EVS24
Stavanger, Norway, May 13-16, 2009

Model reusability and cooperation in model based HEV control system development

Antti Leivo [1], Jussi Suomela [2], Ari Hentunen [3]
Helsinki University of Technology, PL 4300, 02015 TKK, Finland, antti.leivo@tkk.fi
[2] Helsinki University of Technology, jussi.suomela@tkk.fi
[3] Helsinki University of Technology, ari.hentunen@tkk.fi

Abstract

Model based control system methodologies are increasingly used in HEV control system engineering to address rise in total system complexity and development effort. Models are often developed in parallel by lots of modelers. Models are also reused in all project phases as well as in other similar projects. There is constant threat of project failure due to delays and poor quality if sufficient preparation is not done. In heavy work vehicles the control software cost is even more critical due to small production series. Applying strict development process guidelines, overall architecture design and proper training have been tested by us and it has resulted in increased efficiency in HEV control system development.

Keywords: control system, controller, modeling, training, simulation

1 Introduction

Software controlled systems are taking an important role of the functions implemented in vehicles – from small passenger cars to heavy duty work machines [1]. This is even more true in case of hybrid electric vehicles. The progress of embedding software into vehicles has been from minor auxiliary and comfort systems to safety critical vehicle core functions as legislation and development in technology are allowing it more and more. Modern passenger cars have dozens of embedded processor units and a complicated communication bus structure between those electronic control units (*ECU*s). To address increasing effort and complexity in software development as well as achieving overall understanding of the total system behavior, many developers have started to use model based development tools with rapid control prototyping systems [2].

2 Research case

This paper is made in *HybLab* project (see *Acknowledgments*). HybLab is an ongoing project whose purpose is to develop methods for hybridization of mobile working machines.

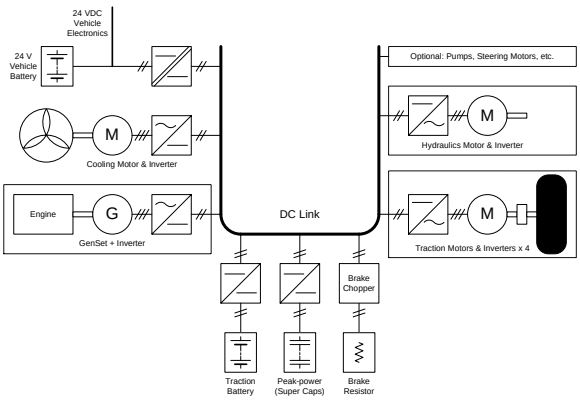

figure 1 Series hybrid power electronics network

In the case studied here we have got an underground mining loader that has a hydrostatic

driveline and hydraulic cylinders for bucket and steering. Hydraulic pump is driven by diesel powered internal combustion engine (*ICE*). In hybridization process the hydraulic driveline will be replaced by an electric driveline. The work machine will be series hybrid vehicle with ICE as a primary energy source and batteries and/or super capacitors for energy storage (see figure 1).

From the modeling and software point of view, the goal is to produce a simulation model of the vehicle and its control system using model based approach. Every significant actuator and energy components with their efficiency data are included in the model. Performance and efficiency can be compared between conventional and hybrid vehicle models with desired work cycle. Models of electric components and power flow make it possible to develop control and study stability of hybrid system.

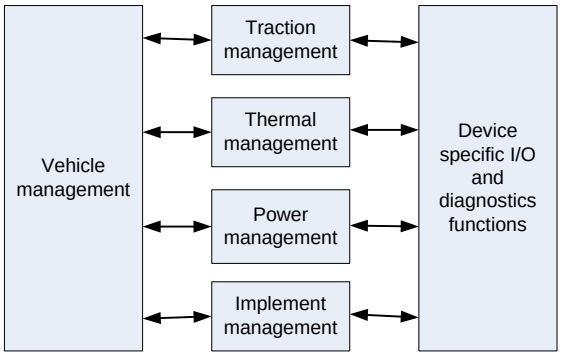

figure 2 Control system modules

In addition to performance and efficiency information gained from the simulation model, the component and environment models are used as a simulated sensor feedback when developing control system software. Control system consists of separate device controls and device diagnostics functions as well as total system level management systems such as overall hybrid electric power management, initialization and shutting down procedures, dataflow between systems and failure situation handling. (see figure 2).

The Mathworks MATLAB/Simulink and many of its additional packages such as *SimScape* and *SimPowerSystems* are used as modeling tool. *dSPACE* rapid control prototyping tools are applied in building control system demonstrator.

Modeling is done in parallel by multiple persons in different physical locations at the same time. The cooperation should be fluent and transparent. Modules of the simulation model should be possible to be reused in similar projects and possibly by different people. The challenge is to develop guidelines and model architecture to succeed in the project.
Our system consists of several control units communicating together. However, most of the subsystems are provided with their own ECU and our task is more to act as an integrator (see figure 3).

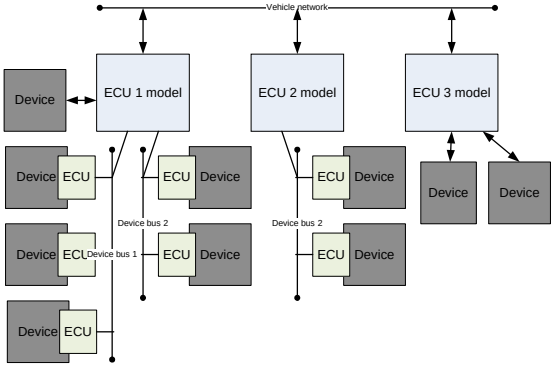

figure 3 Possible control system layout

As we have done these tasks in earlier projects, we have a starting point where we know quite well the possible problem points. In this paper will be explained the challenges one can face in this field and how they could be overcome.

3 Model based software development process

Software development can be seen as a sequence of progressive phases:

- concept design
- requirements specification
- system level design
- unit design
- implementation
- unit testing
- integration
- system testing
- field tests

The previous list could be expanded by test design, documentation, electronics design, further development, updates and maintenance. Development is in practice an iterative design-implement-test process, and commonly expressed

as the V-cycle [3]. This iteration can be simplified by a graphical modeling tool which makes it possible to define, simulate, visualize and document model structure and behavior in an unified environment (see figure 4).

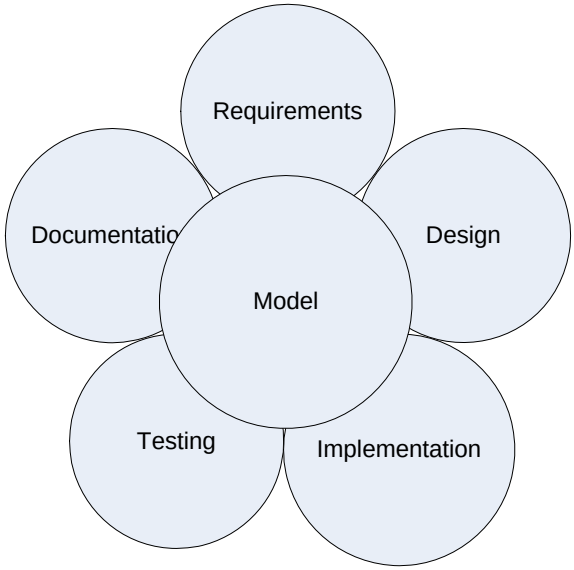

figure 4 Tasks integrated into model

In addition to actual control system model, the vehicle and its environment as well as drive cycle are also modeled with the very same methods. A unified model environment enables virtual testing without the need for real devices in early phases of the project, allowing many kinds of what-if-scenarios easily. This is called *model-in-loop-simulation* (see figure 5). Terms *controller model* and *plant model* are often used to separate the controller(s) that are being developed from the plant that is used as a testing environment.

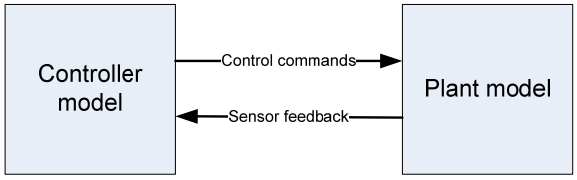

figure 5 Controller and plant model separation

Modeling is often aided by so called rapid control prototyping (*RCP*) systems. Those systems consist of one or more prototyping ECUs with abundant memory, CPU and testing resources. Those are connected to a host-PC. The code, most often C-code, is automatically generated from the models. It is then compiled and loaded into the RCP units. The behavior of the running models (or more precisely, software generated from model) can be monitored and controlled in real-time via host-PC. With those systems, one can rapidly implement the control system and verify the concept [4]. This is the final phase that we are focusing on this paper, but after RCP there would be automatic code generation (ACG) for the production-intent control hardware and better quality requirements. To minimize the need for field tests (not totally replacing them), hardware-in-loop –testing systems (HIL) would be applied. HIL system emulates behavior of real device and environment by giving ECU realistic input/output signals. The overall picture of model based development workflow is shown in figure 6.

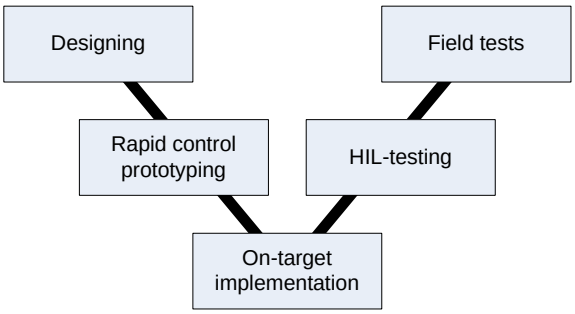

figure 6 Workflow of model based software development

4 Challenges and problems

As with all things, implementing something complex is never straightforward. Here is a list of some of the possible problems that one may confront in this context.

Human coordination problems. Project is divided for practical reasons into several subtasks and several people start to make things on their own way. Integration of the modules may be very troublesome and many parts have to be remade if the specification is inadequate. It may also be that the working roles are overlapping and the same task is specified and done by several people, different ways. It may also be that good specifications are done, but information does not transfer between people.

Data coordination problems. Modelers work with different versions of one or more subsystems or files. Modifications can be overwritten with old versions if data is transferred “manually”, slowly and without systematic backup feature. Editions have to be done again and again. Also, separate specification documents and models often have lots of dependencies. If one changes e.g. CAN

message specification document, then one must change the CAN database file and the control models. This introduces lots of possible error points.

Documentation and startup problems. Documenting is not enforced enough or is only done after the project, when nobody uses it anymore. Almost nobody likes to document willingly. Documentation requires general guidelines as well. There are often changes in personnel. Exiting people cause losses in intellectual property and quiet knowledge. Adding new people into a software project at late stages can generate more work than it is worth [5].

Attitude problems. There can be, from the old school C/C++ programmers, negative attitude towards new methods such as model based software development and automatic code generation. It is also possible that the specifications made are not obeyed as own solutions are seen as better ones.

Modeling tool problems. There are many kinds of time consuming, hard-to-anticipate issues with software licenses, installation, corporate/university security settings, immature tools, insufficient support from development tool vendor. Different simulation and development tools can have complex mutual version compatibility requirements. Differences in software versions between development staff can cause compatibility issues.

External delays and other problems. Component delivery times in demonstrator vehicle projects may be long and variable. New technology components are often prototype quality versions, not yet off-the-shelf/series production. They may require additional engineering and communication with vendor to make them fully functional. Their documentation may also be missing or contain errors or a wrong version. These failures in delivery times may cause enormous delays and unplanned work.

Design considerations. It may be that because of a totally new type of project, system overall picture is not very clear at initial point and it evolves during the project. This may result in a lot of re-engineering. Parts of the models are used in simulation, rapid control prototyping as well as implementation and testing phases. Models have to be designed to support this kind of reuse.

5 Solution

To counter the potential issues, actions have been taken. Solutions have been searched in four different ways: through guidelines, system architecture, software tool and communication.

5.1 System architecture design considerations

Concept of plant model and controller model as in figure 5 has been developed in following way:

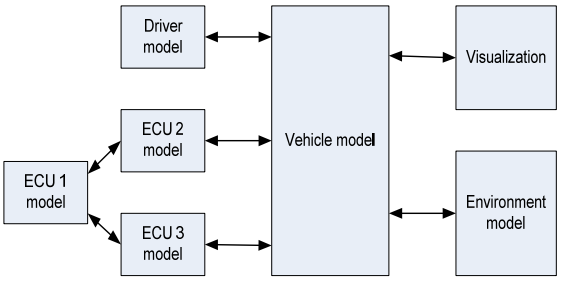

figure 7 Generic MIL-simulation architecture

Controller model is divided into separate ECU models, representing a physical ECU. Plant model is divided into *driver model*, *vehicle model*, *environment model* and *visualization*. With this division, one can take or leave the visualization, change terrain (work cycle environment), change drive style and work cycle actions (driver model; switching between joystick input, drive cycle input, constant input is applied here) or change vehicle parameters and structure. Work cycle power demand does not just come into an abstract ECU. It comes from the vehicle state, affected by the environment model (terrain data). Driver model HMI commands (throttle, brake, steering) as in real vehicle and these are brought to ECU via I/O signals.

When one wants to implement the ECU functions in a real RCP hardware, the setup is as follows (ECU 2 from figure 7 is reused as an example in figure 8):

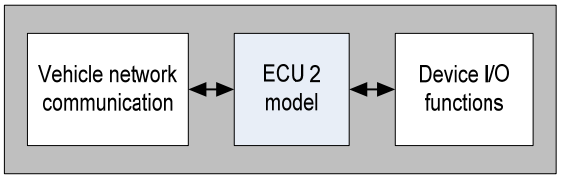

figure 8 RCP model components

The ECU model itself remains unchanged, but hardware I/O abstraction brought into the model

via *Vehicle network communication* and *Device I/O functions* interface model blocks. It is also worth noting that handling these mentioned functions in separate blocks and communicating via grouped signal bus resembles an *AUTOSAR* software function [6] and this may be helpful if you will be migrating your system into AUTOSAR standard later.

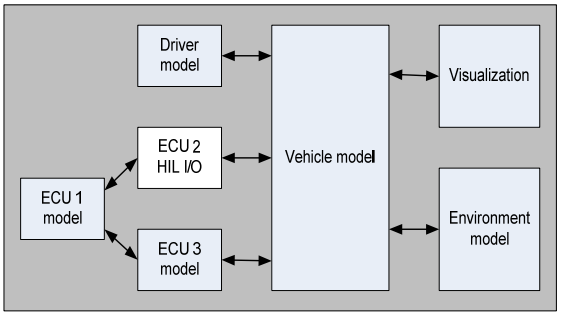

figure 9 HIL-simulator model structure

In HIL-phase, one wants to test ECU 2 which runs in a real ECU hardware. A HIL-simulator runs the whole plant model and brings signals to the ECU 2 via HIL-simulator's *ECU 2 HIL I/O – interface* (see figure 9).

As well as separation of big concerns (controller, vehicle, I/O, driver, visualization, environment, communication) it is necessary to separate each component model (ICE, generator, battery, DC/DC-converter as well as cooling system components) into a module. Figure 10 shown main main division which should be further subdivided into single components. Make them modular, parameterized and generic. Always, if possible, use the same signal interface for components of the same type.

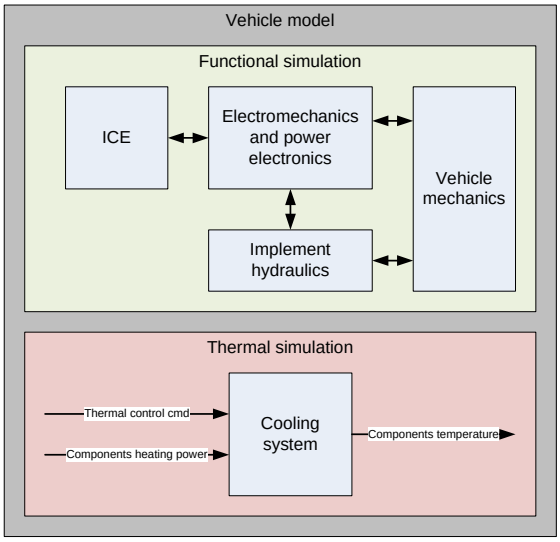

figure 10 Generic vehicle functional model structure

5.2 Modeling guidelines

There are some common modeling guidelines available already. For MATLAB/Simulink there are for example *MAAB* [7], J-MAAB [8], dSPACE guidelines and MISRA rules. Guidelines for achieving compatibility with AUTOSAR architecture already exists for MATLAB/Simulink. It is necessary that some or all those are applied in your projects and further adapted for your need. Guidelines for communication and cooperation policies are needed as well. Mutually agreed and applied guidelines make models understandable, editable, readable and compatible together.

5.3 Communication and coordination

As has been seen in previous chapter *Challenges and problems*, most of the problems arise from poor communication and incomplete overview of the task that is under work. Someone has to be responsible for the overall system architecture before and during modelers do their job. As people tend to avoid complicated things, information has to be easily available and publishable. Not only project manager, but also all of the members need to aware of the status of the project. It is necessary to train and supervise personnel for desired practices and guidelines.

5.4 Assisting software tools

To succeed in all above, tasks can be assisted with proper tools. Those may include version control systems, shared publishing platform (wiki-style), model verification tools and real-time communication channel to handle easily who is

doing what. Use only tools that are easy to use and train.

6 Results

Mentioned actions were taken into use in this project (III). Here is a list of compared projects.

- Earlier project (I): Most of the time approximately 5 active modelers, lots of changes in personnel. Simulation and RCP. Loose specification for models and modeling. Severe module integration problems and lots of remaking.
- Earlier project (II): 3-5 active modelers, some changes. Mostly different people that in project I. Simulation. Strict specification that were not supervised very intensively. Some success and fluent workflow, but not very reusable models and models mostly understandable only by author.
- Current project (III): somewhat variable group, 4-7 active modelers (mostly different people that in previous project I). Simulation and RCP. Strict guidelines based on previous experiences, trained for everyone. Modules were integrated with very light effort and were highly reusable in many project phases and some other tests.

There were quite a lot of changes in personnel, so everything cannot be explained by personal learning of modelers. It is interesting to note, that the results got better even if there were slight increase in modeler team size.

7 Conclusions

Control system software is being developed increasingly in model based methodology due to its dramatic effectiveness in many cases compared to traditional methods. To be fully beneficial in large projects and to unleash full potential of novel development methods, a strictly defined policy is needed to avoid problems in cooperation and model reuse. Proper guidelines increase modelers' productivity and overall awareness of the project status compared to earlier case of lesser controlled projects.

Acknowledgments

This paper is part of an ongoing project *Hybridization of work machines* funded by the Multidisciplinary Institute of Digitalisation and Energy (*MIDE*) of Helsinki University of Technology.

References

[1] Fabbrini, F.; Fusani, M.; Lami, G.: *Software Travels in the Fast Lane: Good news or bad?* ERCIM news magazine number 68. 01/2007. http://ercim-news.ercim.org/content/view/120/263/. accessed on Oct 31st 2008.

[2] Charette, Robert N.: *Why Software Fails*, Spectrum, IEEE online magazine 9/2005. http://spectrum.ieee.org/sep05/1685. Accessed on Oct 31st 2008.

[3] Schäuffele, J.:Zurawka T.: *Automotive Software Engineering.* p. 23. ISBN 3-528-01040-1, Wiesbaden, Wieweg, 2003.

[4] Suomela J., Sainio P., Leivo A., Jakubik P., Degerholm M., Lehmuspelto T., Control System Development for an 8 Wheel Off-Road HEV By Using V-Cycle, The 22nd International Battery, Hybrid and Fuel Cell Electric Vehicle Symposium & Exposition, EVS 22, Yokohama, Japan, October 23 - 28 2006.

[5] Brooks, F.P.: *The Mythical Man*-month, anniversary edition, ISBN 0-201-83595-9, Addison-Wesley, Reading, Mass., 1995.

[6] AUTOSAR Technical overview. http://www.autosar.org/find02.php. Accessed Apr 9th 2009.

[7] Mathworks Automotive Advisory Board: *Control Algorithm Modeling Guidelines Using MATLAB®, Simulink®, and Stateflow® - Version 2.0*. http://www.mathworks.com/industries/auto/maab.html. Accessed Apr 8th 2009.

[8] Japan MATLAB Automotive Advisory Board: PLANT MODELING GUIDELINES USING MATLAB® and Simulink® version 2.1. http://j-maab.cybernet.jp/free/pmwg/english.html. Accessed Apr 9th 2009.

Authors

Antti Leivo received his M. Sc. degree in automation and systems technology from the Helsinki University of Technology, Finland, in 2005. He has been working since 2005 as a researcher at the Helsinki University of Technology. His main research projects are in the field of model based software development of hybrid electric work machines.

Jussi Suomela is senior research scientist and project manager in Department of Automation and Systems Technology in Helsinki University of Technology (HUT). His main research areas are hybrid electric vehicles and field and service robotics. He received his doctoral degree from HUT in 2004.

Ari Hentunen. Hentunen received his M.Sc. degree in electrical engineering from the Helsinki University of Technology, Finland, in 2005. From 2005 to 2007, and since 2008, he has been working as a researcher at the Helsinki University of Technology. During 2007–2008 he worked at Patria Land & Armament as an R&D engineer in the field of model-based software development. His main research projects are in the field of hybrid electric work machines and DC/DC converters.