HAVEit

Highly automated vehicles for intelligent transport

7th Framework programme
ICT-2007.6.1
ICT for intelligent vehicles and mobility services
Grant agreement no.: 212154

The future of driving.

Deliverable D21.4

Bare platform tested and ready for the integration of partner applications

Version number	Version 1.0
Dissemination level	CO
Lead contractor	Continental Automotive GmbH
Due date	31.07.2010
Date of preparation	16.08.2010

Authors

Name	Company
Michael Gutknecht	USTUTT
Sergej Bahnmüller	USTUTT
Philipp Luithardt	USTUTT
Sergiy Prutyanyy	USTUTT

Project Managers

Alfred Hoess
Continental Automotive GmbH
Siemensstrasse 12
93055 Regensburg, Germany
Phone +49 941 790-5786
Telefax +49 941 790 99 5786
E-mail: Alfred.Hoess-EXT@continental-corporation.com

Holger Zeng
Continental Automotive GmbH
Siemensstrasse 12
93055 Regensburg, Germany
Phone +49 941 790 92330
Fax. +49 941 790 99 92330
E-mail: holger.zeng@continental-corporation.com

Project Co-ordinator

Reiner Hoeger
Continental Automotive GmbH
Siemensstrasse 12
93055 Regensburg, Germany
Phone +49 941 790 3673
Fax +49 941 790 99 3673
E-mail reiner.hoeger@continental-corporation.com

Revision and history chart

Version	Date	Reason
0.1	2010/07/05	Initial document
0.2	2010/07/13	Internal upgrade
0.3	2010/07/16	Version for review
1.0	2010/08/05	Integration of reviewer comments
	2010/08/16	Final editing and submission (consortium confidential version)
	2011/10/06	Generation of public deliverable version

Table of Contents

Executive summary

The main objective of the HAVEit project deals with the improvement of safety in road traffic. Therefore, three measures were defined, in order to address that topic: the development of next generation advanced driver assistance systems (highly automated vehicle applications), optimal task repartition between driver and virtual co-pilot system, and the development of failure tolerant, safe vehicle architectures. Especially related to the latter, two safe by-wire-systems are intended to be developed by the HAVEit partners: a steer-by-wire system as well as a brake-by-wire system.

University of Stuttgart's (USTUTT) challenging task is – beside others – to develop a fail-operational computational platform core for both by-wire-systems mentioned above, ensuring the required safety for drivetrain components to adhere to both of those systems. The overall development task was realised threefold: the development and commissioning of the required hardware components of the platform cores, the development of a generic software package, which manages the overall platform behaviour, and the development of a suitable configuration tool, allowing the platform core to be customised to the specific target system as needed.

In January 2010, USTUTT provided the so called generic platform cores. In addition to the hardware environment, the generic platform cores mainly consisted of the configurable redundancy management software-package as well as the suitable configuration tool, allowing the redundancy management – which is executed within the ECUs of the platform core – to be easily configured in a reproducible way.

Since then, USTUTT worked together with its partners to implement the by-wire-systems' specific configurations utilising the above mentioned tool framework. This effort resulted in platform cores configured for the specific vehicles, but still without the partners' applications – the so called "bare platform cores" which are topic of the present deliverable.

Having generated the vehicle specific configurations the bare platform cores were started running. Applying USTUTT's test environment including powerful test benches, the configured platform cores were proved for proper operation. By successfully finishing the testing period, the development of the bare platform cores was successfully finalised.

In the upcoming periods, the so far bare platform cores will be upgraded to their full functionality by implementation of the partner's system-specific control law applications. After a further testing period in the laboratory the resulting platform cores will be finally implemented into the HAVEit vehicles of WP4100 and 4200.

1 Introduction

In the beginning of 2010, USTUTT presented its generic fail-operational platform core demonstrator (see [1]). This is a safe computational platform core, which consists on one side of two duplex ECUs (so called »XCCs«) which communicate via two independent FlexRay buses. On the other side, the generic platform core includes a configurable software package called »Redundancy Management«, whose major purpose is to control the overall platform behaviour and to perform all management tasks necessary when using redundant modules.

In addition to the generic platform core, USTUTT developed a fitting Meta-Tool framework, which supports the configuration task of the above mentioned redundancy management. In consequence, the complete platform core can be “easily” configured to whatever system is intended to be realised (of course within some given restraints).

Within the HAVEit project, the USTUTT fail-operational platform core is intended to be utilised for the realisation of the drivetrain control within the vehicles of WP 4100 (steer-by-wire) and WP 4200 (brake-by-wire) (see figure 1). Here, so called 2E architectures shall be implemented, meaning in principle a full electrical replication of today’s conventional architectures, which use to comprise at least one mechanical fallback link (1E/1M).

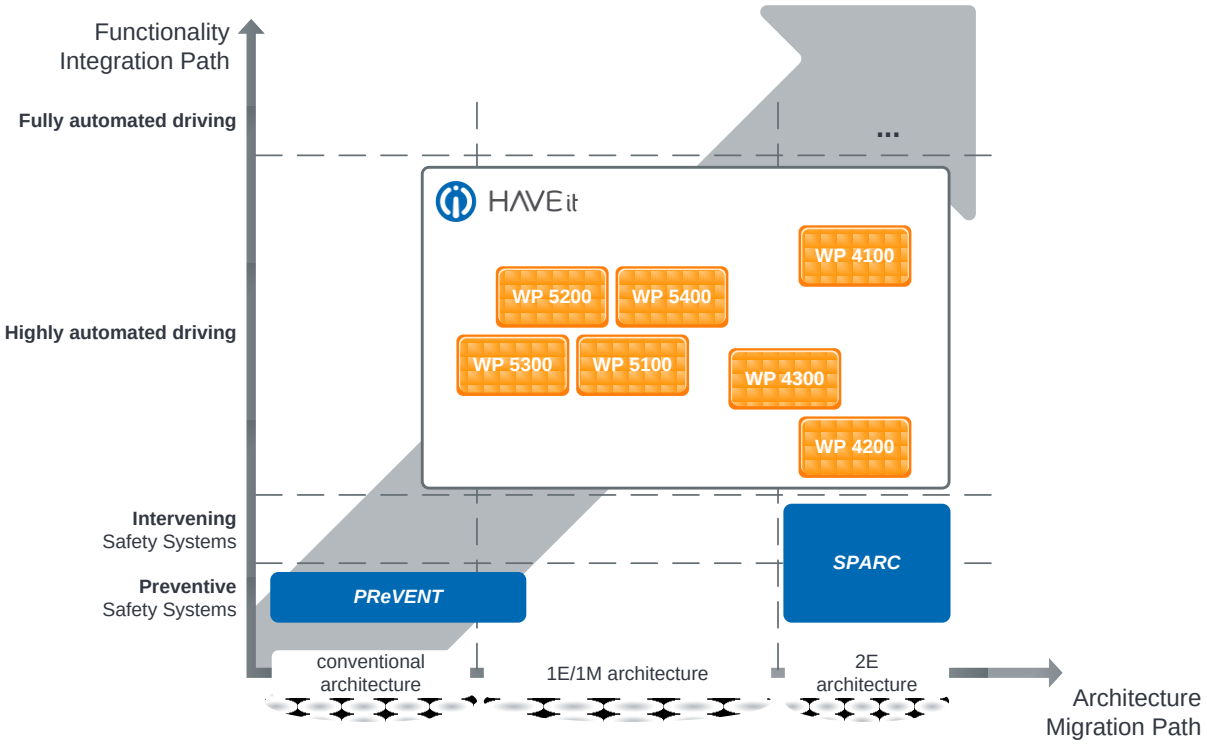

Figure 1: Architecture Overview

The present document is intended to give a brief overview about the process and specific tasks which had to be performed in order to configure and commission the generic platform cores for the steer-by-wire system of WP 4100 as well as the brake-by-wire system of WP 4200. Note shall be given that the results of this progress are platform cores, each confi-

gured for the specific vehicle, but still without the partners' applications (see [1]). That stage of platform is used to be called »bare platform core«.

The document is structured as follows: After a short introduction of the process to define the particular systems' dataflow principally in cooperation with our partners DLR, HALDEX and EXPLINOVO (chapter 2), modelling of the dataflow-specific issues within the Meta-Tool is described in chapter 3 in more detail. Subsequently, derivation of the final configuration settings from the dataflow models is described in chapter 4. Configuration of the more platform core related software modules is enlightened in chapter 5 in a nutshell, followed by a short description of the tests being performed in order to prove the resulting bare platform cores for proper operation (chapter 6). The whole deliverable is completed by a brief conclusion and outlook in chapter 7.

Please take into regard: The present document is a prototype deliverable. In conjunction with the extraordinary amount of data, which was generated in the context of configuration, only excerpts of the outcomes could be given exemplarily within this document.

2 Communication Matrices

Communication matrices were introduced as an appropriate means for definition, collection and management of the complete dataflow information about the two HAVEit by-wire-systems. Excel based spreadsheets were used for this purpose. Those were exchanged and filled by all partners being involved in the particular system development in order to generate a common basis for further processing. Therewith, the communication matrices constitute the very groundwork for configuration of the related platform cores, too.

The content of the communication matrices is two-fold. On one hand, the matrices contain information about the signal dataflow between aggregates and XCCs. This usually includes the definition of the signals which each particular aggregate provides, resp. requests for, definition of the FlexRay frames being exchanged between aggregates and XCCs, or signal packing within those FlexRay frames. Summarised, the responsible partners had to provide the following information about each signal:

- Source
- Sink
- Communication path
- Size
- Attachment to a particular FlexRay frame
- Placement of each signal within its attached FlexRay frame.

The final FlexRay frame slot numbering was defined by USTUTT in a later step, taking into regard the particular service timing for each system. For instance, care shall be given, that transmission of FlexRay frames by the XCCs doesn't start before the particular control law application finalised computation of its output commands.

On the other hand, the communication matrices itemise information which address the specific dataflow within the redundancy management software. Beside others, these data contain information about

- Linking between signals of the aggregates and signal ports of the control law application,
- Detailed information about the signal properties like data-type or physical meaning,
- Very detailed information about each signal, which are beside others used within the redundancy management to determine, whether a signal is faulty or not (e.g. failure codes or the like).

Having completed the communication matrices together with our partners, a very important milestone in configuring the vehicle specific platform cores was achieved.

3 Modelling of the Dataflow

3.1 Overview

As already mentioned in the introduction of this document, USTUTT developed a Meta-Tool framework for configuration of the redundancy management software and therewith for the overall platform core. One major chain link within the Meta-Tool is the GME environment, used for input and modelling of the dataflow (see left side of figure 2).

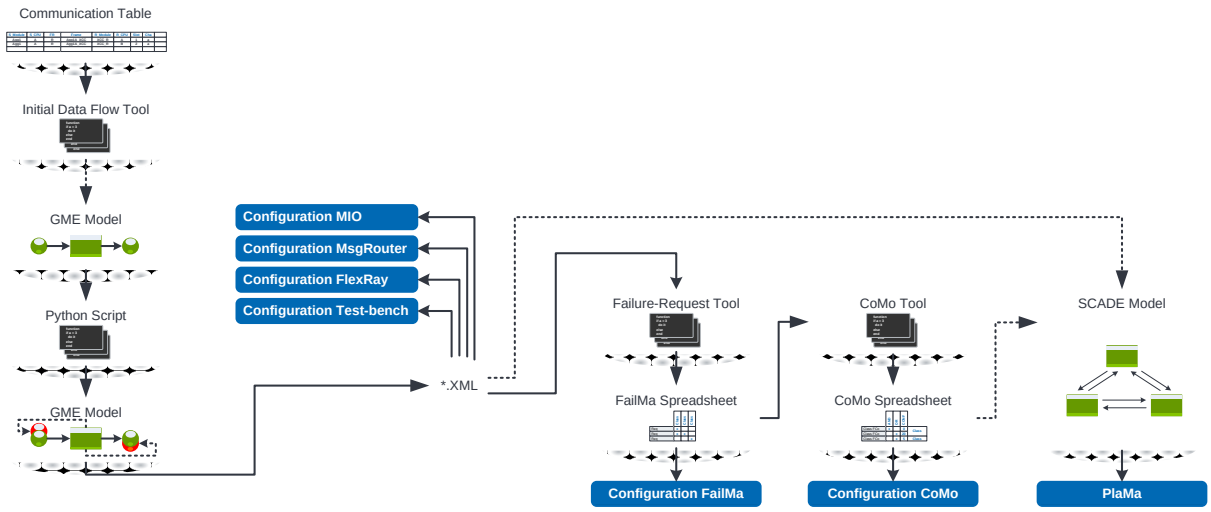

Figure 2: Meta-Tool Chain - Overview

In general, the generic modelling environment (GME) is a free available software tool, allowing creation of models based on pre-defined rules (so called paradigms). By defining the meta-model (including the paradigms) and taking advantage of the COM interface of GME, an appropriate tool was found, allowing comfortable modelling by the user on one side, and extensive, rule-based manipulation of models on the other side.

The overall modelling in GME is split up in the following three layers within the Meta-Tooling:

- System Layer
- Container Layer
- Software Layer

The particular layers, respectively their content is described in the following subsections.

3.2 System Layer

The system layer is the very top layer within the GME modelling. It mainly contains the definition of the platform topology on one side, and the definition of the applications on the other side. Since the system layer is the one layer, which needs to be modelled by the user “manually”, our intention was to keep the required input information as small as possible.

Therefore we promoted the demarcation between

- applications, which include all information about the signals being provided, resp. required by the dedicated functionality and
- topology, which mainly lists the available modules, their bus interfaces and bus connections (see figure 3).

Having modelled the topology on one side and the applications on the other side, the latter have to be mapped to whatever module they are intended to be executed on.

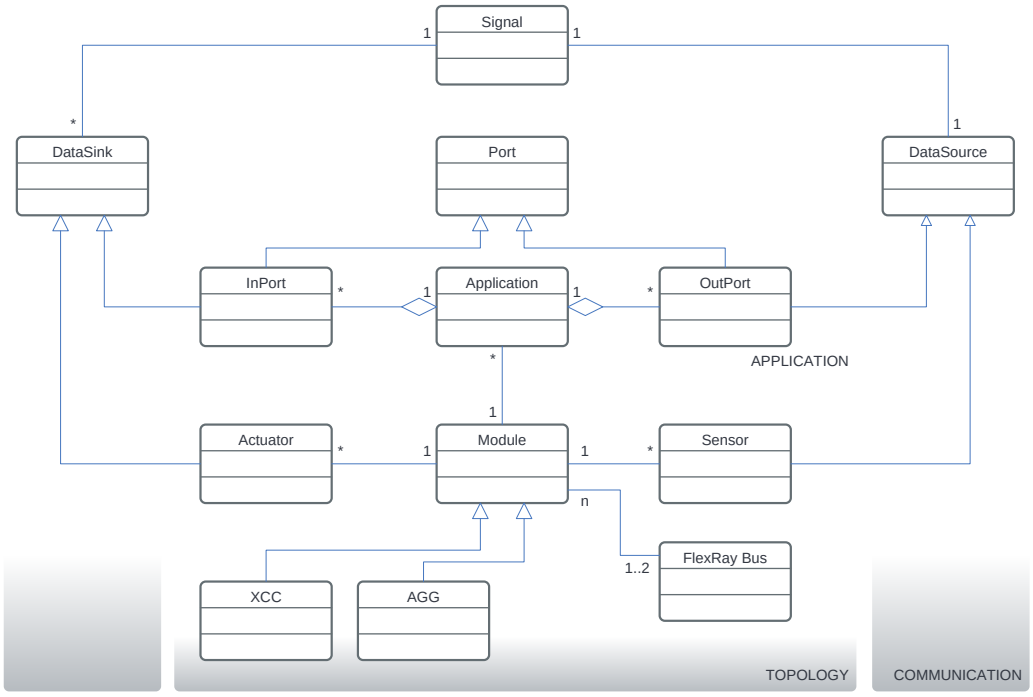

Figure 3: GME System Layer Input

Although we intended to simplify the manual effort of the user as much as possible, modelling of numerous signals within GME (e.g. up to 500 signals for the BbW-vehicle) – each of them including an amount of properties itself – turned out to be a very exhaustive and error prone task. Therefore, we preceded the GME tool with a further chain link, which expands most of the required information out of the communication matrices mentioned in chapter 2 and generate a first basic GME model setup for the applications.

In the following sections, the results of modelling the GME system layer are shown in excerpts for the steer-by-wire-system as well as the brake-by-wire-system.

3.2.1 Steer-by-Wire

Within this section, excerpts of the system layer modelling of the steer-by-wire system of WP 4100 are shown exemplarily in figures 4-8. Although the modelling method for the system layer was chosen to keep the model as simple as possible (i.e. no explicit modelling of redundant signals), the pure amount of signals itself made the models quite large anyhow (see table 1). Therefore, only excerpts of the particular models could be given here, mostly in order to provide an impression about the modelling task.

Table 1: Number of Input- / Output-Signals per Application – SbW

Model	Number of Input Signals*)	Number of Output Signals*)
Application »Clutch«	7	8
Application »Driver Feedback Actuator (DFA)«	12	17
Application »Joint System«	13	18
Application »Vehicle«	9	16
Application »SBW«	89	39

**) Please note: Not all signals being presented in table 1 are visible within the following figures!*

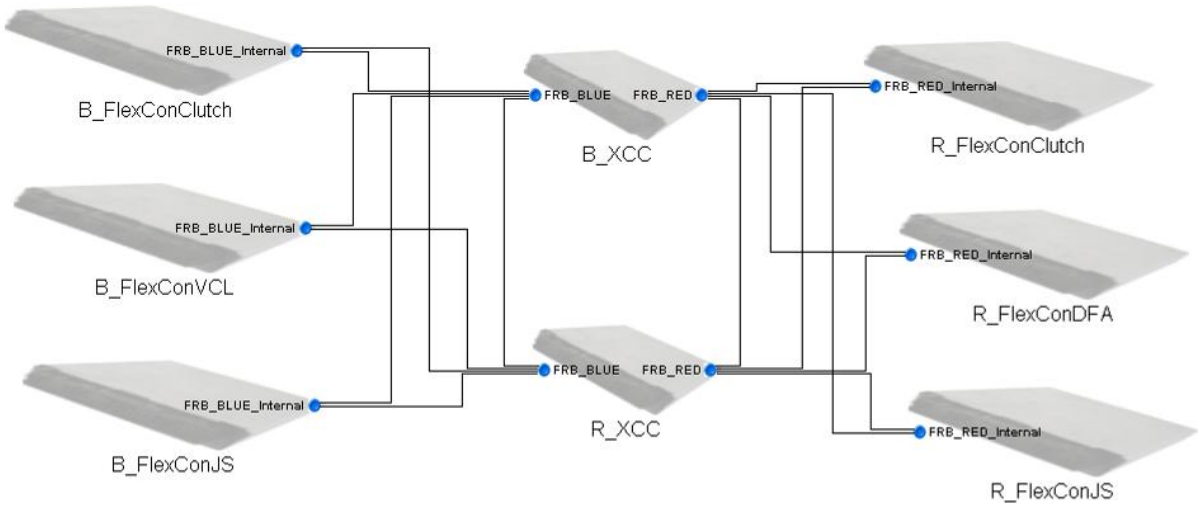

Figure 4: Platform Topology – Excerpt from GME System Layer SbW

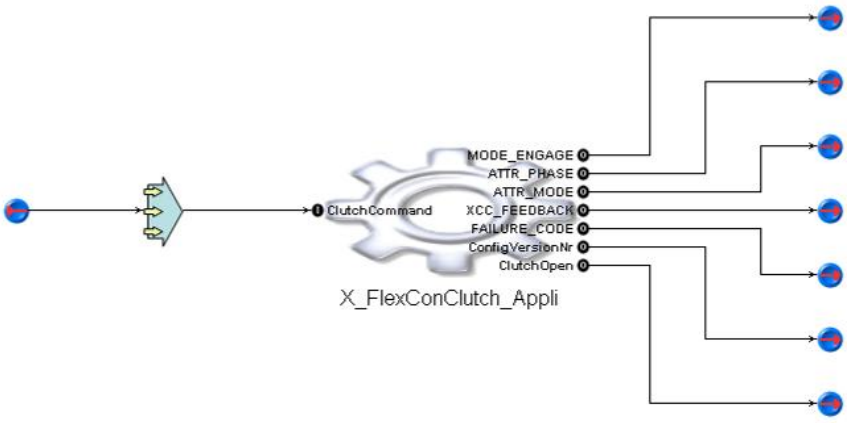

Figure 5: Application »Clutch« - Excerpt from GME System Layer SbW

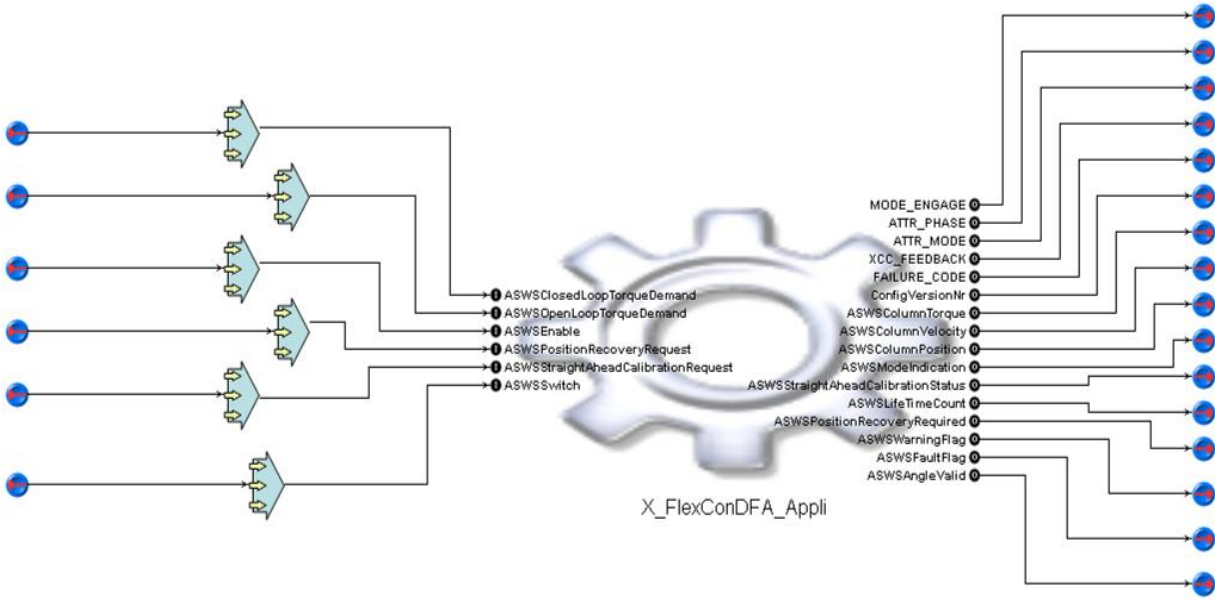

Figure 6: Application »DFA« - Excerpt from GME System Layer SbW

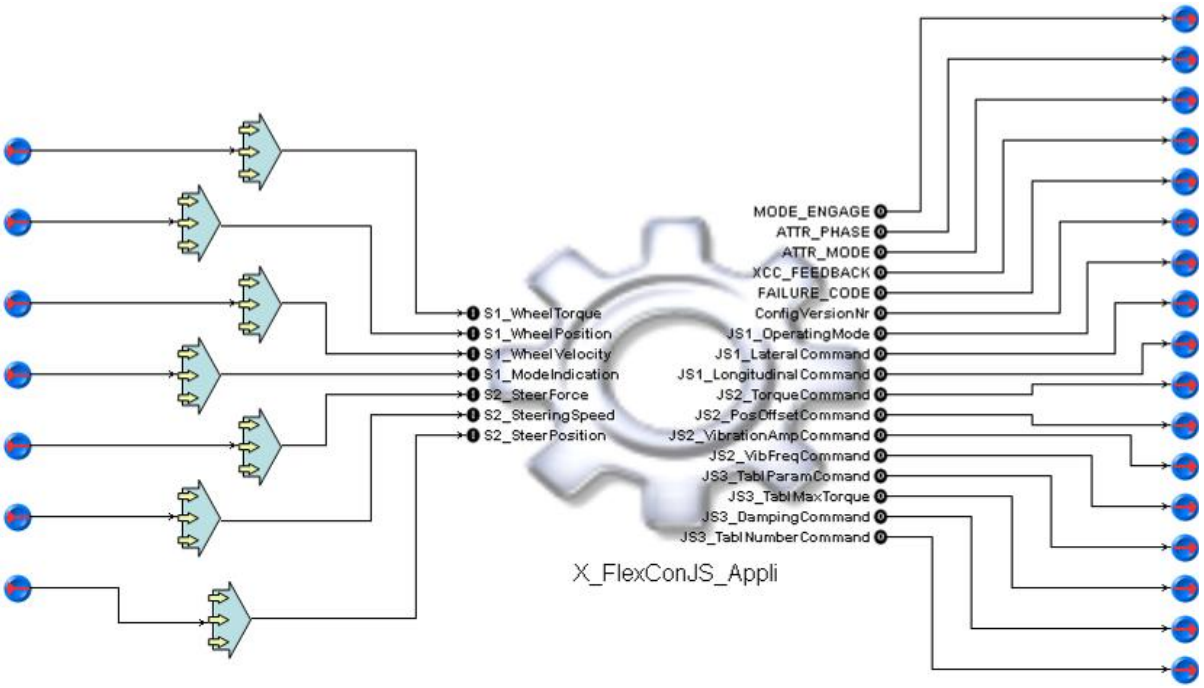

Figure 7: Application »Joint System« - Excerpt from GME System Layer SbW

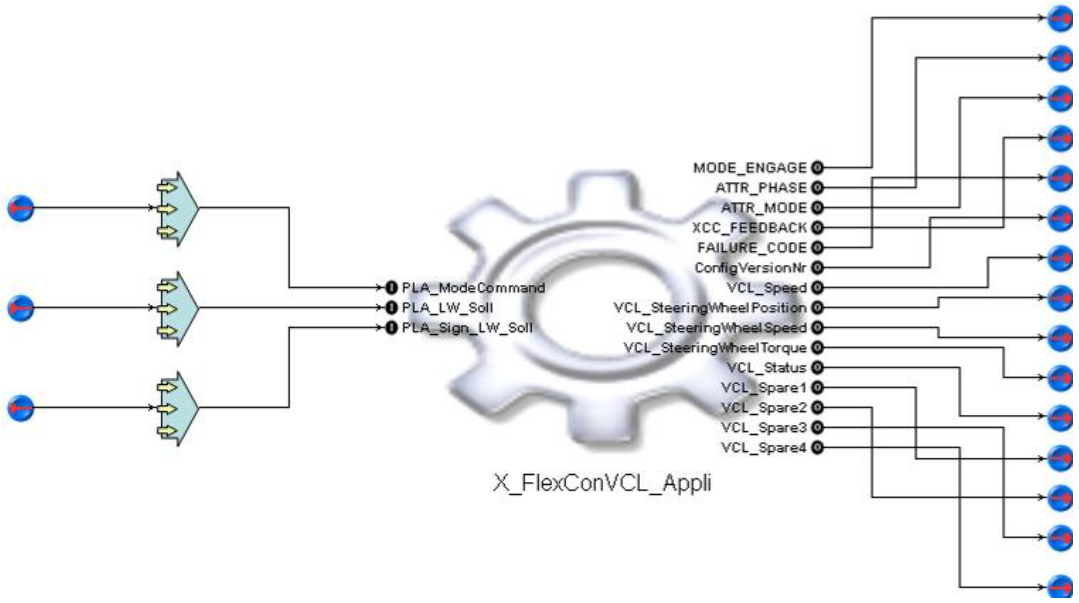

Figure 8: Application »Vehicle« - Excerpt from GME System Layer SbW

3.2.2 Brake-by-Wire

In the following, excerpts of the system layer modelling of the brake-by-wire system (WP 4200) are shown exemplarily in figures 9 to 11. Again, the presented figures solely constitute excerpts, due the size of the models (see table 2).

Table 2: Number of Input- / Output-Signals per Application – BbW

Model	Number of Input Signals	Number of Output Signals
Application »EBA«	20	27
Application »ESP«	8	16
Application »Gateway«	91	83
Application »HMI«	25	18
Application »PDM«	18	29
Application »BBW«	292	186

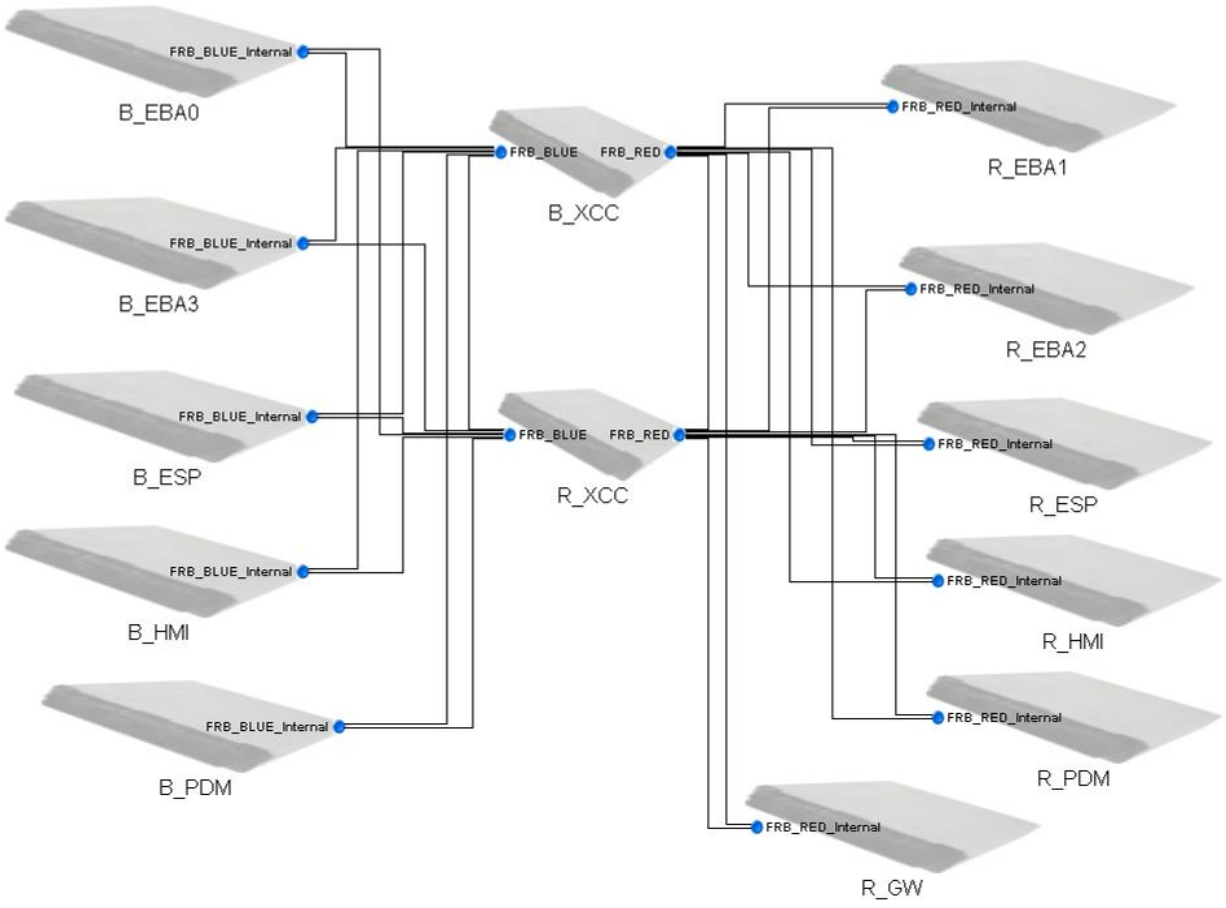

Figure 9: Platform Topology – Excerpt from GME System Layer BbW

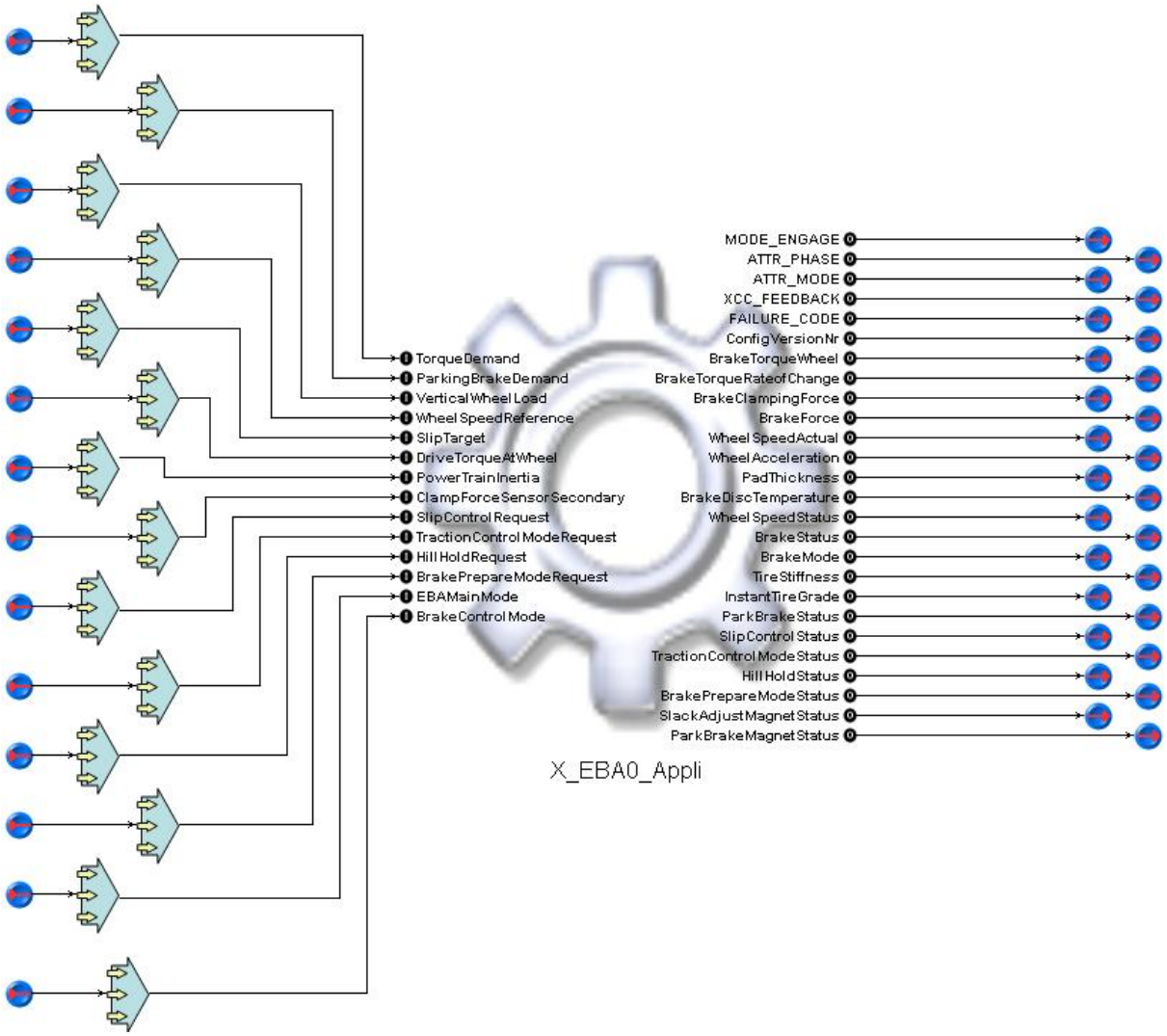

Figure 10: Application »EBA« - Excerpt from GME System Layer of BbW

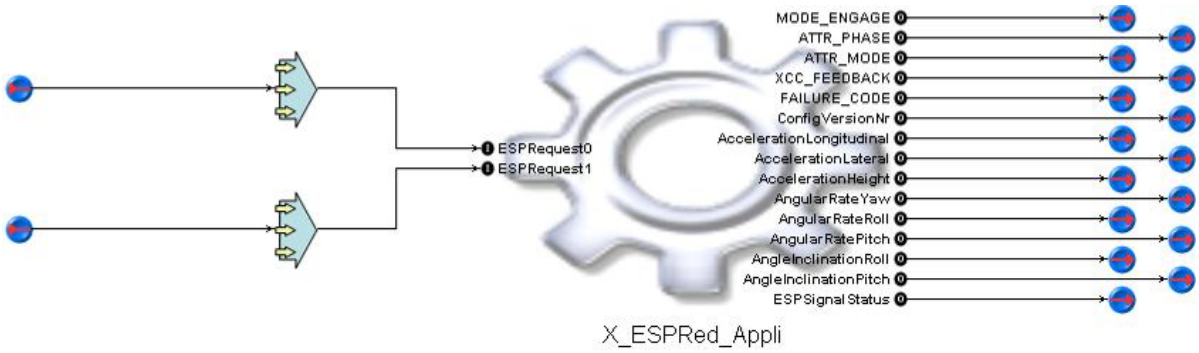

Figure 11: Application »ESP« - Excerpt from GME System Layer of BbW

3.3 Container Layer

Within the container layer, the demarcation between the topology and the applications existing up to this point is broken free, respectively the topology and application models are merged together. This task happens completely independent from any user intervention, based on predefined transformation rules. Datajumpers within the application models are here replaced by real connections for each module the particular application is executed in. In addition, the FlexRay frame's payloads are initialised, functional groups are derived, dedicated signals are referenced where needed and first redundancy management internal status information is automatically added. Just to give an impression about the size of the container layer: for the simple steer-by-wire system, the topology model within the container layer still contains more than 1400 signal connections, while for the larger model of the brake-by-wire system these are even more than 5000 signal connections.

3.4 Software Layer

The aim of the software layer is to finally provide for all redundancy management software modules, the GME model is intended to provide the configuration data for, the required information data. Therefore, during the development phase of the Meta-Tool framework itself, each software module responsible within USTUTT had to provide the specific paradigms for his module. This was due in order to ensure, that by the automatic derivation of the software layer out of the container layer the complete data setting for the target software modules will be generated.

With derivation of the software layer, the dataflow is expanded to its very full complexity. Here, not only the module redundancy is taken into regard, but also the redundancy of the FlexRay buses. Figure 12 is intended to enlighten this: while for example the modelling represents a single signal, which is transmitted from one XCC to another, by solely one single signal connection, in reality, this signal is transmitted via up to 8 transmission lines. By deriving the software layer, the models are expanded in order to reflect that reality. Furthermore, again software module specific status information is generated.

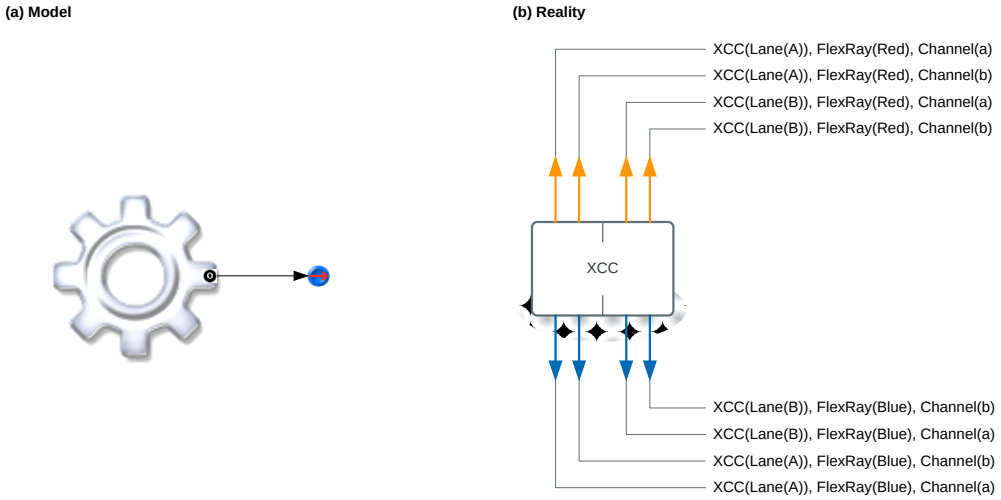

Figure 12: Dataflow-Modelling and Bus Redundancy

By creation of the software-layer, the complete set of information for all software-modules, the GME is intended to provide the data for, is finally available within the model. Having achieved this step, the whole model is exported to an XML file in order to provide a common base for further processing. Latter will be topic of the next chapters.

4 Derivation of Code

After having generated the XML document containing the complete dataflow information (see previous chapter 3), we need to pick out of that global data pool the specific information required by each individual software module and transform them into a format, the software module is able to work with. For this task, XSLT was chosen to be the appropriate tool kit.

The extensible stylesheet language transformation (XSLT) is an XML based language, originally aimed for the transformation of XML documents into other XML documents. Nevertheless, XSLT also provides the ability to transform XML documents into plain text. This feature is used within the Meta-Tool chain to extract the specific configuration data for each redundancy management software module out of the XML file which contains the complete dataflow information, and to directly generate *.c and *.h files containing the configuration data for the particular software module (see “Online Configuration” within [2]).

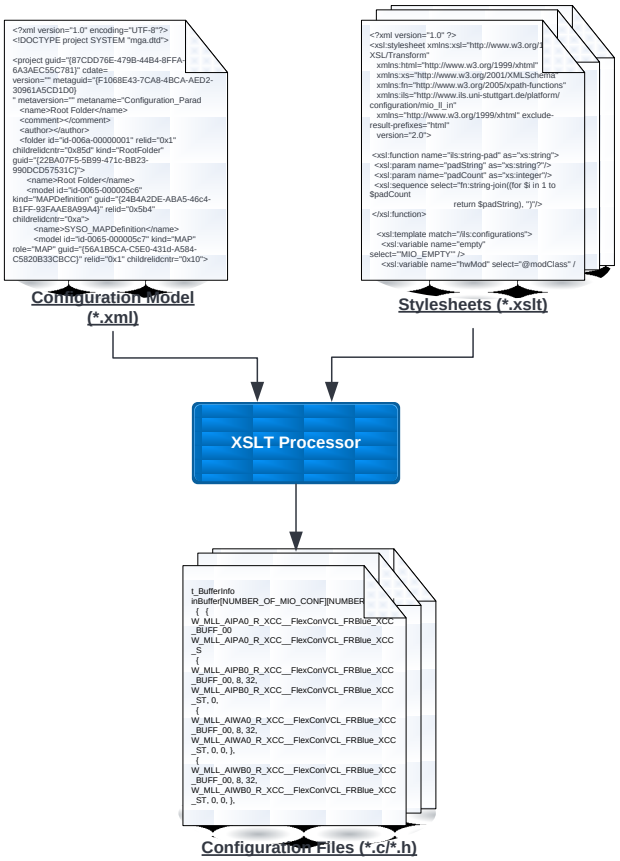

Figure 13: Derivation of Configuration Files

Beside the mentioned, not only the configuration data for the redundancy management software modules are derived this way. As shown in figure 2 additional outcome like the overall database definition for the XCCs (*.c and *.h files), configuration of the driver-handlers within the USTUTT operating system (*.c and *.h files), the required information for configuration of the FlexRay communication controllers for the whole platform (*.xml), as well as configuration of the testbench databases (*.xml) can also be derived using the described method.

5 Platform-Core and OS

The configuration of the more platform core related software modules of the redundancy management (these primarily are »FailMa«, »CoMo« and »PlaMa« (see Figure 2)) is principally generated only once per each specific type of platform core. Within the HAVEit project, both of the by-wire-vehicles, which are planned to apply the XCC platform, will be equipped with the same type of platform core. This is mostly distinguished by the following properties:

- The platform-core consists of two XCCs and two FlexRay buses
- The platform-core hosts only one system functionality (SbW or BbW)
- The platform-core solely communicates via FlexRay to the outside world.

In addition, further characteristics like the absence of a build-in-test or abandonment of wakeup-ability of the whole system via FlexRay accounts for the individual platform-core's type, too. In consequence, the configuration task for the mentioned software modules is mostly sufficient to be done once for both vehicles.

Especially the Platform Management (PlaMa), which controls besides others the platform wide mode engagement, master/slave assignment, or passivation of the own XCC in case of failures, is exclusively dependent on the platform core's type. A first version of this software-module has already been implemented in the generic platform core (see [2]). In the meanwhile, this version was enhanced and – in consequence – PlaMa expanded to its full functionality.

The Failure Management (FailMa) as well as the Consolidation Module (CoMo) are mainly dependent on the type of the platform core, too. However, minor dependability of dataflow-specific issues does exist (e.g. on the concrete amount of FlexRay frames, a XCC is intended to transmit, and the like). Same as with the PlaMa, both – FailMa and CoMo – are used to be configured by hand. Therefore, an Excel-based spreadsheet was developed, which allows the user to perform his / her configuration task quite comfortable. These spreadsheets were equipped with a build-in solution to directly export the required information into *.c, resp. *.h files.

Though nearly all of the redundancy management software modules are configurable by the Meta-Tool chain, there are still a few modifications left to be done completely manually. These mainly affect the Job-Scheduler of the operating system as well as the functionalities being closely related to that. For implementation of the bare platform cores, these tasks were already performed in order to start the whole system running. However, by implementing the final application, which will be the challenge of the upcoming period, the particular settings will be needed to be reworked for sure.

6 Testing

After derivation and completion of the whole platform core configuration, the software package was merged together and uploaded to the two XCCs within the platform core. In addition, the testbenches were configured using the configuration output mentioned in chapter 4. Therewith, the groundwork for testing the individual platform cores was established.

The USTUTT test environment principally provides two different kinds of testing: first is manual manipulation and monitoring of predefined signals within the GUI; second is automated testing by using e.g. python scripts. Especially the latter was the first choice for testing the more dataflow dependent properties where a high amount of signals needed to be manipulated in an analogue manner. Manual testing was preferred for testing the more platform core related software modules for proper operation.

6.1 Dataflow

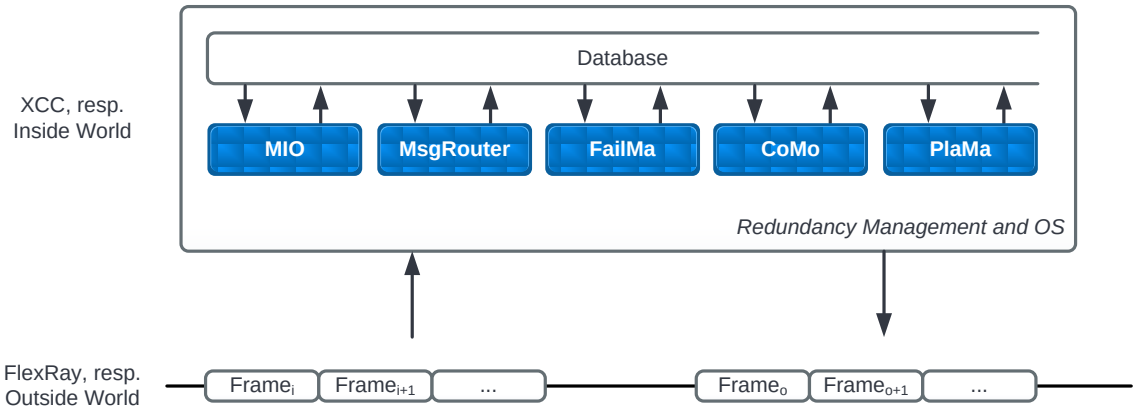

Figure 14: Dataflow Differentiation

Concerning testing of the dataflow, the tests could be principally split into those tests which mostly affect the “outside world” (i.e. transmission via FlexRay) as well as those, which are more concerned with the “inside world” (i.e. dataflow within the XCCs’ redundancy management) (see figure 14).

Testing the “outside world” mainly addressed the correct transmission and reception of FlexRay frames within the platform. Therefore the vehicle aggregates where simulated by physical FlexRay nodes within the test benches. The complete FlexRay dataflow of the related by-wire-system was emulated by appliance of the USTUTT test environment. By doing so, the correct communication interaction of the XCCs within the platform cores with the rest of the platform could be ensured.

By testing the “inside world”, mainly the redundancy management’s failure detection mechanisms were in focus. The test shall prove that faulty signals are detected for sure, and that the overall platform reacts in the expected manner. As already mentioned in chapter 2, for each and every signal being transmitted within the platform, properties were defined, allowing the redundancy management mechanisms to justify whether the signal is faulty or not. Applying the USTUTT test environment, this information was used to intentionally manipulate each signal to behave faulty. In parallel, the particular reaction of the redundancy management was monitored. Thus, the specific configuration of as well as the redundancy management’s failure detection mechanisms themselves could be proved to operate properly.

Additionally, testing of the “inside world” also includes correct signal unpacking and provision to the application interface and vice versa. Therefore, each of the signals was manipulated to take a remarkable value within the data source. In parallel, appearance of exactly that value within the expected data sink was checked, in order to prove the “inside world’s” dataflow for proper operation.

6.2 Platform-Core

For testing the platform core related software modules, dedicated platform core conditions were enforced by usage of the test environment to provide initial states for further reaction. Right after, the reconfiguration of the redundancy management software modules was monitored to ensure their correct behaviour. By doing so, the outcome of the configuration task concerning the platform core related software modules could be proved to operate properly.

7 Conclusions / Outlook

The present deliverable showed, that starting from generic platform cores, which in a broader sense consist of

- two XCCs communicating via two independent FlexRay buses,
- a generic, configurable redundancy management software package,
- as well as a fitting meta-tool framework enabling configuration of that software package

the platform cores could be configured according to the dedicated properties of the targeting by-wire-systems. The resulting configured platform cores were appropriately tested and proved to operate in the expected manner. Therewith, the aim of the deliverable – provision of the bare platform cores, configured for the target vehicles – was successfully achieved, too.

During the upcoming periods, the so far “bare” platform cores will be upgraded to the “full” platform cores by implementation of the partner’s control law applications. After a one month lasting integration and testing phase in the USTUTT laboratory, the platform cores will be finally implemented into the vehicles of WP 4100 and 4200.

References

[1] D21.3 “Generic platform core demonstrator in lab available“, HAVEit deliverable, 2010

[2] D21.2 “Software and Configuration Process Concept available”, HAVEit deliverable, 2009

Annex 1 Abbreviations

BbW	Brake-by-Wire
CoMo	Consolidation Module
DFA	Driver Feedback Actuator
EBA	Electronic Brake Actuator
ECU	Electronic Control Unit
ESP	Electronic Stability Control
FailMa	Failure Management
GME	Generic Modelling Environment
GUI	Graphical User Interface
HMI	Human Machine Interface
MIO	Module Input / Output
MsgRouter	Message Router
OS	Operating System
PDM	Power Distribution Module
PlaMa	Platform Management
SbW	Steer-by-Wire
USTUTT	University of Stuttgart
WP	Workpackage
XCC	X-by-wire Control Computer