RESILIENT INTERFACE DESIGN FOR SAFETY-CRITICAL EMBEDDED AUTOMOTIVE SOFTWARE

Harald Sporer, Georg Macher, Christian Kreiner and Eugen Brenner

Institute of Technical Informatics,
Graz University of Technology, Graz, Austria
{sporer,georg.macher,christian.kreiner,brenner}@tugraz.at
http://www.iti.tugraz.at/

ABSTRACT

The replacement of the former, purely mechanical, functionality with mechatronics-based solutions, the introduction of new propulsion technologies, and the connection of cars to their environment are just a few reasons for the continuously increasing electrical and/or electronic system (E/E system) complexity in modern passenger cars. Smart methodologies and techniques have been introduced in system development to cope with these new challenges. A topic that is often neglected is the definition of the interface between the hardware and software subsystems. However, during the development of safety-critical E/E systems, according to the automotive functional safety standard ISO 26262, an unambiguous definition of the hardware-software interface (HSI) has become vital. This paper presents a domain-specific modelling approach for mechatronic systems with an integrated hardware-software interface definition feature. The newly developed model-based domain-specific language is tailored to the needs of mechatronic system engineers and supports the system's architectural design including the interface definition, with a special focus on safety-criticality.

KEYWORDS

Embedded Automotive Systems, Hardware-Software Interface, Model-Based Design, Domain-Specific Modelling, Functional Safety

1. INTRODUCTION

Electrical and/or electronic systems (E/E systems) in the automotive domain have grown increasingly complex over the past decades. New functionality, mainly realized through embedded E/E systems, as well as the growing connectivity (Car2X-Communication), will keep this trend alive in the upcoming years. Well-defined development processes are crucial for managing this complexity and achieving high quality products. Wide-spread standards and regulations, such as Automotive SPICE® and ISO 26262, provide guidance through the development life cycle. Some of the key aspects of these concepts are full traceability and consistency between the different development artifacts.

In the automotive industry, the E/E system architectural design models are usually created with techniques based on the *Unified Modeling Language (UML)*. Either the meta-model is extended,

Jan Zizka et al. (Eds) : CCSIT, SIPP, AISC, CMCA, SEAS, CSITEC, DaKM, PDCTA, NeCoM - 2016
pp. 183–199, 2016. © CS & IT-CSCP 2016 DOI : 10.5121/csit.2016.60117

or a profile is created to make it possible to use the UML-based approach in embedded automotive system design. A wide-spread example of an UML2 profile is the *Systems Modeling Language (SysML)*, which reuses many of the original UML diagram types (*State Machine Diagram*, *Use Case Diagram*, etc.), uses modified diagram types (*Activity Diagram*, *Block Definition Diagram*, etc.), and adds new ones (*Requirement Diagram*, *Parametric Diagram*) [1].

Even if the UML-based methodologies are valuable for projects with an emphasis on software, they are sometimes too powerful for embedded automotive system design, due to the numerous representation options. Particularly for domain experts who have no or limited knowledge of software development, the large number of elements available for modelling, turns system architectural design into an awkward task. However, it is not the intention of this work to decry the SysML approaches created so far. They are a good choice for a multitude of tasks. Instead, this paper showcases an extension of these SysML approaches, which makes the architectural design process easier, placing a special focus on the specification of the hardware-software interface for UML non-natives.

A model-based domain-specific language and domain-specific modelling (DSM) has been developed for the specific needs of embedded automotive mechatronics systems. Additionally, a software tool has been created to support the new modelling techniques. By linking development artifacts such as requirements (e.g. technical system requirements, software requirements, etc.), and verification criteria to the design model, the traceability mentioned earlier is assured.

The main goal of this work is to contribute to the improvement of the existing system architectural design methods by facilitating the specification of the hardware-software interface. The approach presented has mainly been created for the development of embedded mechatronics-based E/E systems in the automotive field. However, the techniques are also suitable for other domains. Improvements have been made by extending the system modelling approach presented in previous publication using HSI specification capabilities.

Section 2 presents an overview of related approaches, domain-specific modelling and integrated tool chains. Section 3 provides a description of the proposed hardware-software interface specification approach for the model-based system engineering. An application of the methodology described is presented in Section 4. Finally, this work is concluded in Section 5, which gives an overview of the presented work.

2. RELATED WORK

In recent years, a lot of effort has been made to improve the model-based automotive E/E system design methods and techniques. Today, the advantages of a model-based approach are clear and without controversy. Meseguer [2] grants much more reliability, reusability, automatisation, and cost effectiveness to software that is developed with modelling languages. However, model transformation within or across different languages is crucial to achieve all these benefits.

Traceability and consistency between the development artifacts have always been important topics. However, these properties have become even more important due to the increasing number of electronic and electric-based functionalities. According to the international standard ISO 26262 [3], released in 2011, traceability between the relevant artifacts is mandatory for safety-critical systems. A description of the common deliverables relevant to automotive E/E

system development, and a corresponding process reference model is presented by the de facto standard Automotive SPICE [4]. Neither the functional safety standard nor the process reference model enforces a specific methodology for how the development artifacts have to be created or linked to each other. However, connecting the various work products manually is a tedious and error-prone task.

One of the early work products found in the engineering process is the system architectural design. In the field of automotive E/E system development, a wide-spread and common approach is to utilize a UML-based technique for this design, such as the UML2 profile SysML. Andrianarison and Piques [5], Boldt [6], and many other publications (e.g. [7], [8], [9]) present their SysML methodologies for system design. As stated by Broy et al. [10], the drawbacks of the UML-based design are still the low degree of formalization, and the lack of technical agreement regarding the proprietary model formats and interfaces. The numerous possibilities of how to customize the UML diagrams and how to get a language for embedded system design, are behind these drawbacks. Even if there is an agreement to utilize a common UML profile such as SysML, there are plenty of design artifact variations. This scenario does not provide an optimal base for the engineer who has to design the embedded automotive system from a mechatronics point of view. Ideally, the tool should be intuitive and it should be possible to use it easily without specific knowledge of UML.

Mernik et al. [11] describe a domain-specific language as a language that is tailored to the specific application domain. This tailoring should lead to a substantial increase in expressiveness and ease of use, compared to general-purpose languages. Even if expressiveness is increased by the utilization of SysML-based modelling techniques, the ease of use for embedded automotive mechatronics system design has not been improved.

Preschern et al. [12] claim that DSLs help to decrease system development costs by providing developers with an effective way to construct systems for a specific domain. The benefit in terms of a more effective development has to be greater than the investment needed to create or establish a DSL at a company or in a department. In addition, the authors argue that the mentioned DSL development cost will decrease significantly over the next few years, due to new tools that support language creation such as the Eclipse-based *Sirius*[1].

Vujovic et al. [13] present a model-driven engineering approach to creating domain-specific modelling (DSM). Sirius is the framework used to develop a new DSM and the DSM graphical modelling workbench. The big advantage of this tool is that the workbench for the DSM is developed graphically. Therefore, knowledge about software development with Java, the graphical editor framework (GEF) or the graphical modelling framework (GMF) is not needed.
Although it is obvious that an unambiguous specification of the various signals between the items of an embedded automotive system design is vital, publications on embedded automotive hardware-software interface definition are rare. This contribution aims to extend a model-based development approach for an ISO 26262 aligned hardware-software interface definition presented by the authors of [14]. More background on the origin of HSI characteristics is presented and the model-based support is shifted from a classic SysML-based methodology to a domain-specific modelling methodology for the E/E system architectural design of mechatronics-based systems.
The domain-specific modelling (DSM) language definition is presented in [15].

[1] https://eclipse.org/sirius/

3. APPROACH

The main goal of this contribution is to convey the importance of the hardware-software interface for today's *Embedded Automotive Systems* and how it is supported by the approach described. Moreover, the key driving factors for establishing a well-defined interface, which is also suitable for safety-critical applications, will be shown within this section. Before describing the HSI specification approach in detail, the utilized domain-specific model-based system architectural design technique shall be introduced. This domain-specific modelling method has been developed to outline mechatronics-based system architectures in the automotive sector and therefore serves as a basis for the specification of the hardware-software interface found in our approach.

3.1. Embedded Mechatronics System Domain-Specific Modelling

The key objective of domain-specific modelling is to provide a lean approach for engineers to facilitate embedded automotive mechatronics system modelling on a high abstraction level. The approach described focusses on the model-based structural description of the E/E system under development. Additionally, the signals and interfaces are an essential part of modelling.

The existing SysML-based design method (see also [14]) is extended by the newly developed *Embedded Mechatronics System Domain-Specific Modeling (EMS-DMS)* for automotive embedded system architectural design. It is not intended to replace the SysML-based solution created so far. Instead, the EMS-DSM is integrated into existing methods. Hence, the whole tool-chain, starting from the SysML-based system architectural design tool and finishing at software / hardware architectural design, can be utilized if desired. An overview of the tool integration is shown in Figure 1.

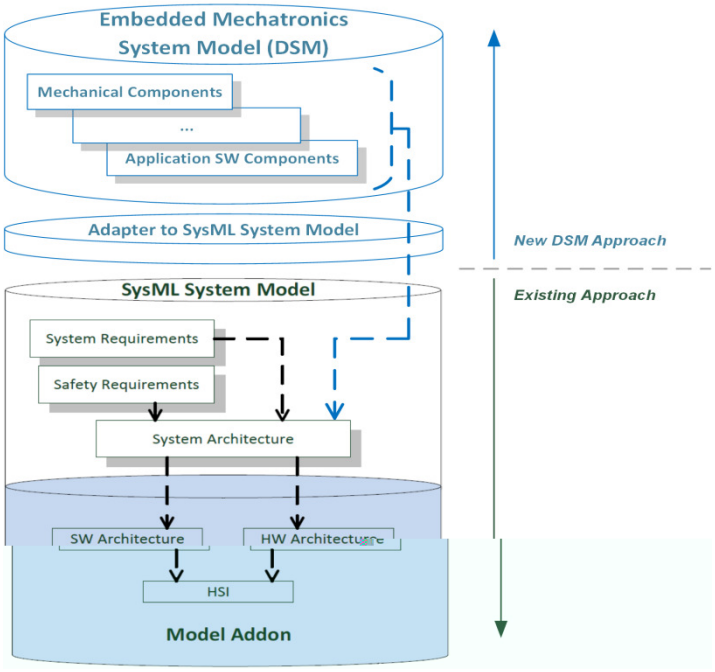

Figure 1. Tool-Chain Integration of DSM and SysML Model Approach (based on [16])

The definition of the newly developed model-based domain-specific language is shown in Figure 2. The *EMS-DSM Component* is the origin of all other classes regarding language definition. The six attributes of this class are

- *ID* - unique identifier of the particular instance in the architectural design model, set automatically.
- *Name* - name or short description of the particular instance, chosen by the design engineer.
- *Mask* - graphical representation of the particular instance, set by the engineer responsible for the design tool.
- *Requirement* - in this approach, a link to the Redmine requirements database is set by the designer.
- *Verification Criteria* - similar to Requirement, a link to the Redmine verification criteria artifact is set by the designer.
- *Specification* - link to further information about the actual component, e.g. a CAD drawing or a data sheet.

The *EMS-DSM Component* serves as the base node of the EMS-DSM definition, and declares the common attributes of the derived classes at the lower levels. Therefore, this component is not instanced for the design process. At the next language definition level, the following component classes are available:

- *Mechanical Components* - used by all mechanical, domain-specific components, e.g. the *Mechanical Pressure Regulator* class in the use-case shown in Section 4.
- *Compartment Components* - gives the opportunity to specify areas or compartments, where mechanical and hardware components are installed.
- *E/E Item Components* - an abstract component class definition, which serves as a basis for the hardware and software components at the lower levels. Additionally, the property *ASIL*, corresponding to the ISO 26262, is stated.

The majority of the non-abstract component classes are derived from the hardware component class:

- *Sensor Component* - used for all domain-specific sensor components.
- *Control Unit Component* - used for all domain-specific control unit components.
- *Actuator Component* - used for all domain-specific actuator components.
- *External Control Unit Component* - special class, to make signals from an external system available in the considered system.

All hardware components and their instances in the system design model, with the exception of the *External Control Unit Component*, are capable of containing a software design model. This means that any kind of software component instance is only allowed to be implemented in a software design model which belongs to an instance of a hardware component. This special language characteristic is defined by the *Aggregation* relationship between hardware and software components, which also implies the hardware-software interface.

The last part of the EMS-DSM definition description is related to the classes (derived from the software component):

- *Basis Software Component* - used for all low-level, hardware-dependent software components.
- *Application Software Component* - used for all functional software components.

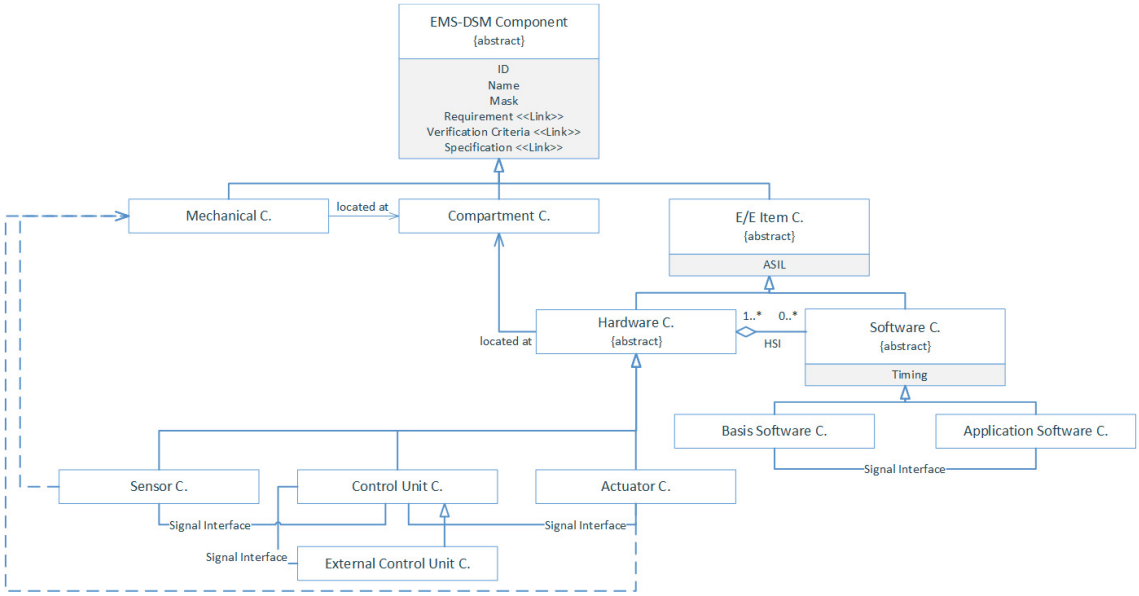

Figure 2. EMS-DSM Language Definition

As mentioned in Section 2, a more detailed description of the domain-specific modelling language can be found in [15].

3.2. Influence of Process Reference Model on HSI Specification

Due to their broad dissemination in the automotive sector, the two most important reference models are Automotive SPICE [4] and CMMI [17]. Both pursue similar targets: they (a) determine the process capability/maturity, and (b) aspire a continuous process improvement in the particular development team and/or company. The reference models do not exist in order to specify how processes have to be implemented. Instead, desired process outcomes (Automotive SPICE) or goals (CMMI) are defined and described in more detail by best practice characterisation (base or generic practices at Automotive SPICE, and specific or generic practices at CMMI). The *Automotive S(oftware) P(rocess) I(mprovement) and C(apability) (D)e(termination)* reference model is based on the international standard ISO 15504 and is

primarily used in Europe, as well as in some parts of Eastern Asia. The latest version, which was analysed for this approach, is 3.0 and was released in July 2015. The *C(apability) M(aturity) M(odel) I(ntegration)* reference model has been developed by the Software Engineering Institute (SEI) at Carnegie Mellon University. CMMIs currently exist for *Acquisition*, *Development*, and *Services*. As CMMI is not widespread in the European automotive sector, the remaining part of this section will focus on Automotive SPICE as the relevant process assessment and reference model. The model does not address the demand for a hardware-software interface directly, but some guidance on HSI specification can be extracted from general interface topics.

Table 1 lists the elements of the Automotive SPICE reference model that provide information about interfaces between system components. As expected, interface work products are needed for *Architectural Design* and the *Integration* topics. In addition to the *Process ID* and the *Process Name*, the corresponding *Base Practice IDs* are indicated. These give more detailed information on what the outcome should look like. In SYS.3.BP3, the definition (*identify*, *develop*, and *document*) of system element interfaces is stipulated. This equally applies to the hardware-software interface. In SYS.3.BP4, a description of the dynamic behaviour of and between the system elements is provided. The possible operating modes of the system, which determine the dynamic behaviour, have to be taken into account in the HSI definition. Base Practice SYS.4.BP3 postulates that the interfaces between system items have to be covered by the system integration test to show consistency between the real interfaces and the architectural design. With regard to the HSI, SWE.2.BP3 and SWE.2.BP4 can be interpreted in a similar way to their system level counterparts (SYS.3.BP3, SYS.3.BP4). SWE.2.BP5 claims the determination and documentation of the resource consumption objectives of all relevant software architectural design elements. To support this using the hardware-software interface definition, information on resource consumption shall be included in the description of the signals, wherever applicable. An interface definition is also demanded at process *SWE.3 - Software Detailed Design and Unit Construction*. However, in this case, the specification belongs to the signals communicated between the components on the lowest (most detailed) software level. Hence, this communication specification does not directly belong to the hardware-software interface, and will not be taken into consideration in this approach. The last process/base practice in Table 1 is SWE.5.BP3. It demands a description of the interaction between relevant software units and their dynamic behaviour. Again, this base practice can be interpreted in a similar way to its system level counterpart (SYS.4.BP3).

Table 1. HSI Accompanying Automotive SPICE Processes.

Process ID	**Process Name**	**Base Practice ID**
SYS.3	System Architectural Design	BP3, BP4
SYS.4	System Integration and Integration Test	BP3
SWE.2	Software Architectural Design	BP3, BP4, BP5
SWE.5	Software Integration and Integration Test	BP3

In the Automotive SPICE reference model, *Output Work Products* are also defined and linked to the base practices previously stated. From this contribution's point of view, the relevant work products are:

- *System Architectural Design* - the main aspects to consider regarding the HSI are *memory/capacity requirements*, *hardware interface requirements*, *security/data*

protection characteristics, *system parameter settings*, *system components operation modes*, and the influence of the *system's and system component's dynamic behaviour*.

- *Interface Requirement Specification* - the main aspects to consider regarding the HSI are *definition of critical timing dependencies or sequence ordering* and *physical interface definitions*.

3.3. Influence of Automotive Functional Safety on HSI Specification

The international standard *ISO 26262* for *Functional Safety* in the automotive electrical and/or electronic system domain was released in 2011. Since then, many best practice articles and books have been published on how to develop according to the standard. However, with the exception of the safety-critical view, the hardware-software interface has rarely been highlighted in these publications.

According to ISO 26262, the HSI is to be specified during the phase *Product Development at the System Level* (see Figure 3), which is described in Part 4 of the standard. As a prerequisite for specifying the hardware-software interface, a system design has to be established. While preparing the system architectural design, the technical safety and non-safety requirements are allocated to the hardware and software. Subsequent to this allocation, an initial interface description can be prepared. The HSI shall be continuously refined in the ensuing hardware and software product development phases, which are described in Parts 5 & 6 of the ISO 26262.

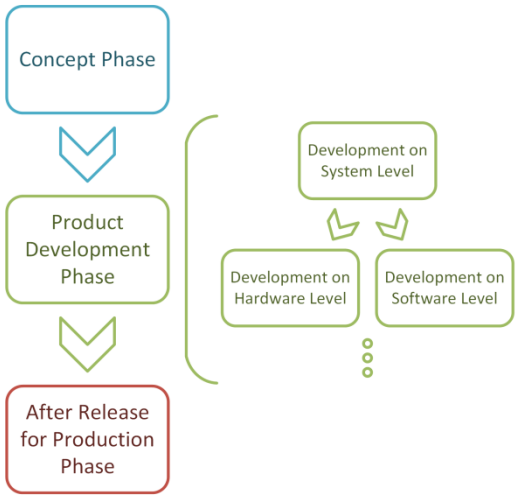

Figure 3. Development Phases According to [3]

The majority of information concerning how to specify the interface aligned to functional safety can be found in Clause 7.4.6 of Part 4 of the standard. In our approach, most of the HSI characteristics demanded by this clause, such as *operation modes of the hardware device* and *shared/exclusive use of the hardware resource*, are described in the *Detailed Hardware Specification (DHS)* documents, which are linked to the main HSI document. A detailed description of the various development artifacts and their relationships is presented in Subsection 3.4. Additionally, the informative *Annex B* of Part 4 of ISO 26262 provides information concerning the possible content of the interface definition.

3.4. Incorporated Hardware-Software Interface Specification

Two main objectives have to be achieved when developing a new HSI specification approach:

1. identification, development and documentation of the essential HSI specification attributes & characteristics, and

2. support for the linking of related information to ensure full traceability.

The principle of the hardware-software interface specification approach described here is based on three origins, two of which have been described in the previous subsections:

a. the process reference and assessment model *Automotive SPICE*,

b. the automotive functional safety standard *ISO 26262*, and

c. the industrial experience of authors in past automotive E/E system development projects.

It is important to note that the hardware-software interface specification does not only consist of a single spreadsheet with a description of all signals between hardware and software. Further information belonging to the HSI specification can also be found in various development artifacts. Figure 4 shows the different aspects of our HSI specification approach:

- *Hardware-Software Interface Signal List* - spreadsheet with data of all signals between hardware and software. The attributes describing each signal have been derived from sources (a) - (c), which were mentioned at the beginning of this subsection.

- *Resource Consumption Objectives* - depending on the particular project, the objectives are described in spreadsheet(s) and/or free text document(s). Regardless of the type, the documents are linked to the software components in software architectural design (see attribute *Specification <<Link>> Software Component* class in the EMS-DSM language definition in Figure 2).

- *Detailed Hardware Specification* - depending on the particular project, the objectives are described in spreadsheet(s) and/or free text document(s). Regardless of the type, the documents are linked to the hardware components in system architectural design (see attribute *Specification <<Link>> Hardware Component* class in the EMS-DSM language definition in Figure 2).

- *Model-based Architectural Design* - this item represents the central source of information. The defined domain-specific modelling language facilitates the creation of the system and the software architectural design within the same design environment and allows the linking of all other relevant development artifacts. From a HSI specification perspective, the three previous items in this list are the most important development artifacts to be linked to the architectural design models.

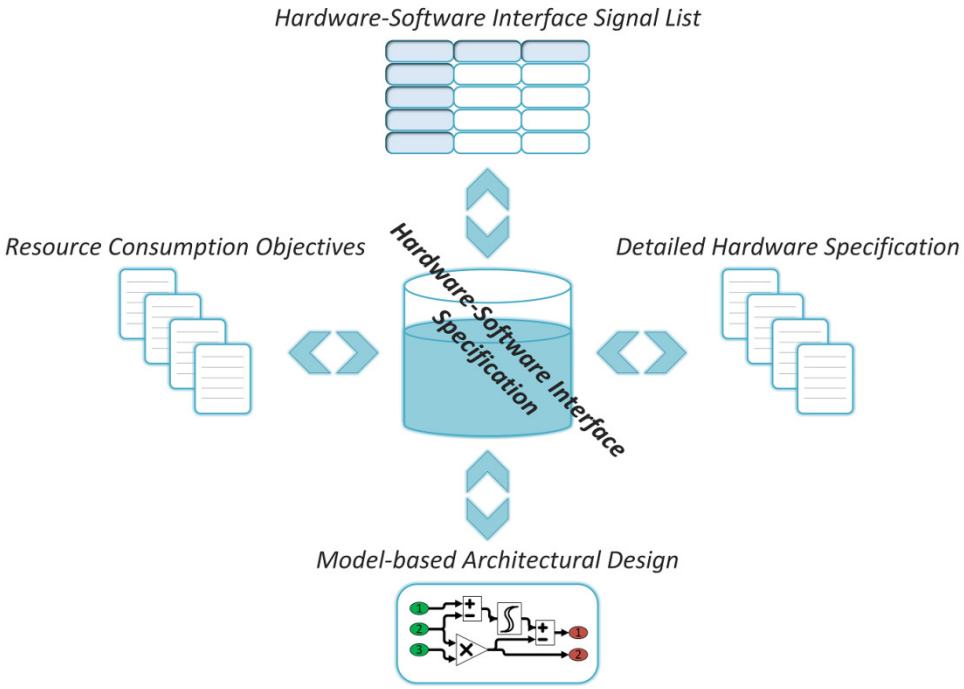

Figure 4. Distributed Hardware-Software Interface Specification

Establishing full traceability between the *Resource Consumption Objectives*, the *Detailed Hardware Specification*, and the *Model-based Architectural Design* is an easy task, accomplished by linking the related documents in the architectural design.

The integration of the *Hardware-Software Interface Signal List* data into the design model is more technically challenging. In [14] the authors described the functionality of the *HSI Definition Exporter and Importer*, which was developed to achieve a seamless transformation of the HSI representation between the SysML-based architectural design and the spreadsheet tool. The *HSI Definition Exporter* is an extension (*dynamic link library*) for the model-based development (MBD) tool, which is written in C# and allows the modelled HSI to be exported to a spreadsheet document (either in *csv* or *xls* format). The *HSI Definition Importer* is the counterpart of the *HSI Definition Exporter*, which is also implemented as a *dynamic link library* using the spreadsheet tool's API. It allows the import of all HSI information from the spreadsheet document or a selective update of the HSI model artifacts. Using both the export and import functionality leads to a round-trip engineering capability regarding the HSI signal list and the HSI signals modelled in the architectural design. In this approach, the libraries of the exporter and importer extensions are slightly adapted to the needs of the domain-specific modelling language.

To conclude the description of our approach, the HSI signal attributes and their origins are listed in Table 2.

Table 2. HSI Signal List Attributes.

Attribute	Comments	Origin
Signal Direction	Input or Output, out of the controllers view	Author's Experience
Signal Description	A short signal description or the signals name	ISO 26262-4 (Annex B)
Sensor / Actuator	Type or identifier of signals source/sink	Author's Experience
Supply Voltage	-	Author's Experience
Physical Min Value	-	ASPICE SYS.4.BP3
Physical Max Value	-	ASPICE SYS.4.BP3
Accuracy	In % of range of values	ISO 26262-4 (Annex B)
Physical Unit	E.g. V, A, ...	ISO 26262-4 (Annex B) ASPICE SYS.4.BP3
HW Interface Type	E.g. Digital In, Analog Out, CAN, ...	ISO 26262-4 (Annex B) ASPICE WP 17-08
HW Pin #	Pin number or identifier at e.g. ECU	ISO 26262-4 (Annex B)
Message ID	In case of bus communication	Author's Experience
Start Bit		
Internal Cycle Time	E.g. 10 ms	ISO 26262-4 (Section 7.4.6) ASPICE SYS.4.BP3, SWE.5.BP3, WP 17-08
External Cycle Time	Only applicable for digital signals	Author's Experience
HW Timer / Interrupt / Watchdog	Identifier of triggered e.g. interrupt	ISO 26262-4 (Section 7.4.6)
Operating Modes	Information if signal is needed special operating modes (e.g. start up, calibration, ...)	ISO 26262-4 (Annex B) ASPICE SYS.3.BP4, SWE.2.BP4, WP 04-06
HW Diagnostic Feature	E.g. short circuit detection, ...	ISO 26262-4 (Section 7.4.6)
Memory Type	E.g. RAM, EEPROM, ...	ISO 26262-4 (Annex B)
Security/Data Protection	Information on special security issues	ASPICE WP 04-06
Critical Timing Dependencies or Sequence Ordering	-	ASPICE WP 17-08
Signal Name @ SW	Identifier of signal as used in application software	Author's Experience
Initial Value	-	Author's Experience
Data Type	E.g. UInt16, Float, ...	ASPICE SYS.4.BP3, SWE.5.BP3
Scaling LSB	Scaling information in case of fixed-point arithmetic	ASPICE SYS.4.BP3, SWE.5.BP3
Scaling Offset		
Min Value @ SW	-	ASPICE SWE.5.BP3
Max Value @ SW	-	ASPICE SWE.5.BP3
Accuracy @ SW	In % of range of values	ISO 26262-4 (Annex B)
Physical Unit @ SW	E.g. km/h, Nm, ...	ASPICE SWE.5.BP3
Default Value @ SW	Default value in case of an invalid input signal	Author's Experience
Detection Time	Time until a fault is diagnosed	ISO 26262-4 (Section 7.4.6)
Reaction Time	Admissible reaction time after a fault was detected	ISO 26262-4 (Section 7.4.6)
ASIL	Automotive Safety Integrity Level classified A - D, or QM if no safety-relevance is given	ISO 26262-4 (Annex B)
Signal ID	Identifiers required for the support of the domain-specific modelling approach	Author's Experience
HW Device ID		

4. APPLICATION

In this section, the HSI specification approach is applied to the development of an automotive fuel tank system for compressed natural gas (CNG). For an appropriate scale of the showcase, only a small part of the real-world system is utilized. The application should be seen as illustrative material, reduced for internal training purposes for students. Therefore, the disclosed and commercially non-sensitivity use-case is not intended to be exhaustive or representative of leading-edge technology. Before the showcase is illustrated, tool support regarding both domain-specific modelling and requirements management shall be explained briefly.

4.1. EMS-DSM Language Tool Support

Generally speaking, the EMS-DSM language can be supported by various tools, but at the time when the research project was initiated, the highest possible flexibility was desired, as was full access to the tool's source code. To avoid developing an application from scratch, the open source project *WPF Diagram Designer* (see [18]) was chosen as a basis for tool development. The corresponding documentation has about 540,000 views and the source code has been downloaded more than 24,000 times. Therefore, the source, which provides standard functionality such as file handling and basic graphical modelling, is well reviewed. The source code is written in C# and provides good expandability. New functionalities have been implemented for the diagram designer, named *EASy-Design* (**E**mbedded **A**utomotive **Sy**stem-**Design**), to facilitate engineering with EMS-DSM models. However, EASy-Design is just one possibility for EMS-DSM tool support. The methodology and its C# implementation can be ported to e.g. Enterprise Architect[2] by the provided *Add-in* mechanism. Another alternative is the Eclipse[3] framework, or rather the Eclipse-based project *Sirius*, which enables the creation of a graphical modelling workbench, by facilitating the Eclipse modelling technologies without writing code.

4.2. Project and Requirements – Management Tool Support

The web-based open source application *Redmine*[4] is used for topics such as project management and requirements management in this approach. Owing to its high flexibility through configuration, new trackers have been added for development according to the de facto standard Automotive SPICE [4]. The process reference model already mentioned in Section 3 defines three different types of requirements of the engineering process group: *Customer Requirements*, *System Requirements*, and *Software Requirements*. The hardware focus is missing from the embedded E/E system view. Additionally, requirements and design items for mechanical components have to be introduced for the design of an embedded mechatronics-based E/E system. Similar to the Automotive SPICE methodology on a system and software level, engineering processes have been defined for these missing artifacts. To sum up, the available requirement and test case types for this approach are: *Customer Req*, *System Req*, *System TC*, *System Integration TC*, *Software Req*, *Software TC*, *Software Integration TC*, *Hardware Req*, *Hardware TC*, *Mechanics Req*, and *Mechanics TC*.

The test case and requirement items are connected to each other by their unique identifier. For a safety-critical development according to ISO 26262, additional issue types such as *Functional*

[2] http://www.sparxsystems.com/
[3] http://eclipse.org/
[4] http://www.redmine.org/

Safety Requirements have been added. By reconfiguring the project management tool Redmine, all requirement types mentioned have been implemented.

4.3. CNG Tank System Showcase

Figure 5 illustrates the EMS-DSM tool *EASy-Design* with the system architectural design model of the simplified showcase. The CNG fuel tank system consists of seven mechanical components, which are blue coloured (Tank Cylinder, Filter, etc.) The medium flow between mechanical components, which is CNG in this case, is displayed by blue lines with an arrow at the end. Furthermore, five hardware components are placed at the system design model level, which are yellow coloured (In-Tank Temperature Sensor, Tank ECU, etc.) The signal flow between the components is displayed using yellow lines ending with an arrow. A communication bus is inserted between the *Control Unit* and the *External Control Unit* component, shown by the double compound line type and arrows at both ends.

By selecting a model element and clicking the button *Link Requirements*, the elements requirements dialogue is opened and a link between the selected element and an item from the requirements database (e.g. *System Requirement*, see Subsection 4.2) can be established. Already linked requirements from Redmine's MySQL database are listed with their ID, Type, Title, ASIL, and Core functionality attribute. With a click on *Link Specifications*, various documents, such as detailed hardware specifications and datasheets, can be linked to the selected model element.

The *Hardware-Software Interface Specification* emphasis of this contribution is also supported by EASy-Design. Again, a hardware element of the model has to be selected and can be defined with a subsequent click on the button *Edit Hardware-Software Interface* in the *Element Properties* group, the interface of the selected hardware item. In Figure 5, the Tank ECU has been selected and in Figure 6, the newly opened HSI definition dialogue for the Tank ECU is illustrated. Within this dialogue, all operations needed to add, modify or delete signals can be triggered by clicking the relevant button:

- *Add New Signal* - a new dialogue window is opened and a signal can be created by entering the properties described in Table 2 (see Figure 7).
- *Add Connected Signal* - the hardware elements in the architectural system design can be connected by (yellow) lines as described in Subsection 4.1. Every output signal from any connected hardware element can be added as an input signal in the HSI signal definition in the actual hardware element.
- *Modify Signal* - at the HSI signal definition main dialogue (illustrated in Figure 6), a signal has to be selected, for which the modification dialogue will be opened after a click on *Modify Signal*. The signal modification dialogue is similar to the *Add New Signal* dialogue.
- *Import Signal(s)* - the *HSI Definition Importer*, as described in Subsection 3.4, is selected, and signals from a HSI signal definition stored in spreadsheet format can be added to the system architectural design model.
- *Export Signal(s)* - the *HSI Definition Exporter*, as described in Subsection 3.4, is selected and signals from the HSI signal definition in the system architectural design model can be exported to a HSI signal definition in spreadsheet format.
- *Delete Signal(s)* - the signals have to be selected from the main HSI signal definition dialogue and are removed from the interface when the button is clicked.

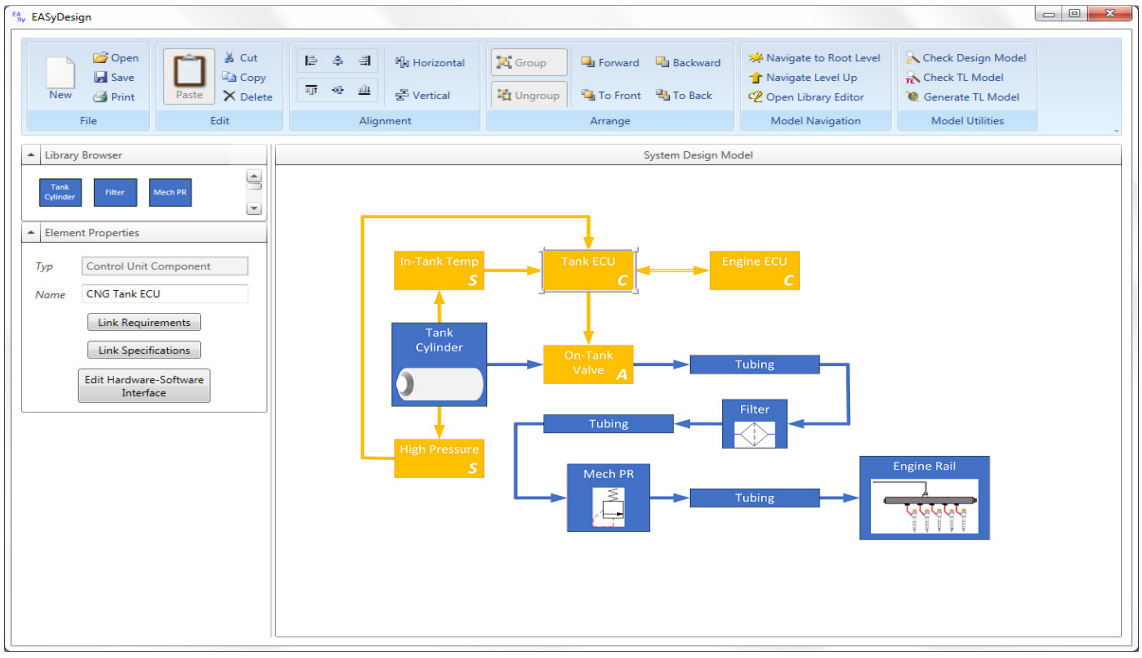

Figure 5. Self-developed tool *EASy Design* with a Simplified CNG Tank System Architectural Design

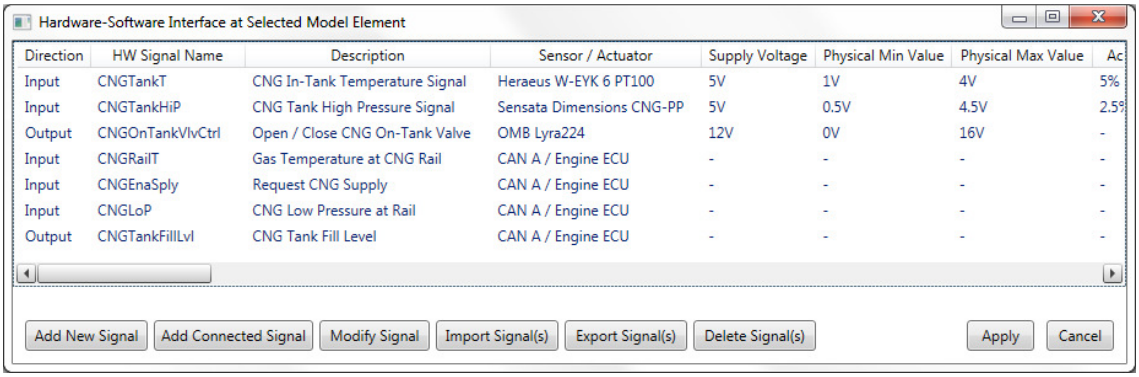

Figure 6. Hardware-Software Interface Dialog at *EASy Design*

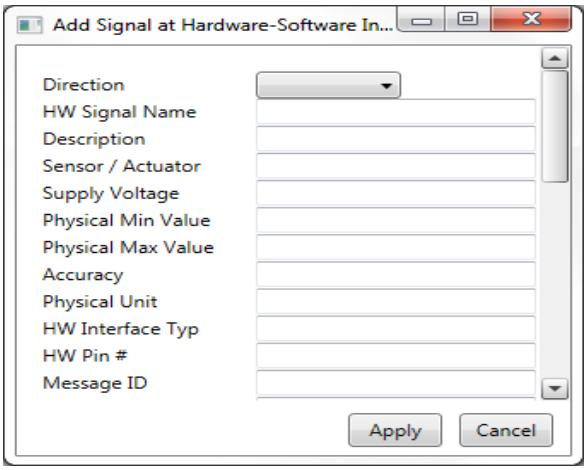

Figure 7. Hardware-Software Interface Add New Signal Dialog at *EASy Design*

As can be seen in Figure 5, no *Software Components* are modelled at this level (System Design Model). With a double-click on a *Hardware Component* (e.g. *Tank ECU*), the next modelling level is opened (named *E/E Item Design Level*). The (green coloured) *Basis Software Components* and *Application Software Components* can be placed here. At each basis software component, the input and output signals from the HSI definition in the particular hardware component can be used and therefore connected to the software.

5. CONCLUSION

Previous sections described the factors influencing the development of our hardware-software interface specification approach as well as the supporting tools. A domain-specific modelling method for the design of embedded automotive mechatronics-based E/E systems formed the basis for this work. This approach has the potential to bring together the different engineering disciplines involved in E/E system development by facilitating the HSI specification process. Additionally, many artifacts such as requirements, verification criteria, and various specifications can be linked to the models, created with the new, domain-specific modelling language. With the help of the linked artifacts, vital traceability can be established. Depending on the respective tool chain and the organisation's process landscape, the EMS-DSM models can also facilitate a single point of truth strategy.

First use case implementations show promising results. However, there are several features that still need to be implemented. Options for describing the system's behaviour, e.g. a kind of task scheduling definition, are to be introduced. Furthermore, the Model-to-Model-Transformer between the domain-specific and traditional SysML system architectural design model has to be extended to achieve an automatic transformation of the HSI signal definition between the different modelling strategies.

REFERENCES

[1] S. Friedenthal, A. Moore, and R. Steiner, "OMG Systems Modeling Language (OMG SysMLTM) Tutorial," in INCOSE International Symposium, 2006.

[2] J. Meseguer, "Why Formal Modeling Language Semantics Matters," in Model-Driven Engineering Languages and Systems, ser. Lecture Notes in Computer Science, J. Dingel, W. Schulte, I. Ramos, S. A. ao, and E. Insfran, Eds., vol. 17th International Conference, MODELS 2014, Valencia, Spain, no. 8767. Springer International Publishing Switzerland, 2014, keynote.

[3] "ISO 26262, Road vehicles - Functional safety," International Organization for Standardization, Geneva, CH, International Standard, November 2011.

[4] VDA QMC Working Group 13 / Automotive SIG, "Automotive SPICE Process Assessment / Reference Model," Tech. Rep. Revision ID: 470, July 2015, version 3.0.

[5] E. Andrianarison and J.-D. Piques, "SysML for embedded automotive Systems: a practical approach," in Conference on Embedded Real Time Software and Systems. IEEE, 2010.

[6] R. Boldt, "Modeling AUTOSAR systems with a UML/SysML profile," IBM Software Group, Tech. Rep., July 2009.

[7] H. Giese, S. Hildebrandt, and S. Neumann, "Model Synchronization at Work: Keeping SysML and AUTOSAR Models Consistent," LNCS 5765, pp. 555 –579, 2010.

[8] R. Kawahara, H. Nakamura, D. Dotan, A. Kirshin, T. Sakairi, S. Hirose, K. Ono, and H. Ishikawa, "Verification of embedded system's specification using collaborative simulation of SysML and simulink models," in International Conference on Model Based Systems Engineering (MBSE'09). IEEE, 2009, pp. 21–28.

[9] J. Meyer, "Eine durchgängige modellbasierte Entwicklungsmethodik für die automobile Steuergeräteentwicklung unter Einbeziehung des AUTOSAR Standards," Ph.D. dissertation, Universität Paderborn, Fakultät für Elektrotechnik, Informatik und Mathematik, Paderborn, Germany, July 2014.

[10] M. Broy, M. Feilkas, M. Herrmannsdoerfer, S. Merenda, and D. Ratiu, "Seamless Model-Based Development: From Isolated Tools to Integrated Model Engineering Environments," Proceedings of the IEEE, vol. 98, no. 4, pp. 526–545, 2010.

[11] M. Mernik, J. Heering, and A. M. Sloane, "When and how to develop domain-specific languages," ACM computing surveys (CSUR), vol. 37, no. 4, pp. 316–344, 2005.

[12] C. Preschern, N. Kajtazovic, and C. Kreiner, "Efficient development and reuse of domain-specific languages for automation systems," International Journal of Metadata, Semantics and Ontologies, vol. 9, no. 3, pp. 215–226, 2014.

[13] V. Vujovic, M. Maksimovic, and B. Perisic, "Sirius: A rapid development of DSM graphical editor," in 18th International Conference on Intelligent Engineering Systems (INES). IEEE, 2014, pp. 233–238.

[14] G. Macher, H. Sporer, E. Armengaud, E. Brenner, and C. Kreiner, "Using Model-based Development for ISO26262 aligned HSI Definition," in Critical Automotive applications: Robustness & Safety, ser. CARS@EDCC2015, Paris, France, 2015.

[15] H. Sporer, "A Model-Based Domain-Specific Language Approach for the Automotive E/E-System Design," in International Conference on Research in Adaptive and Convergent Systems (RACS 2015), ser. RACS '15, Prague, Czech Republic, 2015.

[16] G. Macher, E. Armengaud, and C. Kreiner, "Bridging Automotive Systems, Safety and Software Engineering by a Seamless Tool Chain," in 7th European Congress Embedded Real Time Software and Systems Proceedings, 2014, pp. 256–263.

[17] Software Engineering Institute, "CMMI for Development, Version 1.3," SEI, Carnegie Mellon, Tech. Rep. CMU/SEI-2010-TR-033, ESCTR-2010-033, November 2010.

[18] Code Project, "WPF Diagram Designer - Part 4," Online Resource, March 2008, http://www.codeproject.com/Articles/24681/WPFDiagram-Designer-Part, accessed Mar 2015.

AUTHORS

Harald Sporer received a MSc. degree in Telematics from Graz University of Technology. He worked as software development engineer on Hardware-in-the-Loop (HIL) systems at AVL List GmbH and as functional software developer for embedded automotive systems at Magna Powertrain AG & Co KG. Currently he is working on his PhD at the Institute of Technical Informatics at Graz University of Technology. Parallel to his PhD thesis he is also active in the field of embedded automotive system design, engineering process improvement, and functional safety engineering.

Georg Macher received a MSc. degree in Telematics and worked as software development engineer on prototype vehicles at AVL List GmbH. Currently he joined the R&D department of AVL's powertrain engineering branch and is working on his PhD at Institute for Technical Informatics at Graz University of Technology. Parallel to his PhD thesis is also active in the field of system, software, and functional safety engineering.

Dr. Christian Kreiner graduated and received a PhD degree in Electrical Engineering from Graz University of Technology in 1991 and 1999 respectively. 1999-2007 he served as head of the R&D department at Salomon Automation, Austria, focusing on software architecture, technologies, and processes for logistics software systems. He was in charge to establish a company-wide software product line development process and headed the product development team. During that time, he led and coordinated a long-term research programme together with the Institute for Technical Informatics of Graz University of Technology. There, he currently leads the Industrial Informatics and Model-based Architectures group. His research interests include systems and software engineering, software technology, and process improvement.

Prof. Dr. Eugen Brenner is Associate Professor at the Institute for Technical Informatics of the Graz University of Technology. He completed his master in Electrical Engineering 1983 in Graz. His PhD in Control Theory was finished 1987 also in Graz, dealing with optimal control in systems with limited actuating variables. He joined the institute in 1987, being the first scientific staff member. His post-doctoral lecture qualification in Process Automation was achieved in 1996.He has been member of the senate, of the curricula commission for Bachelor and Master-Programs, and Dean of Studies for Telematics. He currently is head of the Study Commission and Vice-Dean of Studies for Telematics. Eugen Brenner's primary research interests developed from FPGA-based hardware extension to parallel systems, real-time systems and process control systems. The most recent focus targeting embedded systems is on modelling, software-development, systems engineering and systems security, including agile programming methods and smart service engineering.