Assuring Dependability of Consumer Devices: -Automobiles, Consumer Robots, Smart Houses, Avionics, etc-

White Paper

Yutaka Matsuno
University of Tokyo
matsu@cc.u-tokyo.ac.jp

Kenji Taguchi
AIST
kenji.taguchi@aist.go.jp

Yoshihiro Nakabo
AIST
nakabo-yoshihiro@aist.go.jp

Akira Ohata
IPA
a-oohata@ipa.go.jp

15 December 2011

Abstract

Consumer device is a newly coined term which refers to a new category of industrial products used by end users including automobiles, service robots, consumer electronics and smart houses. Unlike traditional industrial machineries, consumer devices are used in diverse, open, and dynamic environments. Furthermore, as accountability of manufacturing companies becomes crucial, they need to assure that their products are dependable whenever required during the development and operational phases. To satisfy these requirements, this paper proposes a process model of simultaneous development of embedded control software and dependability cases [6]. In the process, control software and the dependability case are simultaneously refined and updated. These perspectives are created according to the experiences that Japan has provided highly safe and reliable products with the reasonable prices in the world. These will become more important than before because the complexity will be rapidly progressing such that consumer devices are connecting with the social network including information and energy systems. This paper gives an overview of our effort to standardise the functional safety of consumer devices, their development methodologies and how their dependability is assured.

1 Introduction

Today, every system in our society provides value-added services by interacting each other. Each system shares their sub components, and changes according to various requirements and the environments. They are called “Systems of systems.” Sustaining dependability of them is much harder than that of conventional systems.

1.1 Consumer devices as systems of systems

Consumer devices such as automobiles are now becoming systems of systems as shown in Figure 1. Figure 1 depicts a relationship among four kinds of systems: traffic systems, energy supply systems, information systems, and distribution systems which can be considered as sub systems of consumer devices. They face with various requirements and changes from environments: safety, quality, cleanness, climate changes to traffic systems; efficiency to energy supply system; security to information system; and increasing population to distribution systems. Because these systems interchange each other and share information and resources, it is not easy to satisfy the requirements and adapt to the changes separately: we need to consider them as systems of systems, and find a solution to sustain dependability of the systems as a whole.

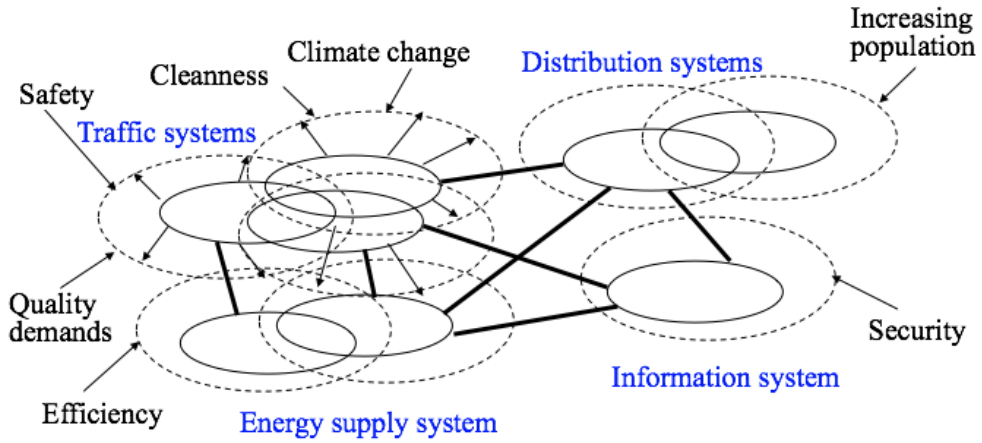

Fig. 1 Automobiles as Systems of Systems

Consumer device is a newly coined term compared to conventional industrial device, used by end users such as automobiles, service robots, consumer electronics, and smart houses. Major differences between consumer and industrial devices are shown in Table 1. The most important thing is that

Table. 1 Comparison of Industrial and consumer devices

	Industrial	Consumer
number	a few to many	massive
user	experts	general users
cost (for each)	very high	low
maintenance frequency	often	seldom
environments	factory environments almost stable	user environments open, dynamic, diverse

consumer devices are used in open, dynamic, and diverse environments. Safety and reliability are highly affected by the environments. For example, a huge number of the users drive their cars under the various conditions from the heavy rain to the blue sky, a rain forest to a dried desert, the Dead sea to the Mexico city level altitudes, and the bumpy to the iced roads.

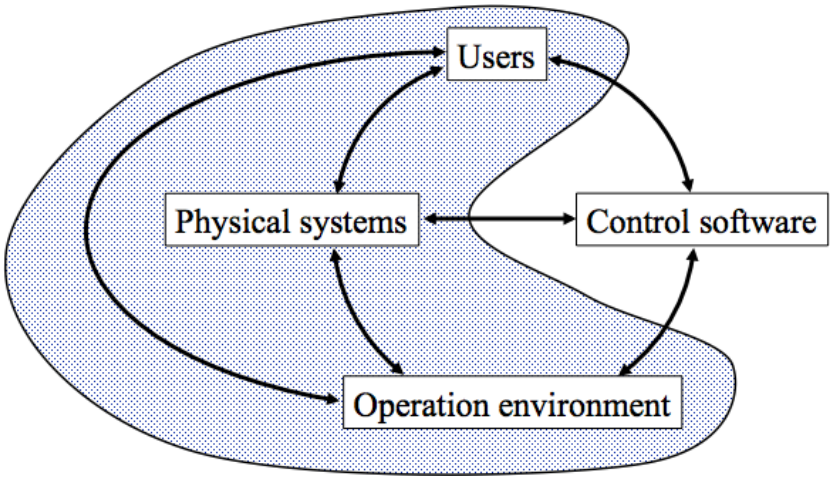

Fig. 2 Frequent Interaction between physical system and software

For dependability of consumer devices, we focus on *embedded control software* which is a crucial component as the connection between software and physical systems, and it has become extremely important for almost all value-added services. There are frequent interactions between control software and the physical system consisting of the controlled objects, the users, and the operation environments as shown in Figure 2. This is the major difference between embedded control software and other software in enterprise systems.

As consumer devices become indispensable to our lives, accountability has emerged as one of the most crucial quality for the companies. Recently it has been reported that some companies might lose the trusted brand image by a small defect, and got difficulty to account for the defect to the public and authorities. Also, as each system is independently developed and operated, communication among various stakeholders inside and outside of systems is very important. For dependability, *risk communication* is crucial.

1.2 Our proposal

Toward achieving dependability of consumer devices, we aim to develop a process for consumer devices. In particular, we focus on development process for embedded control software. The key characteristics of our proposed process are the following.

- To adapt to changes in requirements and environments, the process should be *iterative* and *model-based*. By iterative process, we are always ready to adapt to changes (specially when some failures happen), and update and fix the system as quickly as possible. By model-based development, we can understand (or predict) certain structure and behavior of consumer devices, which are now systems of systems, before implementing them.
- For accountability and risk communication, we focus on *assurance cases* [4], which has been recognized as a key concept to assure safety, reliability, and other crucial system properties of developed system to the stakeholders. Assurance cases are called dependability cases [6] when arguing dependability of systems. Safety cases (assurance cases for safety) have been applied to ISO 26262 [8] for automotive functional safety of electric and electronic system.

As far as we know, rapid iterations with continuous improvements approach for consumer devices are not mentioned in ISO 26262 and other standards for safety in depth. Furthermore, it is very

important to manage unknown factors, because new developments may contain unknown factors and that affects dependability of the system. Rapid iterations with continuous improvements are crucial for dealing with unknown factors. Also, there have been only a few studies focusing on the dependability assurance in rapid iterations and continuous improvements.

These perspectives are created according to the experiences that Japanese manufacturing companies have provided highly safe and reliable products with reasonable prices in the world. These will become more important than before because the complexity will be rapidly progressing such that consumer devices are connecting with the social network including information and energy systems.

The structure of the paper is as follows. In Section 2, we briefly explain technical background of the paper: assurance cases and model based development for embedded control system. In Section 3, we show our proposed process model. Section 4 presents technical details of how physical models can be incorporated into effective embedding of control codes and how they help to ease their verification. In concluding remarks, we show our current standardisation activities of the proposed process.

2 Technical Background

2.1 Assurance Cases

System assurance has become a great importance in many industrial sectors. Safety cases (assurance cases for safety of systems) are required to submit to certification bodies for developing and operating safety critical systems, e. g., automotive, railway, defense, nuclear plants and sea oils. There are several standards, e. g. EUROCONTROL, Rail Yellow Book [11] and MoD Defense Standard 00-56, which mandate the use of safety cases.

There are several definition for *assurance cases* [4]. We show one of such definitions as follows [1].

> "a documented body of evidence that provides a convincing and valid argument that a system is adequately dependable for a given application in a given environment."

Two major graphical notations for assurance cases are *GSN* (Goal Structuring Notation)[9] and *CAE*(Claims, Arguments, and Evidence)[1]. There are two standardisation efforts for assurance cases; The System Assurance Platform Task Force (SysA PTF) at the OMG (Object Management Group) and GSN standardisation effort [7]. OMG has standardized the meta-model for assurance cases called ARM (Argument Metamodel) [10] by which both notations are in fact interchangeable. The main aim of the ARM is to align two major notations and facilitates the tool support.

2.2 Model Based Development (MBD)

The concept of MBD is shown in Figure 3. A control system consists of the controlled object and the electronic control unit (ECU). Combining both, the developed control system is validated through many evaluation tests. The same structure is constructed in the model. Therefore, there are two types of model: the controlled object model and ECU models. Controlled object is called plant in the control engineering area, so we call controlled object models as plant models in this paper.

Plant model is described with the differential and the deference equations. The ECU model is described with the deference equations. The combination of Simulink and Stateflow is widely used for plant and ECU simulation. Plant and ECU models are combined to form a closed loop model, and the model is validate in such simulation tools. This kinds of simulation is called *SILS* (Software In the Loop Simulation).

Interesting things of this environment, there are two links between the real and the virtual world. One link means that the ECU model controls the actual plant. That is called *RPE* (Rapid Prototyping

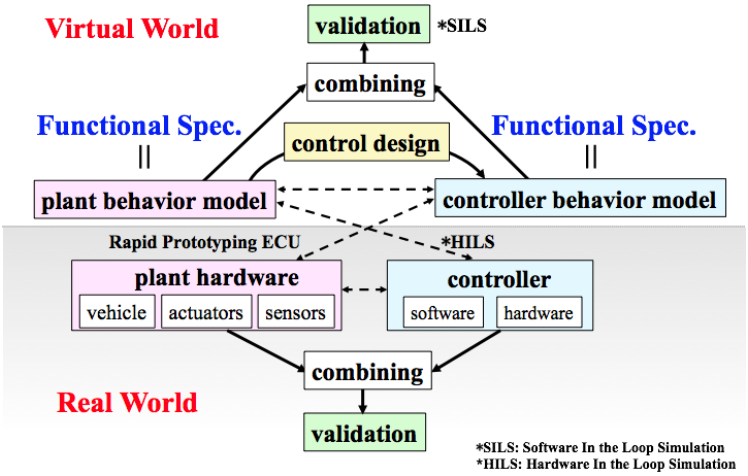

Fig. 3 MBD for Embedded Control Software

ECU). The other link is that the plant model is controlled by the actual ECU. That is called *HILS* (Hardware In the Loop Simulation). It takes time to develop an actual ECU and it has the limitation in execution speed and memory sizes. Thus, RPE is very effective to save time to develop the actual ECU and get a high ECU performance. By HILS, it becomes easy to simulate observed defects and set the test conditions that are dangerous in real testing.

The most important thing of MBD is that both models are used as the functional specifications. Models provide good references of dynamical behaviors and that allows us to accurately verify the outcomes of the development. Moreover, auto-code generation is possible from the ECU model. Recently, the memory efficiency of auto-code generation has reached about 110% compared with the best effort hand codes.

2.2.1 Plant Model

Control theory teaches us that control algorithm is derived when the plant model and the desired behavior are defined under the given constraints. Plant model corresponds to the description of knowledge about the considered plant and the definition of desired behavior corresponds the objectives for control design, such as the criteria in optimal control designs and assigning poles. Unfortunately, current control theories can be applied to only simple plant models. To apply control theories for large scale and complex plants, we need to formally define the plant modeling process to allow us to have a common modeling environment. Figure 4 depicts a picture of such process.

There are two types of plant model in the control engineering area. One is physical model derived from physical laws. The description has the structure reflecting physics. Physical model corresponds to white-box model having the structure consisting of the equations. The other is experimental model corresponds to black-box model based on the function approximation and the system identification theories. The coefficients are determined according to the statistic theory so that the model error is minimized. This type model does not have the structure reflecting physics. It is generally observed in the field that the combination of physical and empirical models is mostly practical.

Difficult issue of plant modeling is that there is no common method and process among researchers and engineers. Even for physical model, the process definition is not clear although modeling methods are well developed in specific physical domains. However, the method combining model components across physical domain boundaries highly depends on the developers. Therefore, it is not easy to reuse developed models by the other developers.

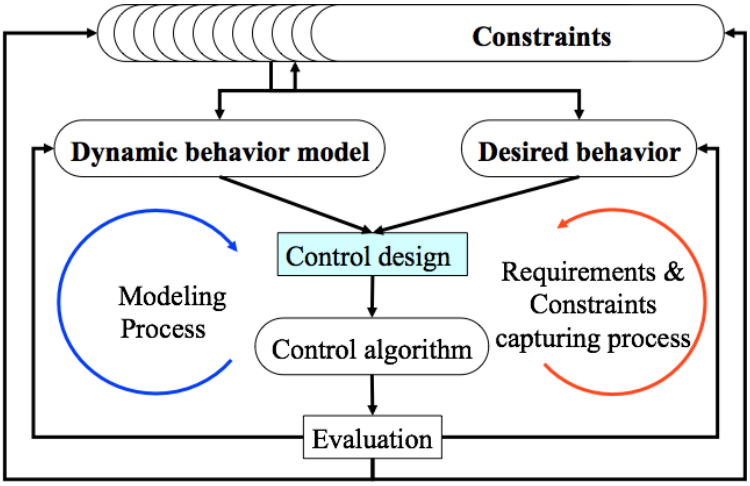

Fig. 4 Iterative process capturing plant models and requirements

2.2.2 ECU model

ECU model determines the relationship between the inputs and the outputs of the ECU. In MBD, ECU model must be executable on the targeted ECU or other computers. In any methods, control models consist of the components called features or modules which have hierarchal structures. Partitions and the interfaces among the components are called the architecture. Unfortunately, theoretical and systematic methods to determine the architecture of large scale and complex ECU have not been well established.

3 Iterative Process for Embedded Control Software

Figure 5 shows our proposed process model in which both embedded control software and dependability cases are simultaneously developed and generated, respectively. The process model consists of two loops: embedded control software development loop and dependability case (and certification document) generation loop. The embedded control software development loop consists of two sub loops: model-based loop and (actual) implementation loop. The loop for dependability cases is based on previous methods such as in [6].

The central idea of this paper is not to create dependability cases after the development but to do so while systems are being developed. Dependability analysis does analyze system properties such as safety, reliability, availability, etc, defined in [5]. The result of this analysis affects goal definitions in the assurance process and objectives and requirements definition/specification in the development process. Requirements and constraints are refined during the development.

Dotted lines in Figure 5 references between the two loops. Evidences for dependability cases are collected from V&V activities in the development process and argumentation may be constructed through the system development.

By simultaneous development of embedded control software and dependability cases, the manufacturing company is able to assure that their products are dependable whenever required, even the development is rapid and iterative. Certification documents are then generated from the dependability cases. ISO/FDIS 26262 partly requires safety cases, but as far as we know there have not been safety/dependability standards which require the certification documents to be generated from assurance cases.

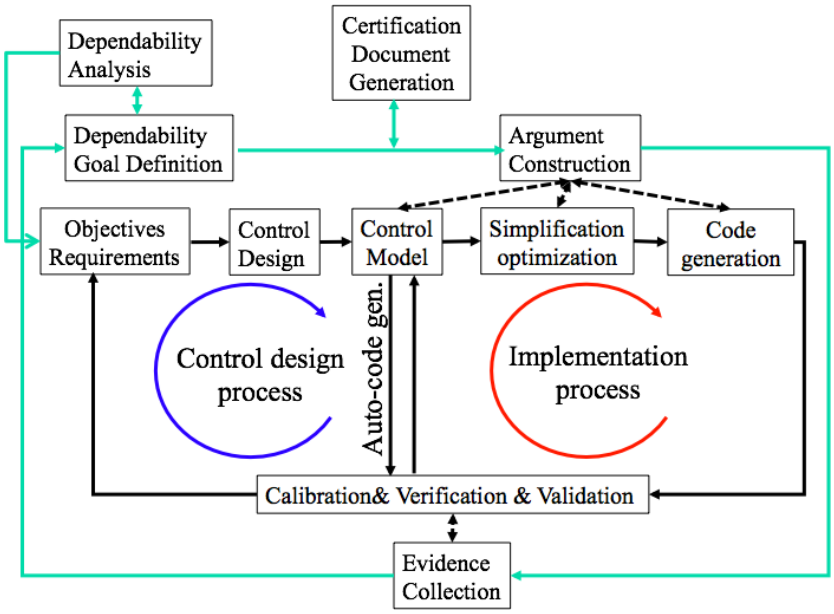

Fig. 5 Simultaneous development of embedded codes and dependability cases

We compare our proposed process model with conventional process model.
A typical software development process is shown in Figure 6. The input of the process is the pro-

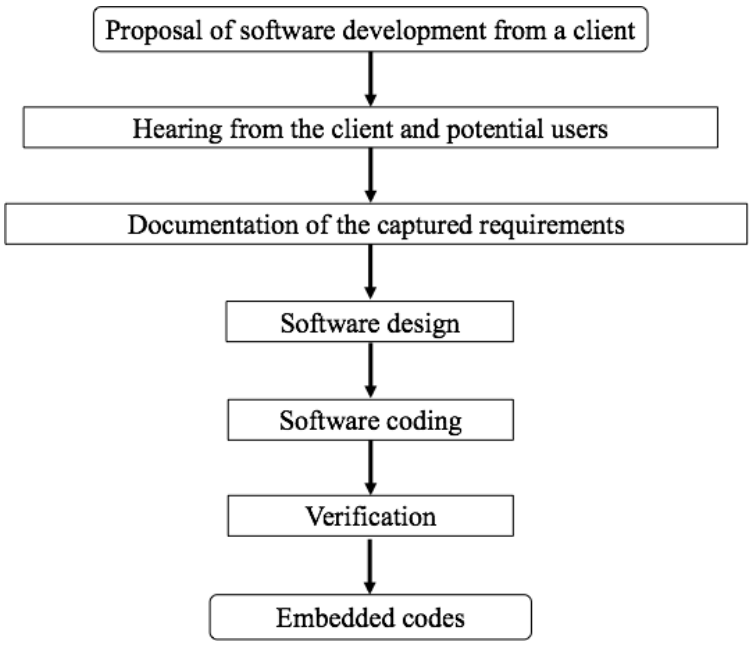

Fig. 6 Software development process

posal of software development from a client and the output is the embedded codes. The first step of the development is hearing from the client and potential users. According to the results, the document of the requirements and the constraints are made. The next step is the software design followed by

coding. The captured requirements and the constraints are sometimes ambiguous and unclear. Thus, the process to capture the requirements and the constraints always problem and software engineers want to know where the requirements and the constraints come from. For the purpose, software engineers intend to create use cases and capture domain knowledge. However, it is usually a tough work for software engineers to do that because it requires deep knowledge and long term disciplines in the specific domain.

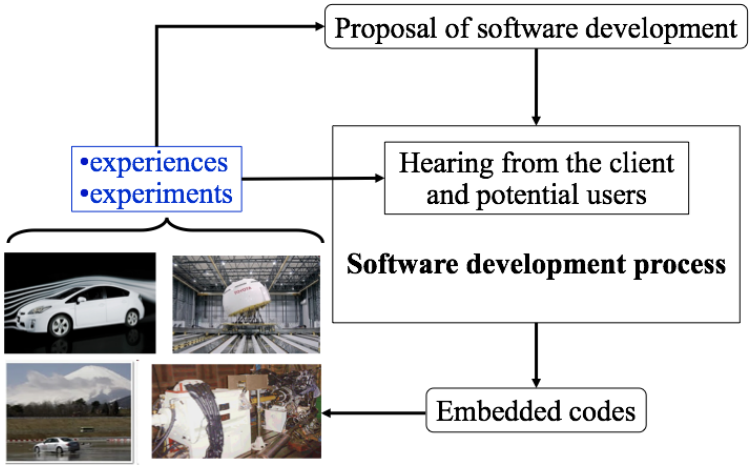

Fig. 7 Entire Software development process

Figure 7 shows the entire software development process of consumer device. The proposal is based on the experiences and the experiments in the client. For example, the automotive manufactures have many test facilities including their test grounds as shown in Figure 7. They need a huge number of tests to know if the developed cars work right under various conditions. If control software includes a defect, the issue is immediately fixed because many cars drive in various situations and environments once they are put into the market place. Therefore, the car manufactures must sufficiently evaluate safety and reliability of automobiles through a huge number of tests before the production. Embedded codes are necessary for the test because the automobiles can not drive without control software, therefore the process should be iterated.

In this context, it is obvious that requirements and the constraints are captured during the system development and the iterated works are completed at the end of the development. It can be said that a new development surely includes unknown factors and development means the process to clarify the essential unknown factors. That means the process shown in Figure 6 is only a part of the system development.

Figure 8 shows a current embedded control software development process. The required embedded codes are developed during the iterative control design and the specification is made when the process is completed. Usually, the specification is transferred to an ECU supplier that develops the final embedded codes. The codes are calibrated and validated through actual tests. Therefore, coding is performed in two different phases. The former coding is in the advanced development and the latter coding is in the production development. The specification is made from the embedded codes developed in the advanced development.

Figure 9 shows a typical process proposed by software engineers. The specification is made before coding. But, it is not easy to make the specification that includes all requirements and constraints as mentioned above. Thus, this process must contain iterations. A headache of software engineer is that control parameters are determined in the actual tests of the calibration process and the values affect

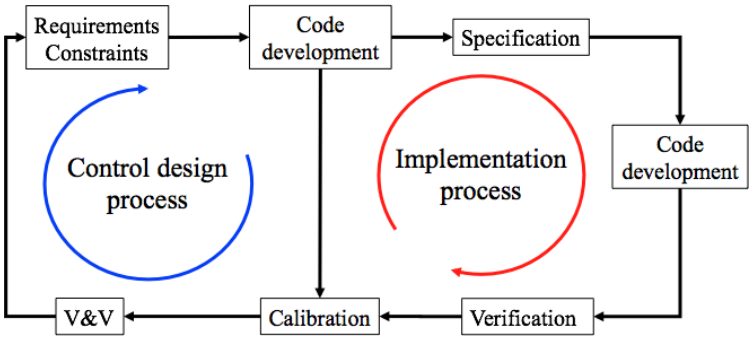

Fig. 8 Current embedded control software development process

codes. That means iteration loops are inevitable also in the implementation process. This observation shows us that the process in Figure 9 does not faithfully reflect on actual development process

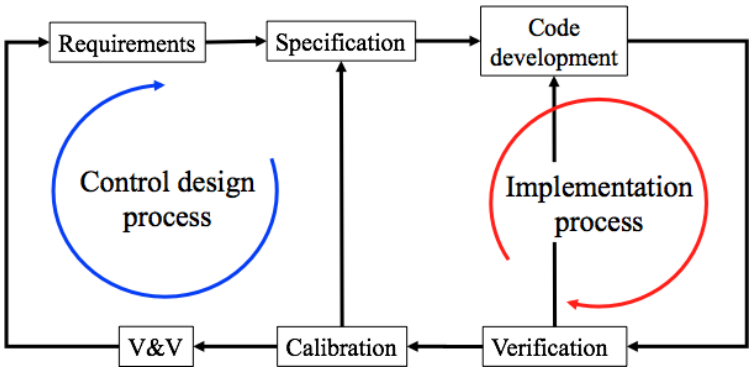

Fig. 9 Development process proposed by software engineers

Recently, Model-Based Development (MBD) based on state of art simulation technologies has been proposed. In MBD, models mean the dynamic behavior models of the controlled object described by differential and difference eauations, that is called plant models in control engineering, and the ECU described with the deference equations. From the ECU model, the embedded codes are automatically generated. That is called Auto-Code Generation (ACG). Simulink combined with Stateflow is a popular tool for the model description. Modelica has been getting popular for physical modeling applied to the description of controlled object dynamic behavior. We can realize the MBD process shown in Figure 10 once we have the models developed in the high level.

In MBD process, the control process and the implementation process are separated to each other. Control model describes the dynamic ECU behavior and the embedded codes are generated by ACG. The control design process is highly enhanced by SILS (Software in the Loop Simulation), HILS (Hardware in the Simulation), and PILS (Processor in the Loop Simulation). Thus, the required experiments are immediately possible. The following verification and the validation are also possible and that allows us rapid iterations. The control model is transferred to the implementation process as the executable specification after it is completed. The implementation is defined as the recreation on the target ECU of the relationship between the inputs and the outputs defined by the control model. The final code should be simplified and optimized in the ROM/RAM memory sizes and the execution time.

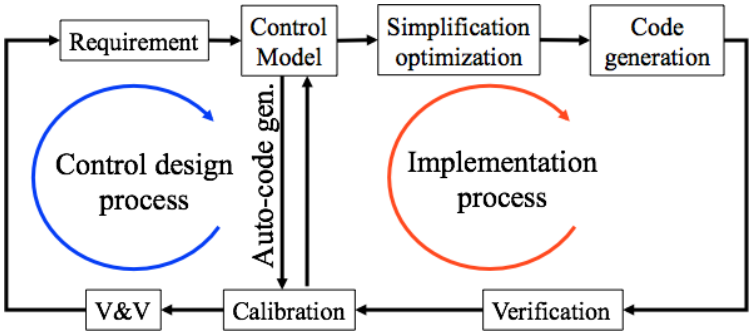

Fig. 10 MBD control design process

4 Physics in software

Calibration and validation are placed in the latter part of the development. It is very difficult to change control embedded codes around the final stage of the development because the evaluations of the impact changing codes require very time consuming works. However, for instance of engine control developments, they have a method to mitigate this issue. Engine control software has many maps. It gives a flexibility of fixing defects without changing codes to the automotive industry. In the case of a single task, the control logic is described with

$$\frac{dx}{dt} = f(x(k), u(k)), \tag{1}$$

where $x = [x_c, x_d]$ is state variables and $u = [u_c, u_d]$is the input variables of the control algorithm. The suffix c means continuous event variables and the suffix d means the discrete event variables including logical variables. The right hand term can be approximated by a map because it is a static function. Map tend to be the high dimensional maps difficult to implement. Thus, the automotive industry approximates the map with the combination of low dimensional maps and simple functions. The big advantage of this method is that (1) can theoretically describe any behaviors. Therefore, the maps approach can provides the flexibility even if they do not well notice that. We can enhance the flexibility if we aggressively apply the map technology.

They say that it is very difficult to determine the values of the maps. This is true and the automotive industry spends much time to do it. To determine the map through experiments is called calibration. But, recently, they introduces model-based technology into the calibration process. Plant models can mitigate the calibration efforts considerably. Plant models and the ECU model can be extensively used for the purpose.

For example, engines behave as continuous event systems. Thus, the control can be continuous. But, it is difficult to distinguish the portions of continuous and discrete events as software behaves discrete event system basically. But, these variables reflecting physics and the variables with sufficient resolutions are considered as continuous event variables.

There are many logical branches like the C-code in Figure 11. It can be considered as an approximation of required continuous logic as shown in the right side of Figure 11. Figure 12 shows the comparison between the logic and the physically desired curve.

The combined system of continuous and discrete systems is called hybrid system. Almost all system have hybrid features. The condition of logical branch divides the state space into two portions. That means the divided portions are exponentially increasing with the logical branches. Discrete events

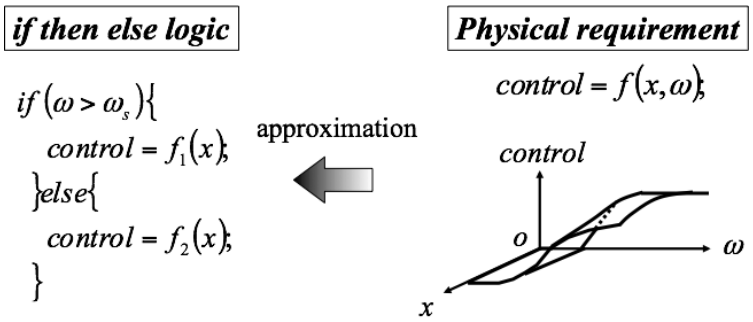

Fig. 11 logical branch and physical phenomena

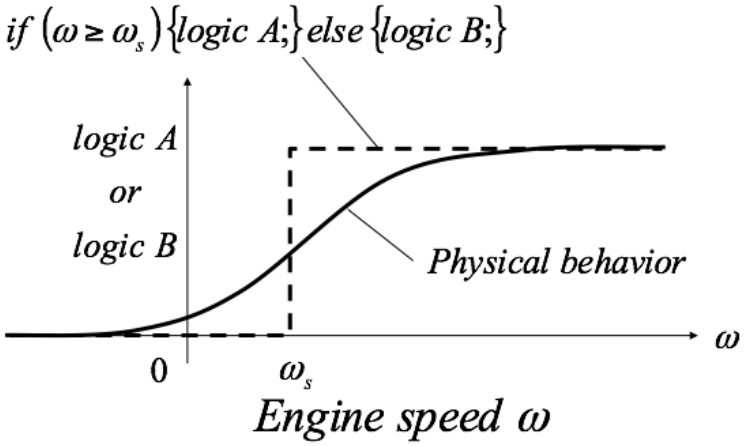

Fig. 12 Example of logical branch in automotive engine control

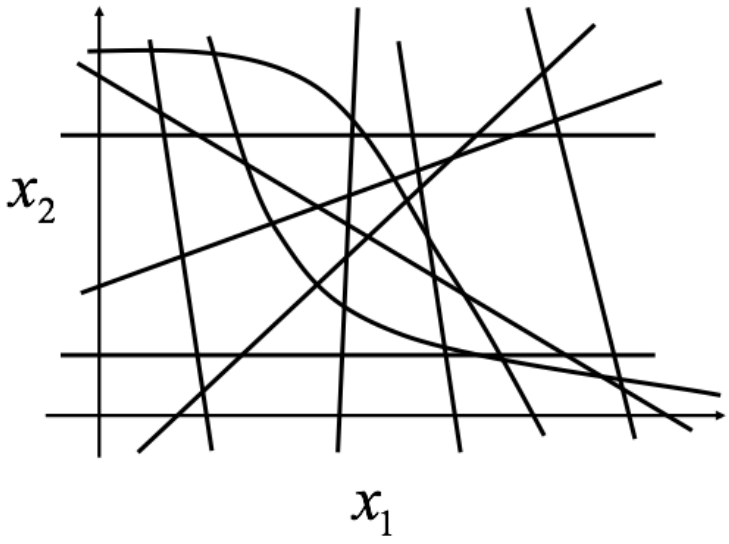

Fig. 13 dividing the state space by logical branches

make verification very difficult as the state space is divided into the many pieces by the logical branches as shown by the dotted line in Figure 13 is replaced by the solid line corresponding continuous event. Evaluation needs a huge number of the tests such that a divided portion is evaluated by the trajectory going through the divided portions.

Thus, the reduction of the logical branches is very effective to mitigate the difficulty. For the purpose, an idea is that the logical branch shown in Figure 12 is regarded as the corresponding continuous event. The logical branch can be transformed to the continuous system or neglected in the verification process. Thus, we have to distinguish the continuous and discrete event portions in software.

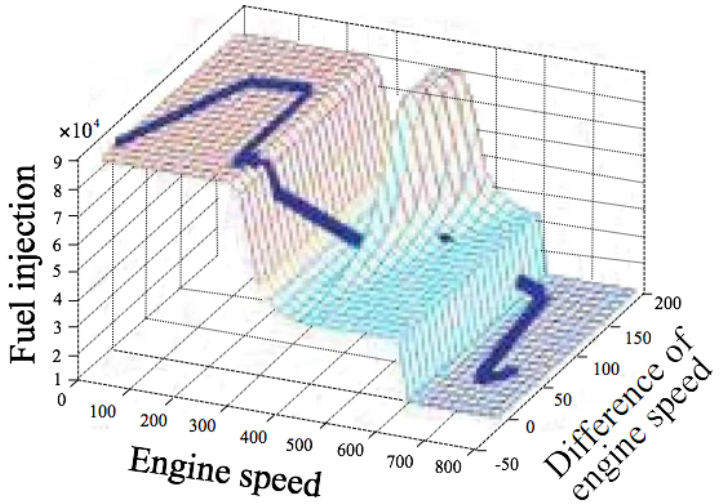

Fig. 14 Example of verification based on physical considerations

An example of the verification based on the concept is shown in the Figure 14. The application is a prototyping engine cold start control. It has many logical branches and it is very difficult to catch the physical background of the code. This feature has resulted in the many logical branches inserted by many developers. The figure in Figure 14 shows the sections of the state space. The steep decrease of the fuel injection is seen in the left figure and the isolated mountain appears in the right figure. That is very strange from the physical point of view although almost all trajectories do not go up the mountain, fortunately. However, the figures tell us that the codes should be revised.

5 Concluding Remarks: Our OMG Standardisation Activities

Iterative and simultaneous development of embedded control software and dependability cases for consumer products used by general users has been proposed. Dependability cases are refined during the development and completed at the end of the development. The documents submitted for certification is generated from the final dependability cases. In order to adapt rapid iterations to dependability standard, dynamic behavior models are used as the functional specification of the controlled object and the ECU. Shared modeling environment of the controlled object is one of the most important challenges.

We have already organized a special seminar at OMG [2] and discussed on systems assurance and safety for consumer devices. As a result of the discussion, we have decided to start a standardization process in OMG Systems Assurance Platform Task Force (SysA PTF) [3] and, firstly, we will publish a Request For Information (RFI) hopefully in this year. Anybody can reply to the OMG RFI and Request

for Proposal (RFP) will be published based on the whitepaper replied to the RFI. The RFP denotes what should be standardized at OMG. By the reply to the RFP, an OMG standard will be published finally and can be used for tool supports, process definition and/or others. We expect various information from many different domain areas.

References

[1] http://www.adelard.com/web/hnav/ASCE/choosing-asce/cae.html.

[2] http://www.omg.org/news/meetings /tc/agendas/ut/SysA_info_day.htm.

[3] http://sysa.omg.org/.

[4] *Workshop on Assurance Cases: Best Practices,Possible Obstacles, and Future Opportunities, DSN 2004*, 2004.

[5] Algirdas Avizienis, Jean-Claude Laprie, Brian Randell, and Carl Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Trans. Dependable Secur. Comput.*, 1(1):11–33, January 2004.

[6] Georgious Despotou. *Managing the Evolution of Dependability Cases for Systems of Systems*. PhD thesis, Department of Computer Science, University of York, 2007.

[7] GSN contributors. GSN standard version 1.0, 2011.

[8] ISO. ISO 26262 road vehicle - functional safety -, part 1 to part 10. Technical report, 2010.

[9] Tim Kelly and Rob Weaver. The goal structuring notation - a safety argument notation. In *Proc. of the Dependable Systems and Networks 2004, Workshop on Assurance Cases*, 2004.

[10] OMG. Argument metamodel (ARM). OMG Document Number Sysa/10-03-15.

[11] Railtrack. Yellow book 3. Engineering Safety Management Issue3, Vol. 1, Vol. 2, 2000.