

USE OF FORMAL VERIFICATION FOR THE SOFTWARE DEVELOPMENT IN THE AUTOMOTIVE AREA

A. Köhler¹, D. Kant²

¹ Volkswagen AG, Technical Development - Vehicle System Electronics - Software Platform,
Address: Letter Box 1722, Germany, D-38436.
Phone: +49 (0) 5361-947257, Fax: +49 (0) 5361-95747257, e-mail: andreas3.koehler@volkswagen.de

² Audi Electronics Venture GmbH, Technical Development – Networking Innovations – Time Triggered Architecture
Address: Sachsstraße 18, Gaimersheim, Germany, D-85080
Phone: +49 (0) 841 8942841, Fax: +49 (0) 841 8943792, e-mail: dietmar.kant@audi.de

Abstract: The method of Formal Verification is getting more and more important for the software development in the automotive industry because of completeness and the support of Frontloading. This has been investigated for real applications in the context of established tools in this area. Integrating this method into the software development process requires further improvements on the performance, interfaces and supported language of the tools. But in the context of deterministic time triggered architectures there are additional possibilities to support Frontloading in the development process for software. These interesting topics are represented in this paper.

1 INTRODUCTION

The trends in the automotive industry for software development are getting closer and closer to a process, which uses formal methods. There are certain reasons for this. The most important one is to increase the quality of a software product. This results in a design oriented approach, where most parts of the system behaviour are specified in the early phases of the development process. This means on one side to spend more effort for requirements management and requirements engineering to save a lot of effort in testing at the later steps of the development process (Frontloading) and on the other side to have deterministic design approaches for the system. The last issue means, that the system behaves as specified during design time. This results in lower development costs, which must not exceed a certain budget. Formal verification can support this partially because it is a method to proof the fulfilment of requirements completely. Completely means the fulfilment of requirements can be guaranteed. But there should be tools, which

satisfies certain requirements of the automotive industry. The two most important requirements are the input language and an appropriate tool support including a sufficient performance. Up to now there are two tools, which are promising and have therefore been evaluated in Volkswagen and Audi. For the reason of understanding one of this tool is shortly introduced in chapter 2. It was also investigated how to integrate the method of formal verification into current development processes. Results are described in chapter 3. Additional requirements on verification tools comes from the environment of the automotive industry and are written in chapter 4. The last chapter outlines the long term visions.

2 THE SOFTWARE DESIGN PROCESS

The software design process starts with requirements management to get requirements from the stakeholder, to evaluate, attribute and manage them. Tracing and change management are the

important things for the automotive industry to stay flexible and consistent to guarantee high quality. Because this requirements are user requirements they are specified in a non formal way. This can be done with several tools, which support linking to the system requirements designs, implementation and test cases to guarantee traceability. This linking mechanism is important to get an impact analysis easily when a change of a requirement is necessary. Additionally change management should be supported with configuration management of requirements. At the moment the RM tool DOORS™ is used to specify user requirements in an informal way.

After several internal reviews of this work-product, the system requirements are specified. This is the beginning of using formal description languages. MATLAB™ Simulink™ and Stateflow™ are used as a modelling tool to describe the functional behaviour, which is required from the software controller. This model can be used for prototyping to check the user acceptance and to clarify the needs of the user. In general several high performance computers are used to demonstrate the new function in a car. Because of cost reasons this hardware can never be used for the series production of a car. It is used to show possibilities of innovation for the next generation of cars. This model is not completely specified. It represents only the good cases and does not contain any failure and error handling mechanisms.

After the decision to bring a new function into the next car generation, the development of software including all failure and error handling mechanisms for a low cost hardware is started. There is a major difference to other areas of software development like the aerospace industry. This is the number of pieces in the automotive area. The complete Volkswagen group produces about 5 million cars per year as opposite to a few hundreds of planes built within a year by an OEM of the aerospace industry. This results in the fact that most of the costs of a car are generated by production cost, which has to be minimised as far as possible. This includes the use of low cost hardware for the execution of the software. This does not mean low quality but lower performance.

Currently used hardware has a domain of processor performance from 40 MHz to 300 MHz and in general up to a few hundreds of kilobytes of memory. Some platforms have a floating – point unit, others require the use of fixed – point arithmetic. To adjust the model to the underlying

hardware for the series means to develop an implementation model including hardware specific characteristics like scaled data types and operators. But this restricts the re – usability of models for different car platforms what is mandatory for an OEM in the automotive industry.

Therefore the first use of formal verification is the complete proof of requirements on the requirements model.

For this purpose the Simulink™ Model is translated into a SCADE™ model, which contains only strongly typed modelling constructs. Also the sample times are consistently defined with the semantics of the synchronous language Lustre. Lustre is the underlying formal language for the graphical representation of the SCADE™ model. This formal description language guarantees the deterministic behaviour especially for the execution order of operators. The textual Lustre program is the base for formal verification with the Design Verifier. But the requirements, which should be verified, can be expressed in a graphical way in SCADE™ with the support of a special verification library. These formal requirements can be linked to user requirements defined in the RM tool DOORS™.

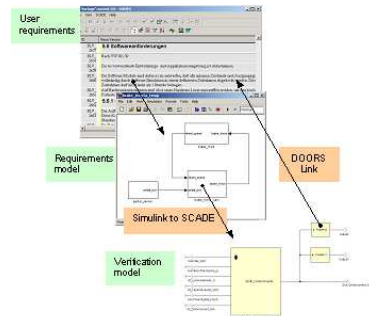


Fig. 1 Design process for formal verification

After this definition the formal verification can be performed like described in the following chapter.

3 FORMAL VERIFICATION IN PRACTICE

In the following we describe the use of formal verification for a safety relevant application, which is part of a Steer by Wire system. In Steer by Wire systems there is no direct mechanical connection between the steering wheel and the steering system. It is replaced by an electrical one. Therefore there is no feedback from the car to the driver. This would result in a steering system, which is difficult to control. Therefore a feedback must be generated artificially. This is the purpose of the force feedback demonstrator developed in the European research project RISE to evaluate different approaches on formal verification in time triggered systems mixed with event – triggered behaviour.

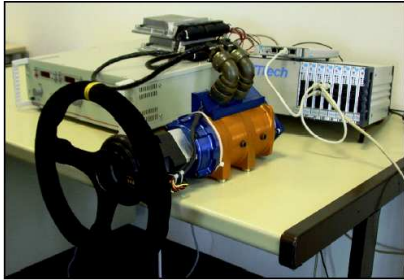


Fig. 2 The force feedback demonstrator

The demonstrator consists of a steering wheel, which is driven via a planetary gear by two brushless DC motors. At the steering wheel sensors are connected to determine the position of the steering wheel and the moment of force given by the driver. The control of the steering wheel is performed by a so called development cluster delivered by TTTech. This development cluster is like a network of electronic control units connected via a time triggered communication bus.

3.1 The model for the demonstrator

In the following the application of a conventional steering system with the feedback by restoring

torque is presented. The application was developed in a model based way, which starts with the implementation in a MATLAB™ Simulink™ model. This model was partitioned in three subsystems, which are shown in the following figure.

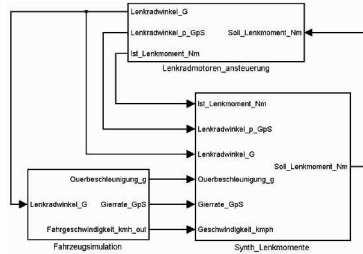


Fig. 3 The partitioned Simulink™ model

In this model not only the calculation of the feedback is modelled but also the emulation of the rest of the car. The emulation of the vehicle behaviour is performed in the subsystem *Fahrzeugsimulation*. This module takes the angle of the steering wheel (*Lenkradwinkel_G*) and calculates the yaw rate (*Gierrate_GpS*) and the lateral acceleration (*Querbeschleunigung_g*). For this purpose the values are calculated with a linear vehicle model. In this vehicle model the characteristics of the wheels is assumed to be linear. Additionally the height of the balance point is set to zero, whereby the wheels of this axle can be reduced to one wheel.

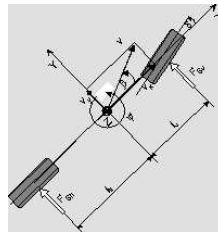


Fig. 4 Principle of a linear vehicle model

The speed of this virtual car (*Fahrgeschwindigkeit_kmh_out*) is set to a constant value within this subsystem, is also used for calculation and is additionally an output value of this subsystem. These values are necessary to calculate the steering moment for the driver (*Soll_Lenkmoment_Nm*) in the subsystem

Synth_Lenkmomente. The subsystem *Lenkradmotoren_ansteuerung* represents the software interface to the steering actuators and sensors. It gets the input variable steering moment in Nm (*Soll_Lenkmoment_Nm*), which should be felt by the driver. The output variables are the steering torque from the driver (*Ist_Lenkmoment_Nm*), the angle of the steering wheel (*Lenkradwinkel_G*) and the speed, which has the steering wheel by changing the angle (*Lenkradwinkel_p_GpS*). These values are delivered by special sensors.

3.2 Defining properties for formal verification

There are two requirements, which must be satisfied by the algorithm. The first one is obviously, when the developer has a look at the powerful electromotor of the demonstrator which can generate a high torque. This has to be limited to a value, which can be oversteered by every driver. For this application 10Nm was chosen and expressed in the following requirement modelled in SCADE™.

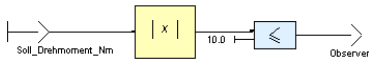


Fig. 5: Requirement 1 modelled in SCADE™ (Property 1)

Another important issue, if a system gives a force feedback to the driver, is the way the force changes its value. This means that the difference between two calculations should not exceed a certain value. Otherwise the driver could be injured. For this demonstrator application it was determined that the value of torque should not differ more than 0.5Nm between two instances of calculation. This is expressed in the following requirement modelled in SCADE™.

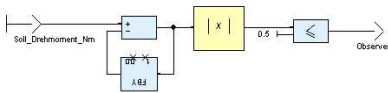


Fig. 6 Requirement 2 modelled in SCADE™ (Property 2)

After defining these two properties in a SCADE™ node, they have to be connected with the corresponding subsystem *Synth_Lenkmomente*, which is the software controller for this demonstrator.

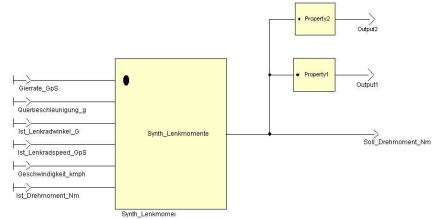


Fig. 7 Software controller with the defined properties

3.3 Defining assertions for formal verification

Because it should not be verified whether the requirements are fulfilled at a vehicle speed of 32767 km/h the value domain of the input variables has to be limited to a reasonable interval, which corresponds to the reality. For this purpose several tests have been driven on the test – bench. The corresponding data have been monitored and replayed in Simulink™ for the purpose of visualisation. The results are shown on the following two figures.

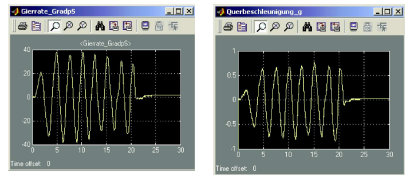


Fig. 8 Measurements for yaw rate (left) and lateral acceleration (right)

It can be recognised that the yaw rate does not exceed 40 degrees per second at all in this worst case driving scenario. The lateral acceleration is always less than 0,8 g whereas g corresponds to the acceleration due to gravity (9,81 m/s²). The value domain for the vehicle speed can be determined by the CAN database. This database shows that the speed of the car is trans-

mitted via the CAN bus and can have values from 0 to 325 Km/h. The steering wheel of this demonstrator can be turned up to 540 degrees in both directions, because there is one potentiometer, which has a physical limit at this position. The maximum value for the speed of the steering wheel has been measured with powerful drivers and does not exceed 128Nm respectively 839 degrees per second in both directions. These physical limitations of the environment can be expressed with assertions in SCADETM.

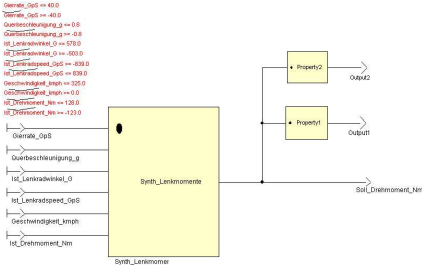


Fig. 9 SCADETM model with constraints of the environment (assertions)

3.4 Results of formal verification

In general the promising method of formal verification can be difficult, if non – linear arithmetic is inside the model. This means a simple construct like $C=A*B$ or $C=A/B$, where A and B are inputs to the model, which are not constant. The software controller of the demonstrator, which is used has some non – linear arithmetic inside. But there are solutions to solve this problem. For this case study the proof of the requirements is done in a numerical way with theorem proving and shows that the model satisfies them. But this requires additional manual work and it has to be investigated if it is acceptable for more complex algorithms.

4 FORMAL VERIFICATION OF A WARNING SYSTEM

In the other European project NextTTA a warning system was used to evaluate the method of formal verification. This warning system is part of a driver assistance system.

This application recognises if a car is moving towards its physical limits and generates a warning to the driver. The controller checks if the warning is permitted depending on several conditions. For example a warning is not permitted if a speed of the car falls below of a certain value. If the warning is permitted it is passed. Of course the warning should have a minimal duration to be recognisable by the driver. So the controller continues to deliver the warning signal, if the minimal duration is not reached. There are two warning signals in the system, one warning for the right and one for the left side.

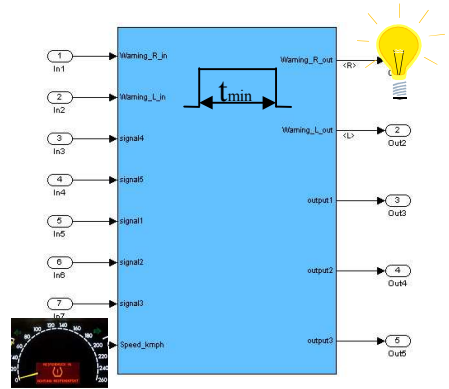


Fig. 10 Subsystem of the warning system

4.1 Defining properties for formal verification

From this short description two functional requirements can be derived, which were verified. One requirement is, that the warning for the left side and the warning for the right side should never be true at the same time. This can be easily expressed in SCADETM with the following property.

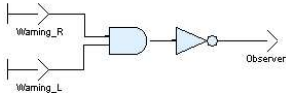


Fig. 11 Property 1 for the warning system

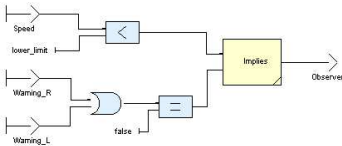


Fig. 12 Property 2 for the warning system

Another requirement is that there should be no warning if the speed of the car falls below a certain value. This can be also expressed very easily with the property shown on the following figure.

4.2 Results of formal verification

The problem we get was, that the signal delay, which was implemented to guarantee, that the warning signal has a minimal duration causes a high sequential depth, which causes a long execution time for the verification process. For example the following part of a system generates a delay of 0.5 seconds with a sampling rate of 1 millisecond:

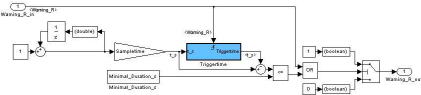


Fig. 13 Timer delay which causes a high sequential depth

This results in 500 states, which is complex for verification. But between two instances of calculation there were no changes on the signal. Therefore it can be optimised automatically by the verification tool instead of manually by the user. This improvement was implemented in the tool used for formal verification and the process was repeated.

Now we got a counterexample, which means that there was a scenario generated which shows the combination of input values where the property fails. First it was checked, if all the values are in a reasonable value domain. This was given by the generated counterexample. Then we use it for simulation to see if there is a bug within the model. The result can be seen on the following figure.

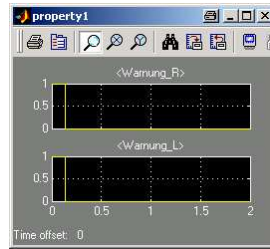


Fig. 14 Simulation results for the counterexample

It can be seen that there can be a warning for both sides at the beginning. This refers that there is an initialising problem in the model. Further debugging and simulation confirms this assumption. This is a typical problem within software development, the access to memory, which is not initialised. This could result in an error, which is difficult to find at the later steps of the development process. We remove the problem and get a positive result within a few minutes.

The verification of the other property brings also a counterexample. It was not obviously at the beginning, when the property was specified. If the speed exceed the threshold and then falls below of it the warning is delayed up to the minimal duration. This means, that the requirement to give no warning under a certain speed of the car is not satisfied by the model. There could be two reasons for this. The first one is that the model contains an error. This could be that the warning should immediately be aborted if the speed of the car falls below of the threshold. The second reason can be that the warning should be delayed in any case up to the minimal duration if it is permitted at the beginning. The first one would refer to an implementation error in the model and the second one would be an error in the specification respectively the wrong conversion of non – formal requirements into formal requirements.

After clarification, it was determined that it was an error in the specification. The corresponding requirement was improved and again formally verified. A positive result was also got within a few minutes.

5 REQUIREMENTS ON FORMAL VERIFICATION

To integrate the method of formal verification into current development processes some important requirements shall be satisfied. These are described in the following.

5.1 Support for re-usability

Re – usability is a very important issue in the automotive industry. This regards also to software. It means to re – use software functions for several car platform without any changes. This reduces development costs and increases the quality. In a model driven development process re – usability is considered on the model level. The code level contains already hardware specific constructs like the data types of the target hardware and is therefore not portable without any additional effort. The requirements model in MATLAB™ Simulink™ can have a generic datatype, which depends on the context in which it is simulated. The same is valid for the sample time. It can be chosen as inherited and depends also on the context of simulation. Therefore the hardware independent requirements model is the level at which software is re – used. Porting a software model to another car requires a parameterised model. But the question is how to handle this parameter value. There are two alternatives. The first possibility is to handle it as a constant value and to repeat the formal verification process every time this value is changed. The second and more reasonable way is to handle it as a input variable and to verify, that the property is satisfied for a certain domain of parameter values. Unfortunately this is not supported by any tool. If this is done manually, it requires a lot of effort and is negative for the performance of the verification tool because of higher complexity of the

model due to the additional input. Therefore the first approach is used. This brings the requirement for the verification tool to support parameters. In MATLAB™ parameters are defined in so called m – files and can then be loaded into the workspace. Because an m – file can contain hundreds of different constructs of the MATLAB™ language, it is difficult for a tool vendor to implement the support of all this language features. The better solution would be to read the parameters from the workspace and to translate it into constants.

5.2 Support of the modelling language

Another important requirement is that most of the language features of Simulink™ and Stateflow™ are supported. Of course there are certain modelling guidelines, which restricts the use of the Simulink™ and Stateflow™ modelling language to a safe subset and describe good modelling techniques. But there should not be restrictions due to the verification tool. It should only be a filter for non – deterministic language features but not a bottleneck for modelling. It is clear that this is a requirement on tools, which is sometimes difficult to fulfil. Bigger problems can arise, if there is a new release of MATLAB™. It could takes some time to support the new features of the new release. Therefore there should be a workaround, which provides a possibility to the user to define his own translation rules.

5.3 Performance requirements

In several evaluation projects performance problems arised during formal verification. For smaller subsystems it is necessary to get results within minutes. For larger models it is imaginable to start the model checker at the evening and to see the results at the next morning, if it is the latest check of the algorithm. Metrics are the number of blocks, subsystems and inputs. Models are not developed to be easily verified but to find a good algorithm for the system to be controlled. Nevertheless it should be possible with usual commercial of the shelf PCs to perform formal

verification within the required timeframe. Additionally it should be possible for the user to define his own optimisation rules. One possibility to perform an optimisation is abstraction of complex blocks or subsystems. For this purpose the verification tool should indicate, where the complexity comes from, that the user recognise, where optimisations makes sense.

5.4 Requirements on code generation

There is an obvious trend in the automotive industry, that software functions, which represent unique selling points are developed by the OEM whereas software, which is irrelevant for competition is standardised and can be used commonly by several car manufacturers. Addressing the first point it is required to generate code, which has the same behaviour than the model. This would bring a lot of benefits for the development process and the corresponding process phase module testing. The corresponding module tests can be left out, if formal verification is performed for the defined requirements because of the completeness of this method. But this requires, that the code – generator performs absolute correct translation of the model into C – Code. This must be guaranteed in any case. The question is how to test the code – generator sufficiently to proof that it performs only correct translation. This is nearly impossible with reasonable effort if the input language is extensive like in MATLAB™ Simulink™ Stateflow™. So the input language must be restricted to a certain subset. This is not desired by the control engineer, who develops the model. Another solution could be to make a translation from several blocks into corresponding blocks with the same behaviour but consisting only of block of the smaller subset. But the problem is, that this causes some overhead in the memory consumption and also in the execution time of the generated code. And it must also be guaranteed that the translation is performed correctly. It seems that there is no appropriate solution for the problem.

In any case an evolving standard for the development of safety critical systems will clarify how to test a tool that there are no code reviews and module tests necessary on the code level. It is a tailoring of the generic IEC 61508 standard like

the activities, which are already done in the railway industry.

After generating code for the application it is very important, that this code can be easily integrated with other software components. As already mentioned there is a lot of software, which is irrelevant for competition and is therefore standardised and can be used commonly by several car manufacturers. The most important standard, which is evolving at the time is developed within the AUTOSAR consortium. This includes the standardisation of different APIs between the AUTOSAR software layers and the API to the application software to facilitate the portability and re-usability of application software. Therefore the generated code should contain function calls or memory access to this APIs and also the interface to be called e. g. by the operating system.

6 LONG TERM VISIONS

Formal verification and the deterministic design with SCADE™ TTA integration brings a lot of improvements for the development process especially for safety critical applications. To combine both methods brings also lot of further improvements. The software – controller can be formally verified as a single function. If the result shows, that all functional software requirements are hold by the model, it can be distributed on a time triggered architecture. This implies generating a communication and task – scheduling. The distributed software controller is modelled by the partitioned algorithm and the timing information on signals (communication schedule) and subsystems (task schedule). This model can also be verified against safety – requirements to show, that the distribution does not introduce a counterexample for the given property. After distributing the algorithm on the network, the hardware has to be taken into account. Some ECU's in the network are quite simple and can only calculate with fixed – point arithmetic. Other ECU's could have other requirements, which makes it necessary to have a floating – point unit on the processor platform. These constraints, which are derived from the hardware have an influence on the verification and should therefore be modelled in the software controller. In SCADE™ this is possible by using the fixed –

point implementation and the implementation types for integer variables. This implementation model could also be verified against safety related requirements. Now we would have a model of the distributed software controller with implementation types derived from the hardware. This model would be formally verified and therefore it can be completely guaranteed that it performs only safe functionality. But the behaviour can be different, when the model is executed on the real hardware. For example, if there is a requirement that states “output z and output y should never be true at the same time”, it can be formally verified and the result can be “always valid”. After the integration of the software controller on the hardware, one hardware output line could cause a longer delay than the other one. It is obviously that this can lead to a non – satisfied requirement.

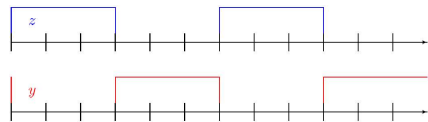


Fig. 15 Example for a property, which is possibly valid for the model, but not on the real hardware

There are three possibilities to take this kind of hardware characteristics into account. One possibility is to model the hardware. This is not reasonable, because it is very complex and would cause a lot of effort. In general several different hardware platforms are used within a network.

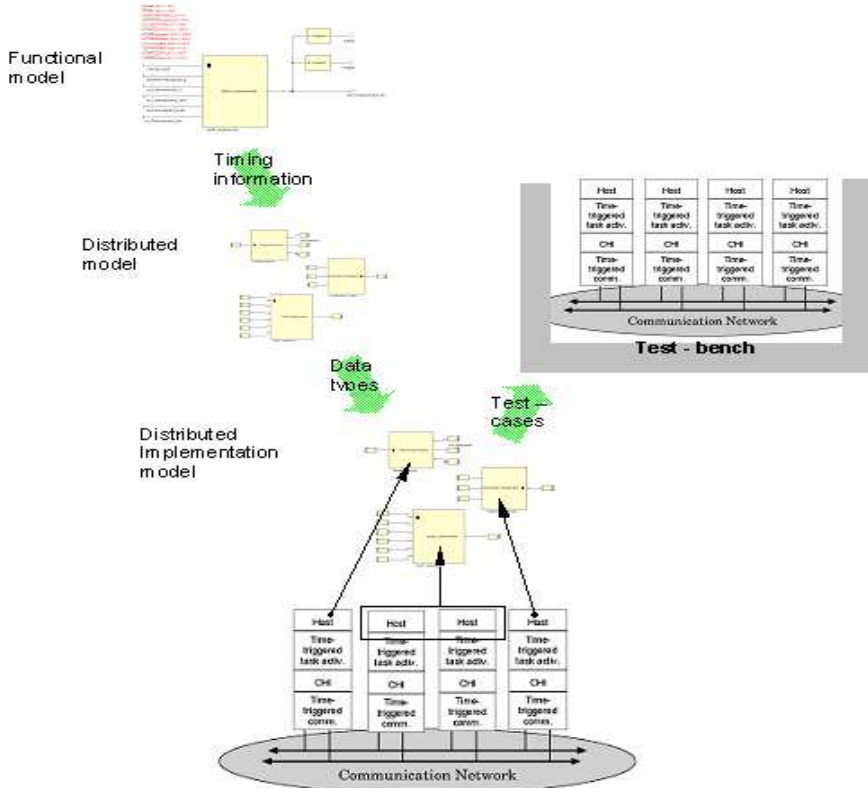


Fig. 16 Long term visions for the use of formal verification in a process

Additionally this approach would be error – prone, because information about the hardware behaviour is difficult to get and approximations must be performed.

The second approach is to consider hardware characteristics and to take it into account when expressing properties. From the process – view software requirements should not be changed because of hardware characteristics. The process should be followed from requirements analysis to the implementation. This means, if the used hardware is not able to fulfil the functional requirements, the hardware must be changed or used with another configuration. But the question is how to verify it completely.

This leads to the third approach, which is aimed and investigated in another project. The idea is to generate test – cases during formal verification of the distributed implementation model and to re – use these test – cases for the hardware in the loop simulation. With these test – cases it should be checked, if all the safety related are satisfied by the model, which is executed on the hardware. If so, this would guarantee completeness of the fulfilment of safety – related requirements from the first functional model up to the real ECU continuously through the complete software development process. This vision will bring a lot of advantages in saving costs and having a save application. It is seen as very important part of the software development process and one key for new innovations for the automotive industry.

All these issues are followed as a whole supported by partners in European research projects and with regard to the usage in the series development.

7 CONCLUSION

As can be seen in the beginning of this paper the method of formal verification can bring improvements in the quality of the software development. But there are constraints with the usability like non – linear arithmetic. But there are solutions which can be added to extend this bottleneck of automation. Additionally there are requirements coming from the environment of the automotive industry which has to be supported by tools in an appropriate way to integrate the method into currently applied development processes. If this is given there are good possibilities to extend this method to use the advantage of completeness for several consecutive phases of the development process.