the industry. Typically one has to specify the communication properties as one of the first steps in development as all further steps depend on it. We take a closer look on two commonly used tools, namely Elekrobit's EB Designer Pro and Vector's DaVinci tool suite.

2.1 EB Designer Pro

EB Designer Pro by Elekrobit [7] is a tool for the design of distributed real-time systems using the FlexRay communication protocol. It aids the user to set up all FlexRay parameters and produces configuration files for FlexRay controllers and the operating system running on the ECUs of the system. Task functions must be provided separately. The tool is available in a full version and also as two separate units, the EB Designer Pro <SYSTEM>, which is limited to OEM design tasks and the EB Designer Pro <ECU>, limited to design tasks performed by ECU suppliers. The developer is guided step-by-step through all required settings to obtain a working system. The steps are divided into a system part and an ECU part which corresponds to the two versions of EB Designer Pro as mentioned above.

Figure 1 outlines the complete development workflow of EB Designer Pro. The first step in the system part is the architecture definition, where the network topology including the number of ECUs and communication controllers in the system and the bandwidth of the FlexRay bus is specified. Next, the detailed settings of the FlexRay protocol must be entered using an optional wizard. The system part is then concluded

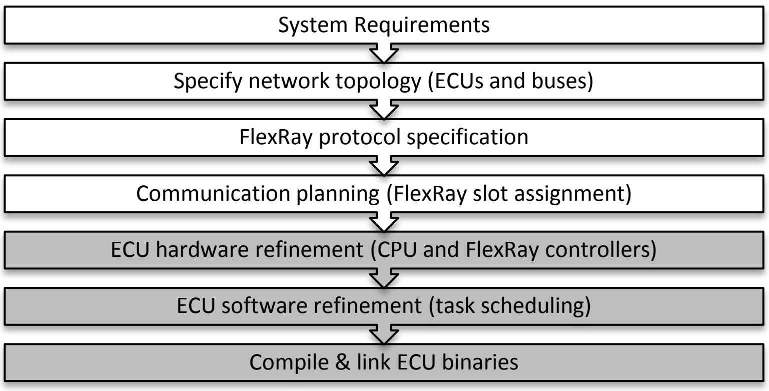
System Requirements
Specify network topology (ECUs and buses)
FlexRay protocol specification
Communication planning (FlexRay slot assignment)
ECU hardware refinement (CPU and FlexRay controllers)
ECU software refinement (task scheduling)
Compile & link ECU binaries

Fig. 1. EB Designer Pro workflow overview (white: OEM, gray: supplier)

with the communication planning, i.e. the assignment of FlexRay communication slots to ECUs in the system.

The next development phase is the ECU part which is typically done by one or more suppliers, who are able to import all the settings the OEM has specified in the system part. The ECU workflow starts with an ECU hardware refinement step, where the type of Microcontroller Units (MCUs) and FlexRay controllers are selected and operating system parameters are specified. Next, the ECU software is refined by defining application and system tasks and assigning them to MCUs. Finally, automatic code generation for every ECU is triggered after the detailed configuration of the communication layer.

2.2 DaVinci Tool Suite

The DaVinci tool suite by Vector Informatik [4] consists of three parts. The System Architect and the Network Designer are typically used by OEMs, whereas the DaVinci Developer is targeted at ECU suppliers. Every tool is used to perform distinct design tasks according to the AUTOSAR methodology. See Figure 2 for an overview of the workflow.

DaVinci System Architect is used to define AUTOSAR software components on an abstract level. This means that no functionality is specified, but only the interface and connections of components, i.e. so-called ports that have a type and a data size. In addition, a network of ECUs is defined and subsequently every software component is mapped to an ECU where it is later executed. After this step, ports can be distinguished by whether the associated software components are mapped to the same ECU and therefore are ECU-local (so-called internal ports) or require network communication as they are located on different ECUs (so-called external ports).

DaVinci Networker Designer is available for different communication buses such as CAN and FlexRay. It is used to set up all properties of the specific protocol, including bandwidth, communication layout, frames and messages. The most important workflow step is the assignment of external ports to messages so that the required values for exchanging data between software components are transferred via the bus.

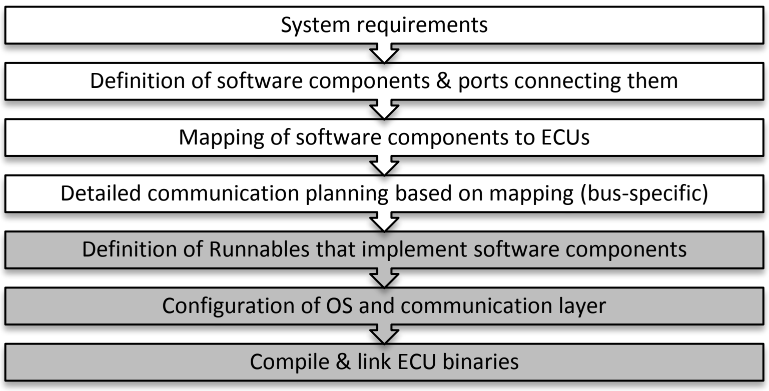
System requirements
Definition of software components & ports connecting them
Mapping of software components to ECUs
Detailed communication planning based on mapping (bus-specific)
Definition of Runnables that implement software components
Configuration of OS and communication layer
Compile & link ECU binaries

Fig. 2. DaVinci Tools workflow overview (white: OEM, gray: supplier)

On basis of the former specification of the system, an ECU supplier can then use DaVinci Developer to create the complete ECU software. So-called Runnables must be defined which are used as a container for user code and finally implement the functionality of software components. Runnables then need to be mapped to operating system tasks where also a priority is assigned to them. Finally, the operating system and the communication layer must be configured before the complete ECU software can be compiled and linked.

2.3 Evaluation

In order to evaluate the flexibility of the workflow of the two tools, let us consider the following example use case: For reasons such as ECU consolidation, a software component of a previously completely specified system needs to be moved from one ECU to another. This typically leads to a change in the communication requirements for the involved ECUs and therefore also to a change required in the communication schedule. For both tools this means that adaptations are required very early in the workflow, and as all subsequent steps depend on it, they all need to be reevaluated and in many cases a redesign is necessary.

When using the EB Designer Pro, it depends on the concrete change that is required to determine to which workflow step one has to go back. If it is sufficient to add or change the contents of individual FlexRay slots, changes in the communication planning workflow step are required. If this is the case, subsequent changes in the ECUs are local to the ECUs involved in the relocation of the software component. If however moving the component requires changes in either the slot size or the communication cycle length, this leads to a change in the FlexRay protocol configuration and thereby invalidates the design of all ECUs in the cluster. In this case all FlexRay controllers must be reconfigured which potentially leads to a change in the timing and therefore the behavior of every single task on every ECU of the system.

Unfortunately, also the AUTOSAR-based DaVinci Tools provide only little support for the described ECU consolidation use case. As the mapping of software components to ECUs is done by the OEM early in the workflow, a change again invalidates all subsequent steps to a certain degree. Most importantly the communication planning step, which is done manually with DaVinci Network Designer, is critical as it later is the basis for ECU development with DaVinci Designer.

The AUTOSAR methodology is meant to promote a less ECU-centric workflow by supporting the reuse of components and the freedom of moving them between ECUs. Indeed, these tasks are simplified by the introduction of the standardized AUTOSAR Basic Software and the introduction of the software component abstraction. However, (a) moving a software component from one ECU to another requires significant manual design and development efforts and (b) it is not guaranteed that the component will behave equally as before. The actual behavior will depend on complex timing issues regarding the layout of the communication schedule, the CPU power of the ECUs, the task priorities of AUTOSAR Runnables and the timing of sensors and actuators, among others. As a consequence, the consolidated system must again be rigorously tested.

3 The TDL Approach and its Impact on the Workflow

The TDL approach is based on the concept of Logical Execution Time (LET), which was introduced in the realm of Giotto [1]. It aims to resolve typical shortcomings of embedded software construction, such as platform dependency and lack of compositionality. These are caused primarily by the fact that timing behavior is not specified explicitly but rather is a result of the system load and the occurrence of unpredictable events at runtime. The LET abstraction offers a solution by abstracting from the physical execution time of tasks and, in the distributed case, from network communication. It does so by specifying that the inputs of a task, which can be values read from sensors or outputs of other tasks, are read at the beginning of the LET period and the outputs provided to other tasks or actuators are only updated at the end of a task's LET. As shown in Figure 3, we call the beginning of the LET the release event and its end the terminate event. As long as physical task execution at runtime and potential network communication take place within the LET of a task, the software will exhibit exactly the same observable behavior on any platform - no matter if it is fast, slow or even distributed. Handling network communication inside the LET leads to the notion of transparent distribution [2], as the fact that a system is distributed does not change its observable behavior, though the physical behavior, in

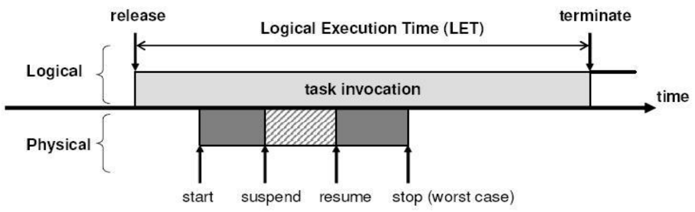

Fig. 3. Logical Execution Time (LET)

particular the order and length of task executions and the time when messages are communicated, may differ.

3.1 TDL Tools

The main TDL tools [10] offered by preeTEC are the TDL:VisualCreator and the TDL:VisualDistributor, where the former is used for platform-independent modeling and the latter for platform mapping.

The TDL:VisualCreator is used to create so-called TDL modules, which are software components that act as a unit of composition and distribution. Modules contain sensor, actuator, task and mode definitions. Only one TDL mode can be active at a time and it contains the timing specification for actuators and the LETs for all tasks running in the specific mode. Functionality code for tasks must be provided separately as source or object code. When using the MATLAB/Simulink integration feature of the TDL:VisualCreator, the functionality code can be generated automatically from the Simulink model by a standard MATLAB tool named Real-Time Workshop Embedded Coder (RTW-EC). The Simulink integration also allows the simulation of the TDL system, which due to the LET abstraction is guaranteed to be equal to the observable behavior on the platform.

The TDL:VisualDistributor is used to deploy TDL modules on a potentially distributed hardware platform. It allows specifying the platform, i.e. the ECUs and communication buses connecting them. Support for new types of ECUs and buses can be added via a plug-in architecture. Note that to support a platform also a corresponding TDL runtime system must be implemented which ensures the proper timing of the system according to the LET semantics. When mapping a TDL module to a concrete ECU, a platform specific Worst Case Execution Time (WCET) must be set for each task of a module running on an this ECU. Furthermore, the sensors and actuators of a TDL module must be assigned either by specifying a C function or via a graphical interface in case the corresponding ECU plug-in supports that. Finally, the complete code for the system can be generated. This also triggers the fully automatic bus schedule generator which determines the communication requirements of TDL

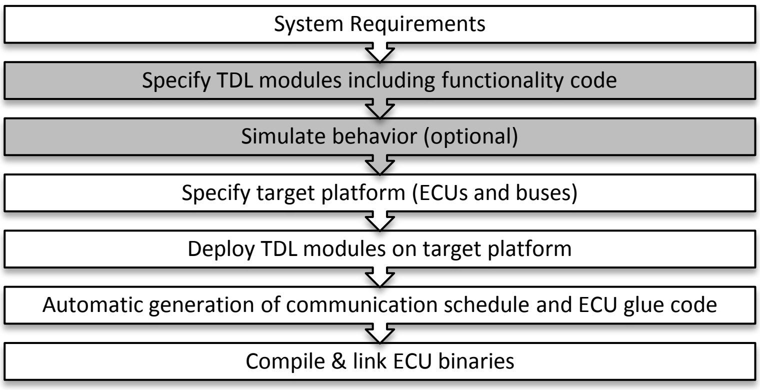
System Requirements
Specify TDL modules including functionality code
Simulate behavior (optional)
Specify target platform (ECUs and buses)
Deploy TDL modules on target platform
Automatic generation of communication schedule and ECU glue code
Compile & link ECU binaries

Fig. 4. TDL tools workflow overview (white: OEM, gray: supplier)

modules by their deployment to ECUs.

Regarding the automotive workflow, the TDL tools can be used as shown in the workflow overview in Figure 4. Suppliers may use the TDL:VisualCreator to model software components according to requirements provided by the OEM. The OEM then uses the TDL:VisualDistributor to map the TDL modules to the target platform.

3.2 Evaluation

Considering the ECU consolidation use case as described in 2.3 above, it can be performed with much less effort. As no TDL modules need to be changed in such a case, only the mapping of modules to the hardware platform must be adapted in the TDL:VisualDistributor. This is done by assigning the module to another ECU and setting the sensor, actuator and WCET properties accordingly. After that, the code of the whole system–including the network schedule–is simply regenerated. Note that if the schedulability check passes and code is generated the observable behavior is exactly the same as before ECU consolidation, without requiring additional testing.

4 TDL Workflow Advantages

The TDL workflow offers a new level of flexibility and productivity for OEMs and suppliers that range from testing to the optimization of hardware platforms.

In contrast to conventional tools and also the generic AUTOSAR methodology, the specification of the communication network is not done manually and early in the development workflow, but instead it is generated automatically as a last step. The design of TDL modules is completely platform-independent and lets the supplier focus on the functionality to implement without having the target platform in mind. When using the Simulink-integrated TDL:VisualCreator, the behavior of the modeled functionality can be accurately simulated. The supplier can also utilize the fact that TDL modules behave exactly the same on any (distributed) platform by testing the functionality in a real car by deploying it to any platform for which a TDL runtime system exists. The fact that it is sufficient to test functionality only in the Simulink simulation or on one hardware platform also greatly reduces the testing efforts.

For the OEM, the TDL methodology provides the flexibility of choosing the hardware platform, i.e. the ECUs and all connecting communication infrastructure, after all functionality is implemented and not beforehand. Suppliers do not provide complete ECUs but instead TDL modules and corresponding functionality code. The mapping of TDL modules to ECUs is then up to the OEM, who can then for example select numerous less powerful nodes or a small number of powerful nodes in an effort to reduce costs, increase reliability or improve electrical stability very late in the development process. Another example is the selection of the communication bus: On basis of the actual bandwidth requirements, the OEM can choose for example between CAN, FlexRay [3] and TTEthernet [6] without redesigning or retesting the software, as it is guaranteed that it behaves the same as long as TDL is able to generate code for the specific hardware platform.

5 Transition from Today's Workflow to the TDL Workflow

As the TDL methodology introduces fundamental changes to the current workflow, we are aware that the transition will be a difficult task. However we think the advantages outlined above are strong arguments and that the transition will quickly pay off. This will be especially true if an OEM does not want to commit to a specific communication protocol and wants to be able to change it easily. The TDL tools provide a single development environment that can be adapted to any existing target platform by developing a plug-in and runtime system for it. Choosing the hardware late in the development process avoids pessimistic hardware choices or complex analysis on what platforms might be adequate to perform the required functionality.

Suppliers can reuse their functionality code or Simulink models and construct TDL modules out of them. However they need to make sure that the functionality still lies within the specification after adding LETs to all functions. The greatest benefit for suppliers is that they can focus on the functionality and develop in a platform-independent way and therefore are released from the burden of testing the same software repeatedly on different platforms.

6 Conclusion

We showed that while even the AUTOSAR methodology fails to fulfill its vision of proper platform abstraction, the TDL tools deliver this vision. We outlined how employing the LET concept finally enables the industry to move away from the traditional ECU-centric workflow to a truly software component-centric workflow. In our view, the newly proposed workflow would have a beneficial impact on the OEM-supplier relationship, leading to increased efficacy, productivity and flexibility.

References

[1] T.A. Henzinger, B. Horowitz, C.M. Kirsch. GIOTTO: A Time-Triggered Language for Embedded Programming. Proc. IEEE 91 (2003) 84–99.

[2] E. Farcas, C. Farcas, W. Pree, J. Templ. Transparent Distribution of Real-Time Components Based on Logical Execution Time. LCTES, Chicago, Illinois, 2005.

[3] Makowitz, R. und Temple, C. FlexRay - A Communication Network for Automotive Control Systems. Proc. WFCS 2006, pp. 207–212.

[4] Vector: DaVinci Tools Suite. http://www.vector-worldwide.com.

[5] J. Templ. Timing Definition Language (TDL) Specification 1.5. Technical Report, University of Salzburg, 2008.

[6] TTEthernet Specification. Available at http://www.ttagroup.org/ttethernet/overview.htm.

[7] Elekrobit: EB Designer Pro. http://www.elektrobit.com.

[8] AUTOSAR. Automotive Open System Architecture. http://www.autosar.org.

[9] The MathWorks: MATLAB/Simulink. http://www.mathworks.com.

[10] preeTEC: TDL Tools. http://www.preetec.com.