2013 39th Euromicro Conference Series on Software Engineering and Advanced Applications

A Holistic (Component-based) Approach to AUTOSAR Designs

Kung-Kiu Lau, Petr Štěpán, Cuong Tran
School of Computer Science
The University of Manchester
Oxford Road, Manchester M13 9PL
United Kingdom
kung-kiu,pstepan,tranc@cs.man.ac.uk

Sébastien Saudrais, Borjan Tchakaloff
Embedded Systems Team
CERIE/Estaca
Rue Georges Charpak, BP76121
53061 Laval Cedex 9, France
s.saudrais,b.tchakaloff@estaca.fr

***Abstract*—The development of new automotive functions in a car must comply with the AUTOSAR standard. Such functions are distributed over the ECUs of the car connected with buses. The development of these distributed functions is not easy in AUTOSAR because there is no global view of the system. In this paper we propose an approach that counters this difficulty by designing automotive systems with a global view and then transforming these systems into AUTOSAR systems.**

***Keywords*—Component-Based Development, AUTOSAR, Embedded Systems.**

I. Introduction

AUTOSAR (AUTomotive Open System ARchitecture) [1] is a standard for automotive electricals/electronics engineering introduced by automotive manufacturers, suppliers and tool developers. AUTOSAR aims to break up the coupling between the hardware infrastructure and the application software, so that by standardising the architecture, it makes it easier for designers to cope with the ever increasing complexity of embedded automotive systems. It also frees up maximum effort and resources that can now be focused on software design. AUTOSAR's development methodology is model-driven. The software architecture, as well as the ECU hardware and the network topology are modelled in a formal way as a meta-model that supports the software development process from architecture up to integration. The functional software is developed externally and then incorporated into the architecture.

The basic unit of AUTOSAR is the Electronic Control Unit (ECU), which provides a set of automotive functions. The functions are implemented by software components deployed in ECUs, and therefore AUTOSAR is said to be based on the component paradigm, which reduces development effort, time and cost, through software reuse. For each ECU, a set of tasks is defined. Each task executes a sequence of operations, which include input/output to/from sensors/actuators, as well as calls to functions (called runnables) defined in the components. The behaviour of an ECU is the execution of all its tasks; and the behaviour of the whole system is the execution of all the tasks in all the ECUs.

An AUTOSAR system is an embedded system consisting of a number of ECUs (around 80 in high-end cars) connected to a bus (Fig. 1). Each ECU contains software components (SWCs in Fig. 1) inter-connected via their ports. Components

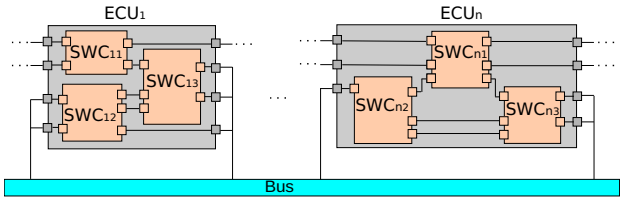

Fig. 1: ECUs with software components.

in different ECUs can be connected indirectly via the bus (or several buses or channels). The unconnected ports in Fig. 1 are connected to ECUs through buses, connections to sensors and actuators are implicit as part of the ECU hardware. Thus all AUTOSAR systems have the same hardware architecture, namely the one shown in Fig. 1. Also, AUTOSAR does not define formal semantics for software or software architectures. Consequently, there are two major drawbacks: (i) there is no software architecture for the whole system (there are only software architectures for individual ECUs); (ii) there are no tasks defined for the whole system (there are only tasks for individual ECUs).

In the context of CBD (Component-Based Development), these drawbacks are especially severe. Without a software architecture or tasks for the whole system, it is difficult to specify or ascertain the behaviour of the whole system in terms of the behaviour of individual components. Consequently, it is difficult to design separate architectures and tasks for individual ECUs whilst ensuring that together they achieve the desired behaviour of the whole system.

In this paper, we propose a way to counter this inherent lack of a global view for AUTOSAR systems. Specifically, we propose a holistic approach that provides a global view of AUTOSAR designs. Our approach is component-based, i.e. it is based on a suitable component model. Our holistic approach constructs a complete automotive system using the component model, and then transforms the system into an AUTOSAR system (with multiple ECUs).

II. A Holistic Approach

Our approach is based on the X-MAN component model [2], [3] and its associated tool [4], [5]. An X-MAN system is a software architecture which also has behaviour. The idea is to first construct the required system as an X-MAN system and

978-0-7695-5091-6/13 $26.00 © 2013 IEEE
DOI 10.1109/SEAA.2013.14

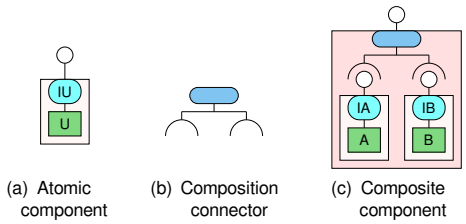

Fig. 2: The X-MAN component model.

then transform this system into an AUTOSAR system. The X-MAN system is completely independent of ECUs, but can be mapped to an AUTOSAR system with an arbitrary number of ECUs. Now we briefly present the X-MAN component model.

A. The X-MAN Component Model

In the X-MAN component model (Fig. 2), there are two basic entities: (i) *components*, and (ii) *composition connectors*. Components are units of design with behaviour, exposed through a component's interface as provided services. Composition connectors are composition mechanisms for components.

Components can be atomic or composite. An atomic component (Fig. 2(a)) is formed by composing an invocation connector (*IU*) with a computation unit (*U*). The computation unit contains an implementation of some behaviour (methods) in a chosen programming language, e.g. C. More importantly, the computation unit must have complete behaviour, i.e. it is *not* permitted to call another computation unit (of another component) in order to complete its behaviour. The invocation connector exposes the methods of the computation unit to the component's interface and allows them to be executed.

Composition connectors (Fig. 2(b)) are control structures. They are used to compose components into composite components. In a composite component (Fig. 2(c)), sub-components do not call each other. Instead, the composition connector *coordinates* the execution of sub-components. Two basic composition connectors are *Sequencer*, for sequencing, and *Selector*, for branching. In addition, there are special connectors that are not used for composition, but for adapting single components. These connectors are called *adaptors*; examples are *Loop*, iteration over a single component, and *Guard*, for gating.

Data routing can be *horizontal* or *vertical*. *Horizontal* data routing is between sub-components within a composite component. *Vertical* data routing is data propagation between the interface of a composite component and its sub-components. A data route is defined by a data channel. A data channel has a capacity of 1, and can have two possible read policies: *destructive* and *non-destructive*. Moreover, a data channel can be initialised to contain an initial value.

An X-MAN architecture is always executable because it has complete definition of control, data and computation. The execution semantics for X-MAN is control-driven. Component execution is initiated and coordinated by (composition) connectors. Connectors initiate control to invoke components. At each component, control then triggers read and write operations on data channels associated with the component in order to supply inputs to the component, and distribute the outputs from the component.

Finally, in X-MAN, components can have contracts that describe functional and compositional properties, e.g. data range, worst case execution time (WCET), etc. The contracts of atomic components enable formal validation and verification of component implementation. From the contracts of atomic components, contracts of composite components and systems can be generated compositionally.

B. Transforming X-MAN to AUTOSAR

In our approach, we first design the structure and behaviour of a system as an X-MAN architecture, independent on AUTOSAR. The next step is to transform the X-MAN system architecture into an AUTOSAR system, based on the specified number of ECUs. In this section we explain this transformation by defining the mappings for basic X-MAN elements (Loop and Guard are omitted for the lack of space).

1) Atomic Components: Mapping atomic X-MAN components to AUTOSAR is straightforward. An atomic X-MAN component corresponds to an atomic AUTOSAR software component (SWC). A *service* in an atomic X-MAN component corresponds to a *runnable* in an atomic AUTOSAR SWC. (Note that an atomic X-MAN component does not define any task in AUTOSAR.)

For example, we have an atomic X-MAN component C1 with two services S1 and S2. These are transformed into an atomic SWC1 with two runnables R1 and R2. In this example, obviously, the two runnables R1 and R2 could be mapped to different ECUs. In general, services from an atomic X-MAN component can be mapped to runnables on an arbitraty number of ECUs.

2) Composite Components: Unlike an atomic component, a composite X-MAN component also defines tasks (as well as SWCs). Since in AUTOSAR, tasks are defined and managed by ECUs, the transformation of composite X-MAN components has to take into account the number of ECUs being deployed. Without loss of generality, we will discuss transformation rules from X-MAN to AUTOSAR for two scenarios: for 1 ECU and for 2 ECUs. The 1-ECU scenario will show how tasks and runnables can be derived (from X-MAN components), whilst the 2-ECU scenario will show how tasks and runnables can be distributed among an arbitraty number of ECUs. For simplicity, we assume that ECUs are 'free'. In other words, we do not take into consideration other configuration factors such as free memory, load limit, etc.

For a composite component, we need to transform its (sub-)components, composition connectors and data channels. Data channels are simply mapped to AUTOSAR data links. Sub-components can in turn be composite and hence their transformation is recursive and top-down. Composition connectors initiate control to coordinate the execution of the sub-components. Therefore they define *tasks* in AUTOSAR. Accordingly we map a composition connector to a distinguished runnable (inside a distinguished SWC) with *triggering data links* to the other runnables in the same task.

As an illustration, consider the generic composite component CC2 in Fig. 3(a) with two levels of composition. As depicted, CC2 has a sub-component CC1 that is in turn another composite component. Suppose that the transformation is to map to two ECUs with CC1 on a separate ECU (from CC2).

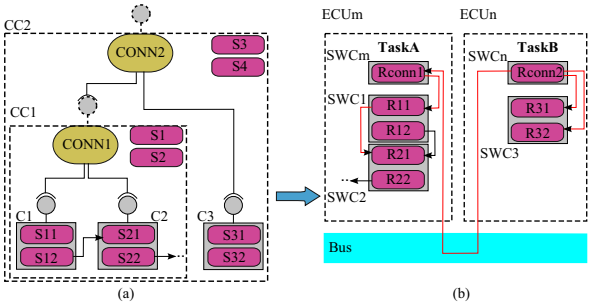

Fig. 3: Transforming a composite X-MAN component.

3) Sequencer: A *sequencer* enables sequential invocations of components according to their indices. Consider the composite in Fig. 4 (a) in which two components C1 and C2 are composed by a sequencer SEQ1. C1 offers services S11 and S12; C2 offers S21 and S22. The composition yields a composite component (dashed box) with two services S1 and S2: the former offers the sequential invocation of S11 and S21; the latter gives the sequential invocation of S12 and S22. Data may be passed between S1i and S2i but we omit that here. The decision to execute which sequence is provided by a control value, which could come from the system user interface.

For 1-ECU configurations, the X-MAN composite component in Fig. 4(a) is transformed to an equivalent AUTOSAR system in Fig. 4(b). X-MAN components C1 and C2 become AUTOSAR software components SWC1 and SWC2 respectively. Services S1i and S2i become runnables R1i and R2i. Transformation of sequencer SEQ1 means transforming all sequential invocations, i.e. S1 and S2, and to implement the sequence selection. To realise the sequences, the runnables are added with data links (and necessary data ports) (e.g. R11→R21) and set to be data triggered. In order to achieve the sequence selection, the transformation creates a new AUTOSAR software component SWC0 containing a time-triggered runnable called Rentry. Rentry has data links to the head runnables (R11 and R12) of the two sequences. Rentry is defined such that it takes a control value and selectively sends a trigger value to one of the head runnables. The periodic triggering of Rentry needs to be *at least* the worst WCET of all sequences Si. Finally, the transformation creates a non-preemptible task called Task1, and places the runnables in the order as they appear in Fig. 4(b). As a result, Task1 will execute Rentry, R1i and then R2i cyclically.

The transformation for 2-ECU configurations results in the same elements as those for 1-ECU configurations. However, the transformation now creates two non-preemptible tasks, Task1 and Task2, for ECU1 and ECU2 respectively. Task1 consists of SWC0 and SWC1; Task2 includes SWC2. Some data links (e.g. R11→R21) therefore have to go through a bus,

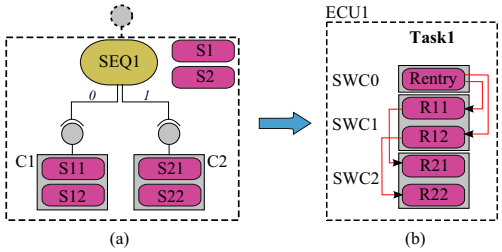

Fig. 4: Transformation for *Sequencer* for 1 ECU.

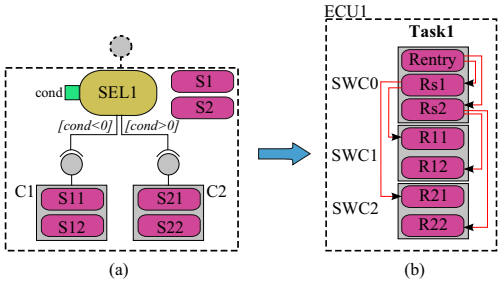

Fig. 5: Transformation for *Selector* for 1 ECU.

which adds some delay to the sequence execution time. In addition, imposing the sequences S1 and S2 and the sequence selection on distributed runnables means that any runnables on Task1 can never execute before any runnables on Task2 finish. In order to achieve this, the calculation of periodic triggering of Rentry has to be *at least* the sum of the worst-case delay (WCD) on the bus and the worst WCET of all sequences Si.

4) Selector: A *selector* allows for conditional invocation of components. Selection conditions are set based on the inputs to the connector. Consider the composite in Fig. 5(a) in which two components are composed by a selector SEL1 with an input *cond*. The selection condition for C1 is *cond*<0 and the selection condition for C2 is *cond*>0. We assume the composite component to have two services S1 and S2. S1 offers selection of S11 and S21, while S2 offers selection of S12 and S22.

For 1-ECU configurations, the transformation of components and services is performed in the same way as before. It is illustrated in Fig. 5(b). X-MAN component Ci becomes AUTOSAR component SWCi and Sij becomes Rij. The transformation of selector SEL1 means realising S1 and S2, and the selection of Si. In order to realise Si, the transformation defines two new runnables Rs1 and Rs2 for S1 and S2 respectively. Rs1 is defined so that it takes an input and outputs a triggering value to R11 iff the input value is less than 0, or to R21 iff the input value is greater than 0. Rs2 is defined similarly for R12 and R22. The transformation also adds necessary data links as depicted in Fig. 5(b), and the runnables Rij are set to be data triggered. The selection of Si is implemented as the runnable Rentry that is defined to take a control value and choose either Rs1 or Rs2. The transformation additionally adds data links to connect Rentry with Rsi, which are also set to be data triggered. The periodic triggering of Rentry is calculated to be at least the worst WCET of all Si. Finally, the transformation creates a non-preemptible task, namely Task1, to include and execute all the runnables in the order as they appear in Fig. 5(b).

For 2-ECU configurations, the transformation creates the same elements as that for 1-ECU configurations. The transformation, however, creates two tasks, Task1 and Task2, for two ECUs. Task1 consists of SWC0 and SWC1; Task2 contains just SWC2. As a result, some data links have to go through a bus. In addition, we have to guarantee that the selection of runnables cannot start until previously selected runnables complete their execution. This requires that Rentry can not be executed before any runnables in Task2 complete. Achieving this requirement means that the periodic triggering of Rentry has to be *at least* the sum of the worst-case delay (WCD) on the bus and the worst WCET of all Si.

5) Implementation of Transformation: We used Acceleo [6] to implement the transformation. The transformation takes three parameters: a model, an allocation configuration, and an output directory. A model is an X-MAN system architecture. An allocation configuration specifies a list of ECUs and an allocation of system components into these ECUs. An output directory is for containing AUTOSAR system descriptions in Artext [7]. The result of the transformation consists of two descriptions contained in two separate files: (i) one contains definitions of components, runnables, ports, and data links; (ii) one contains definitions of tasks.

III. Steer-by-Wire system

Now we illustrate our holistic approach to AUTOSAR designs by applying it to the Steer-by-Wire system.

A Steer-by-Wire system consists of three main physical parts: the steering wheel, controllers and the road wheels of the vehicle. Sensors on the steering wheel detect the angle and torque provided by the driver and feed this data to the controllers. The controller combines that data with torque data from the wheels. The result is then sent to the actuator attached to the wheel to turn it. At the same time, wheel sensors detect the torque and angle of the wheel turn to be fed back to the controllers. This data is combined with the steering wheel torque as well as vehicle speed and is used to compute the amount of feedback to return to the steering wheel. The computed result is then sent to the actuator attached to the steering wheel to provide feedback to the driver to give the sensation of an actual mechanical wheel turn on different kinds of roads [8]. The main functions of Steer-by-Wire are the Feedback Torque and Rack Torque functions, depicted in Fig. 6. The main constraints are to satisfy: (i) the desired response time of the two main functions; (ii) the driver's preference for his feel when he turns the steering wheel.

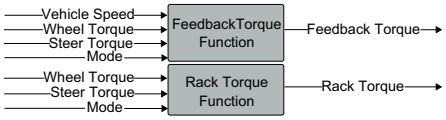

Fig. 6: Main functions of Steer-by-Wire.

A. Steer-by-Wire in X-MAN

The first step of our approach is to build the Steer-by-Wire system in X-MAN. This system is shown in Fig. 7. We have three components: SteerManager_Normal, SteerManager_City, and WheelManager. SteerManager_Normal and SteerManager_City have a service called ManageSteering. WheelManager has a service called ManageWheel.

We build the system in two steps. First, we compose the first two components by a selector SEL1 into a composite component and set the conditions. Second, we compose the composite component with the third component by a sequencer SEQ1 into a bigger composite component, which is the final system. The system has a system service called Manage. In addition, the inputs and outputs of the system service are routed to components and between components by a collection of data channels.

The system behaves as follows: when system service Manage is executed, control is passed by the sequencer SEQ1

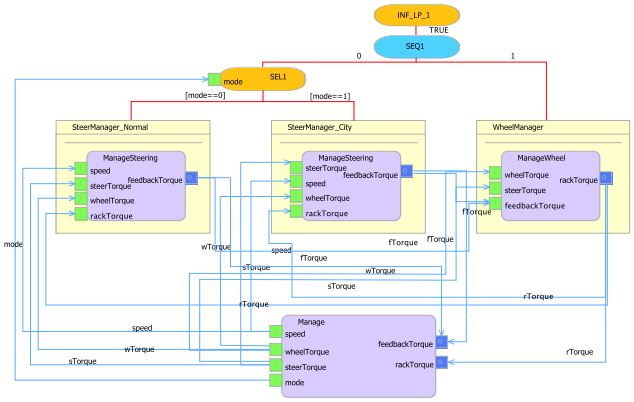

Fig. 7: Steer-by-Wire system in X-MAN.

to the selector SEL1 that then decides based on its input *mode* to invoke either components below it, control is then returned to the sequencer SEQ1 that invokes the last component.

B. Steer-by-Wire in AUTOSAR

Having the X-MAN model, the next step is to transform it into an AUTOSAR system, using our Acceleo transformation tool. However, we first need to set up an ECU architecture to house the input/output of the software system. The AUTOSAR representation is based on two ECUs, one for the steer side and one for the wheel side. Four unconnected ports correspond to the inputs and the outputs of the Steer-by-Wire system (Fig. 6) for the steer side. Two network ports, *speed* and *mode*, have their data coming from a bus, while the four other ports are contained by components and linked to the hardware.

Now, we present the results of transforming the X-MAN model into AUTOSAR for the given architecture. The transformation produces the AUTOSAR elements only for the behavioural part. These elements need to be imported into the empty AUTOSAR architecture with the corresponding inputs and outputs, to/from the hardware and the network. During the importation, links are created between the input/output of the developed system and the existing architecture by replacing the X-MAN Manage service with the existing ECU ports.

The importation of the first description file (component definition, runnables, ports and data links) is performed in the ECU architecture. The transformation has to be parametrised with the 2-ECU option. The choice of component allocation can be made by the designer or with an allocation algorithm [9]. For the steer ECU, three component types are created, one for each X-MAN component (SteerManager_Normal and SteerManager_City) and one for the connector tree, corresponding to the sequencer and selector connectors. A new port type Trigger is created for the triggering of the runnables: it consists of a boolean signal. The other port types are already present in the architecture as I/O of the system. The ConnectorTree component has four runnables, each one triggered by data reception. Two runnables correspond to the selector connector, begin and end, and the two others to the sequencer. The other components have one runnable, also data triggered. Three delegated ports are generated because of the absence of the wheel manager on the ECU and the triggering of the wheel runnable on the wheel ECU. The delegated ports send data over a bus so the bus database is updated with

the new signals. Fig. 8 shows the imported components and their port connections (the existing components of the empty architecture are omitted). The ConnectorTree component has ports connected to itself, due to the composition of a selector and a sequencer. The beginning of the sequencer is directly followed by the selector and the end of the sequencer is connected to the beginning to restart the sequence.

The importation of the second description file (tasks definition) is performed on the complete ECU architecture. Two tasks are already present on the ECU configuration: SchM_Task for the ECU initialization and Task_Sensor for the hardware abstraction and the sensors/actuators runnables. The task mapping of the Steer-by-Wire system is composed of one task TaskSbW, containing all the generated runnables. For each execution of the task, only the triggered runnables will be called. As the two runnables *SteerManager_Normal_Behaviour* and *SteerManager_City_Behaviour* are selected by a selector, they cannot be called at the same time. Two different execution paths are possible for the generated task containing only one of these two runnables. The transformation process also produces the code of the connector runnables.

IV. Evaluation

The generated AUTOSAR systems have to be evaluated to check if they satisfy the timing requirements of the steer-by-wire system. The evaluation is performed first on the AUTOSAR architecture model and then on a NEC V850 board connected with a CAN bus to a simulation bench. We assume that the worst-case execution time (WCET) of the components is known.

Architecture Model Evaluation: An AUTOSAR architecture's timing properties can be checked with a tool like Symta-S using techniques described in [9]. For our evaluation, 5ms is given as the desired response time. The tool explores the different possible system execution paths and compares their execution time to the required response time. The length of the physical cable and the transmission through the low-level layers has an estimated execution time of 1 ms. Two different paths of the generated task have to be explored depending on the value of mode. The two paths are given by the X-MAN model. All the paths are evaluated and the total time of the two paths is less than 5ms, as required.

Application Evaluation: The application is deployed on two ECUs. The generation of the architecture code is done with the DaVinci tool and the code is compiled with the GreenHills compiler. The runnables' code is generated during the transformation based on the type of the X-MAN connector:

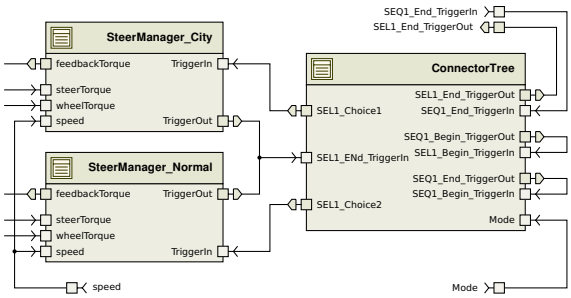

Fig. 8: AUTOSAR steer components for 2-ECUs.

an if statement for a selector or a port writing for a sequencer. The evaluation is performed with the tool Vector CANoe, the bus interface CANCaseXL, and the test bench VT System. CANoe sends sensor data to the VT System connected to the physical input ports of the boards and checks the time reception of the output of the Steer-by-Wire system. CANoe is also responsible for sending the mode and speed messages through the CAN bus using the CANCaseXL interface.

As was the case for the architecture model evaluation, the response time of the two architectures is also valid regarding the expected time response.

V. Conclusion

In this paper we have presented a holistic approach to designing AUTOSAR applications with X-MAN. The designer has a global view of a system, in terms of complete software architecture and overall behaviour, without being restricted to the ECU view used in AUTOSAR tools. A transformation has been defined to produce AUTOSAR models from X-MAN. The same X-MAN model can be transformed into different AUTOSAR architectures with different numbers of ECUs, all of which have the same behaviour as the runnables are data-triggered. The approach has been validated on AUTOSAR compliant boards.

As on-going work, we are optimising the transformation to avoid port connections within the same component and to reduce the number of generated runnables. We also intend to adapt the transformation to other popular platforms, such as AADL [10].

References

[1] AUTOSAR Partnership, "AUTOSAR - the worldwide automotive standard for E/E systems," 2011.

[2] K.-K. Lau, P. Velasco Elizondo, and Z. Wang, "Exogenous connectors for software components," in Proc. 8th Int. Symp. on Component-based Software Engineering, LNCS 3489. Springer, 2005, pp. 90–106.

[3] K.-K. Lau, M. Ornaghi, and Z. Wang, "A software component model and its preliminary formalisation," in Proc. 4th Int. Symp. on Formal Methods for Components and Objects, LNCS 4111. Springer-Verlag, 2006, pp. 1–21.

[4] K.-K. Lau and C. Tran, "X-MAN: An MDE tool for component-based system development," in *Proc. 38th EUROMICRO Conference on Software Engineering and Advanced Applications*. IEEE, 2012, pp. 158–165.

[5] N. He, D. Kroening, T. Wahl, K.-K. Lau, F. Taweel, C. Tran, P. Rümmer, and S. Sharma, "Component-based design and verification in X-MAN," in *Proc. Embedded Real Time Software and Systems*, 2012.

[6] Obeo, "Acceleo User Guide," Tech. Rep., Oct. 2011. [Online]. Available: http://www.obeonetwork.com/page/acceleo-user-guide

[7] C. Knüchel, M. Rudorfer, S. Voget, S. Eberle, R. Sezestre, and A. Loyer, "Artop – an ecosystem approach for collaborative AUTOSAR tool development," in *International Congress on Embedded Real Time Software and Systems*, 2010.

[8] K. Chaaban, P. Leserf, and S. Saudrais, "Steer-by-wire system development using autosar methodology," in *Proceedings of the 14th IEEE international conference on Emerging technologies & factory automation*, ser. ETFA'09. Piscataway, NJ, USA: IEEE Press, 2009, pp. 1110–1117. [Online]. Available: http://dl.acm.org/citation.cfm?id=1740954.1741107

[9] A. Daghsen, K. Chaaban, and S. Saudrais, "Software function allocation and configuration of an autosar-compliant system," in *SAE 2012 World Congress & Exhibition*, Detroit, Michigan, USA, April 2012.

[10] P. H. Feiler and D. P. Gluch, "Model-based engineering with aadl: An introduction to the sae architecture analysis & design language," 2012.