Modeling and Assessment of Safety Critical Systems

Thomas Barth
Department of Electrical Engineering
Darmstadt University of Applied Sciences
Darmstadt, Germany
thomas.barth@h-da.de

Prof. Dr.-Ing. Peter Fromm
Department of Electrical Engineering
Darmstadt University of Applied Sciences
Darmstadt, Germany
peter.fromm@h-da.de

Victor Pazmino Betancourt
Embedded Systems and Sensors Engineering
FZI Research Center for Information Technology
Karlsruhe, Germany

Prof. Dr.-Ing. Dr. h. c. Jürgen Becker
Embedded Systems and Sensors Engineering
FZI Research Center for Information Technology
Karlsruhe, Germany

Bo Liu
Embedded Systems and Sensors Engineering
FZI Research Center for Information Technology
Karlsruhe, Germany

***Abstract*— With growing complexity of embedded controllers and applications, the design of safety critical systems becomes more and more challenging. Tools and frameworks help to manage such challenges but are often pricy and cover only certain aspects of the overall design or implementation workflow. In previous publications, we introduced a lightweight runtime environment and discussed concepts for separation of signal paths on multicore controllers as well as safety monitoring mechanism. As part of the publicly funded ZIM project "Zukunftstechnologie Multicore - Safe&Secure", the Darmstadt University of applied sciences cooperates with the FZI Research Center for Information Technology and the company HighTec. One goal within this project is the development of tooling, which incorporates the results of previous research and not only allows modeling and code generation for safety critical systems, but also allows assessment of the safety cases and their mapping to the actual implementation in order to ease qualification. In this paper, we will demonstrate how free frameworks such as Eclipse EMF can be used to implement modelling tools. We will show how user friendly GUIs can be implemented, how safety assessment can be performed and how code can be generated.**

Keywords—EMF; Eclipse; code generation; multicore; functional safety; RTE; framework;

I. INTRODUCTION

Embedded controllers become more and more powerful in terms of computational power. With rising computational power, software running on such controllers also becomes more complex [1], especially if multicore controllers are used. Complex applications like ADAS have to be executed fast, deterministic and in safe way, while they might be only part of the overall design, interacting with other applications executed on the same device. Designing a state of the art safe control system is not a trivial task and validation and certification of such a system becomes more and more challenging due to the increasing complexity.

Most safety standards like the IEC61508 or ISO26262 require a formal description of the system design, which is often realized in SysML or UML. However, such languages have limitations, especially when it comes to non-functional safety requirements such as freedom from interference. Another challenge is an efficient traceability between the safety case, the architecture and the implementation, which is an important prerequisite for developing a valid safety case. Furthermore, the diagrams often need to be manually synchronized with the implementation, which is prone to error.

In order to provide a possible solution for the stated challenges, the Darmstadt University of applied sciences cooperates with the FZI Research Center for Information Technology and the company HighTec as part of the publicly funded ZIM project "Zukunftstechnologie Multicore - Safe&Secure". The project aims to develop a single tool, which combines hard- and software modeling as well as safety goal mapping, assessments and code generation. As compared to existing solutions, the tool should be lightweight so that it can be used by small and medium sized companies.

II. Modeling Tool Concept

The tool shall work in a "what you see is what you get" fashion, which means that the visualization of the architecture and code generation work on the same data model. This ensures that the code is in harmony with the model, which eases testing/validation/qualification. Moreover, the tool shall have the following features:

A. System Perspective

Starting from safety requirements, down to the hard- and software architecture, all diagrams should be based on the same meta-model. This allows mapping and referencing between entities in different diagrams and helps to keep the overall design consistent. Different graphical and tabular views provide an efficient interface to enter and maintain the model.

B. Signal-Flow based Views

All diagrams follow the concept of signal flows. Functional-Flow diagrams are a rather abstract and more or less technology independent view on a system with the focus on a particular functionality. During early design phases, the functionalities of the system can be designed high-level, before individual software-blocks are mapped to the underlying framework. Signals, connecting software-blocks and realizing data-exchange, shall be generated using the signal-layer presented in earlier publications [2].

On the next level, the abstract functional flow is mapped to hardware and software components. Hardware diagrams represent the physical structure of the system, including ECU's, sensors, actors, busses and similar.

On software level, the static structure of the system as well as the data flow and activation of runnables is modelled. The runnables are mapped to OS tasks and signal pools to encapsulate groups of signals.

C. Code generation

Based on the model, the code framework will be generated, including

- Software components and runnables with their signal interfaces
- MPU protected signals
- Activation triggers (Cyclic, OnStateChange, OnDataUpdate)
- Interfaces to the driver layer

The code generated by the tool has to be easy to read and easy to qualify. In addition to the code, technical documents will be generated.

D. Safety Focus

The tool is focusing on safety related control systems. It shall be possible to define safety goals for safety functions and reference the taken safety measures in order to ease certification.

Once the system is modeled, it shall be possible to perform assessments. Therefore, it needs to be possible to compute relationships between entities in the model and access their attributes.

III. EMF-based Implementation

A framework, that provides the possibility to implement the stated concept, is the Eclipse Modeling Framework (EMF). EMF is an open-source Java modeling framework and code generator, based on Eclipse. Ecore is the core of EMF and the central interface for a variety of available extensions (Figure 1). We chose the "Obeo Designer" solution, which integrates graphical extensions as well as a code generator extension. There is a free community version of Obeo Designer available[1].

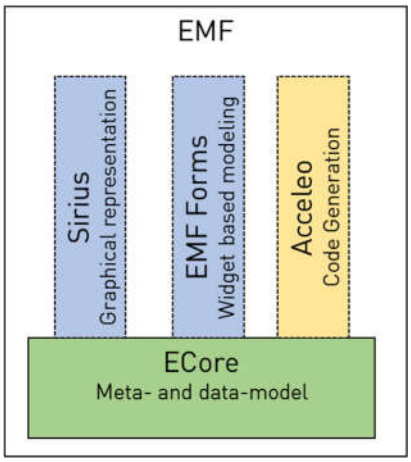

Figure 1 EMF with extensions

A. Meta-model

The meta-model is very close to an UML class diagram. Meta-models define the data-objects and their attributes that can be created within the data-model, where attributes might be relationships to other objects in the meta-model. In our tool, the meta-model describes a complete control system, including hard- and software, signal-flows, safety goals etc. There are own meta-models for each part (e.g. different meta-models for hard- and software), which are linked together in an integration meta-model. The meta-models (Figure 2) are the base of an EMF project and therefore defined in Ecore, which is propagating the meta-models to all extensions.

[1] Obeo Designer is available at www.obeodesigner.com
It is also possible to import the required extensions into a "normal" Eclipse installation, e.g. using the "Modeling Tools Package" from eclipse.org/downloads/packages/release/2018-12/r/eclipse-modeling-tools

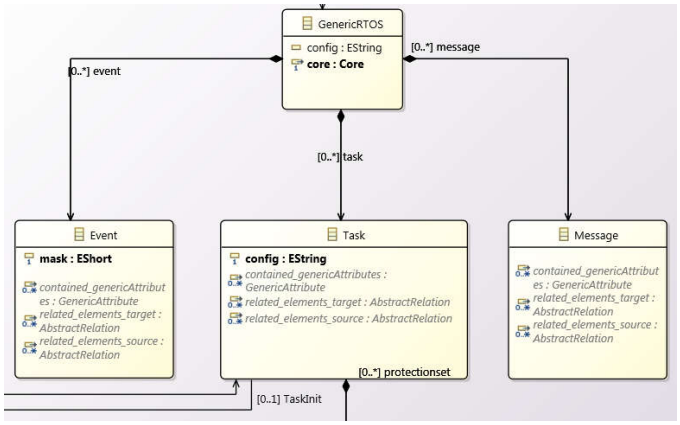

Figure 2 EMF Meta-Model

B. Modeling

Based on the data structure defined in the meta-model, a model of the system under development can be created. EMF stores the data-model in an XML format and a basic editor is part of Ecore. Especially during meta-model design phases, the automatically generated tree-editor (Figure 3) is a helpful feature not only to verify a successful build, but also to perform plausibility checks with test data.

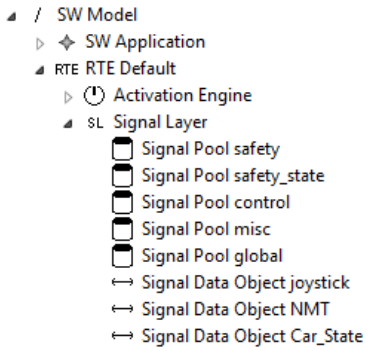

Figure 3 EMF Tree editor

EMF provides a number of editor extensions, which allow data manipulation in the data-model. Since there should be the possibility to design the system using a graphical editor, the "Sirius" (Figure 5) extension was chosen. Sirius allows mapping of data-model entities to visual elements and configuration of the visual appearance of such elements.

Another extension for modeling is "EMF Forms". EMF Forms allows data manipulation with well-known widgets like textboxes or dropdown lists (Figure 4).

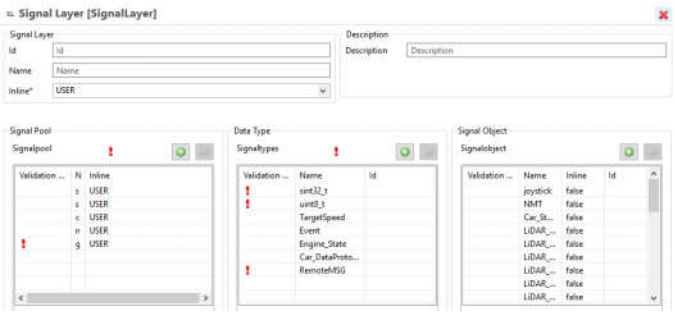

Figure 4 EMF Forms

It shall be mentioned that EMF Forms cannot iterate trough the model to collect data for a certain view, but is primary used to enter details for a single class within the meta-model.

C. Code generation

In order to generate code, the "Acceleo" extension was chosen. Acceleo introduces the Acceleo Query Language (AQL), which is a superset of the Object Constraint Language (OCL), which is part of the UML. For each file to be generated, a template has to be written using AQL, where AQL allows the usage of raw OCL or invocation of Java methods. Since Acceleo is aware of the meta-model structure, it is possible to navigate through the data-model, using simple expressions like in Snippet 1. The output of the generation process can be seen in Snippet 2. With Acceleo, it is not only possible to generate code-files but also documentation or report files.

```
[for (pool : SignalPool | aSignalLayer.signalpool->sortedBy(name))]
[h1(pool.name)/]
/*  [pool.description/]
 *
 * RAM section: [pool.RAMSection.name/] ([pool.RAMSection.description/])
 * ROM section: [pool.ROMSection.name/] ([pool.ROMSection.description/])
 */
.SL.[pool.name/] : {
          . = ALIGN(8);
          PROVIDE([pool.pool_LinkerSym_BEG()/]= .);

          *([pool.RAMSection.name/])

          . = ALIGN(8);
          PROVIDE([pool.pool_LinkerSym_END()/] = .);

} > <memory>


[/for]
```

Snippet 1 Acceleo script output section

```
/*=========================================[ control  ]======*/
/*  Contains all signals required for controlling the car
 *
 * RAM section: .sl_control ()
 * ROM section: .rodata ()
 */
.SL.control : {
          . = ALIGN(8);
          PROVIDE(ADRL_SL_CONTROL_BEGIN= .);

          *(.sl_control)

          . = ALIGN(8);
          PROVIDE(ADRL_SL_CONTROL_END = .);

} > <memory>
```

Snippet 2 Acceleo result output section

D. Assesment

As the meta-model is stored as Java classes, it is possible to navigate through the data-model using Java. With Java as a general-purpose language, it is possible to perform even complex assessments on the data-model, while the API for the Java classes is automatically generated based on the meta-model and provides methods to access relationships within the meta-model.

IV. STATUS AND DEMONSTARTOR

Currently (1/2019), the meta-models for the runtime environment and the user-application have reached a first stable version. It is possible to draw basic diagrams using Sirius and generate the signal-layer and parts of the activation engine (invokes runnables) using Acceleo. A first meta-model for the RTOS PxROS is under development.

As part of our research, an omnidirectional robot is built, which shall act as an example. One operational mode of the robot is the autonomous navigation through a maze. As the driving function is considered safety critical, measures for functional safety need to be applied.

The following diagram (Figure 5), which is generated using the Sirius extension, shows the high-level signal flow of the intended functionality. It is possible to create individual diagrams showing only a selected subset of the complete model.

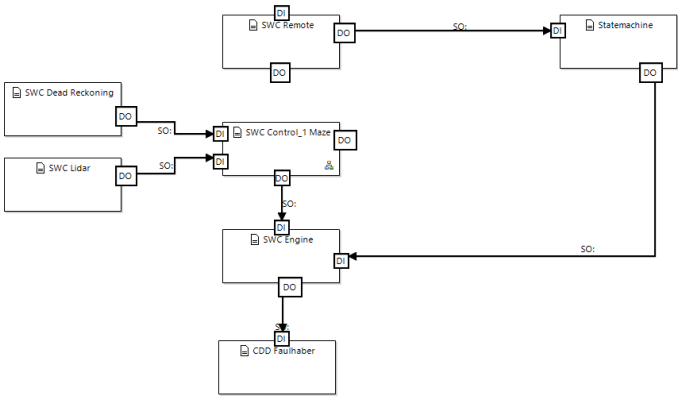

Figure 5 Example Sirius model function view

The remote signal coming from an external control unit is only used to check the connection status as one of the several safety features. In case of a broken connection, the car will be stopped by the safety state machine. The control component, responsible for driving the car, takes the LiDAR scan data and dead reckoning position data as input and calculates a control vector for the engines. This target speed is then passed to the engine and transmitted via a complex device driver to the CAN bus. The following detailed diagram (Figure 6) shows the internal structure of the control component.

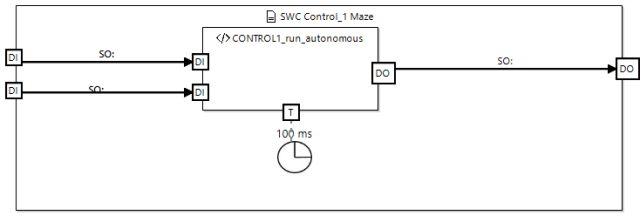

Figure 6 Example Sirius detail view SWC

In this example, we only have one runnable, which is cyclically called and processes the previously introduced input signals and calculates the target speed as output signal.

In the current implementation, the following files will be generated by Acceleo:

A. Signal Layer

For each signal defined in the model, an own C++ class will be generated. This has the advantage, that the signal API is tailored to the signal configuration and that the likelihood of confusion between signal objects is reduced. Each header-file will also contain additional information about the signal (Snippet 3). The user can decide if signals should be used as references or as deep-copy. If a signal shall be used by references, it will be instantiated as singleton, if deep copies are configured, there will be a signal instance for every context. Currently, only references are supported.

```
/**
 * \file   Car_TargetSpeed_signal.h
[...]
 * ----- Description -----
 * Targetspeed for the car
 *
 * ----- Signal properties -----
 * Name:              Car_TargetSpeed
 * SignalPool:        control
 * inline:            false
 * Signal-Type:       TargetSpeed (atomic: false; Primitive: false)
 *
 * ----- Associated drivers -----
[...]
 */

class Car_TargetSpeed{
[...]
```

Snippet 3 generated signal header

Signals are organized in signal pools, which group signals and instantiate the signal objects (Snippet 4). Moreover, they provide linker symbols that can be used to configure memory protection mechanism.

```
/**
 * \file   signals_control.h
[...]

 * ----- pool settings -----
 * inline:            USER
 * RAM Section:       .sl_control
 * ROM Section:       .rodata
[...]
 * ----- Signals associated with this pool (11) -----
 *
 *         Car_Position
 *                    Car position based on dead reckoning data
 *
[...]
 */

[...]
/** \brief Signal Object Car_Position
 *
 * Car position based on dead reckoning data
 */
EXTERN Car_Position so_Car_Position;
[...]
```

Snippet 4 generated signal pool

Besides the documentation in the header files, a markdown document will be generated for the Signal layer (Figure 7).

SignalLayer Overview

Signal Pools Overview

There are signals which have not been assigned to a signalpool!

- TEST (This signal does not have any functionality and is intended for generator tests only.)

Pool Name	Description	Signals
control	Contains all signals required for controlling the car	LiDAR_Feature; Car_Position; Car_TargetSpeed; Remote_MsgFromControl; MAZE_01; LidarScanForVector; DiagnosticTransferIn; LidarVector; DiagnosticScanTransfer; DiagnosticVectorTransfer; DiagnosticMapTransfer;
global	globally accessible signals	LiDAR_Trigger_PDO;

Figure 7 generated signal layer documentation

B. *Activation engine*

The activation engine is responsible to invoke requested runnables. Besides cyclic triggers, it is possible to invoke runnables e.g. if a signal has been updated. It is intended to implement the notification mechanism using RTOS messages in order to allow flexible allocation of tasks, especially on multicore controllers. Tables with all runnables to be invoked upon an event are generated by Acceleo (Snippet 5). Currently, only cyclic triggers are implemented.

```
/*======================[ Activation for task "C1_Control"  ]======*/

//----------- Cyclic events
/** Runnable table for Cyclic event "1000_ms" on task C1_Control. */
const AE_runtab_nr_t AE_C1_Control_CylicRunTable_1000_ms_A0_NR[] = {
    {DR_run_reportPosition, "DR_run_reportPosition"},
};

/** Runnable table for Cyclic event "100ms" on task C1_Control. */
const AE_runtab_nr_t AE_C1_Control_CylicRunTable_100ms_A0_NR[] = {
    {CONTROL_run_selectedFeature, "CONTROL_run_selectedFeature"},
};
```

Snippet 5 generated runnable table

C. *Linker description*

Especially on multicore controllers, every global object needs to be allocated to a specific location in memory. Acceleo automatically will generate the corresponding linker description file for all objects generated (Snippet 2).

V. CONCLUSION AND OUTLOOK

With the current pre-release, it is possible to model and generate the code framework for a complex multicore safety application. The model development for the demonstrator showed, that the graphical representation realized in Sirius for functional, software and hardware level is extremely helpful to understand the system behavior. The EMF Forms solution for entering detailed data is fast and intuitive. The generated code using the Acceleo extension is readable and performant.

Future work will include:

A. *Safety Assesment*

A meta-model for safety requirements needs to be defined. With such a meta-model, it will be possible to map safety measures to safety-goals, providing simple traceability and easier certification. Moreover, common rules like freedom from interference or the consistency of the configuration can be checked automatically.

B. *RTOS configuration*

A real time operating system is part of the meta-model as well. All dependencies towards RTOS objects, such as tasks, can be gathered automatically, which allows automated configuration.

ABBREVIATIONS

ADAS	Advanced Driver Assistance Systems
AQL	Acceleo Querry Language
CAN	Controller Area Network
ECU	Electronic Control Unit
EMF	Eclipse Modeling Framework
FZI	ForschungsZentrum Informatik
GUI	Graphical User Interface
HW	HardWare
LOC	Lines Of Code
MPU	Memory Protection Unit
OCL	Object Constraint Language
OS	Operating System
RTE	Run Time Environment
RTOS	Real Time Operating System
SW	SoftWare
SWC	SoftWare Component
UML	Unified Modeling Language
WCET	Worst-Case Execution Time
XML	Extensible Markup Language
ZIM	Zentrale Innovationsprogramm Mittelstand

REFERENCES

[1] H. Kopetz, “The Complexity Challenge in Embedded System Design,” in *Proceedings / The 11th IEEE Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing: Orlando, Florida, May 5 - 7, 2008*, Orlando, FL, USA, 2008, pp. 3–12.

[2] T. Barth and P. Fromm, “Warp 3 zwischen allen Kernen: Entwicklung einer schnellen und sicheren Multicore-RTE,” in *ESE Kongress 2016*.