UNIVERSITY of BRADFORD

Library

The University of Bradford Institutional Repository

http://bradscholars.brad.ac.uk

This work is made available online in accordance with publisher policies. Please refer to the repository record for this item and our Policy Document available from the repository home page for further information.

To see the final version of this work please visit the publisher's website. Available access to the published online version may require a subscription.

Link to conference webpage: *http://cse.stfx.ca/~CPSCom2017/acceptedlist.php*

Citation: Lefticaru R, Konur S, Yildirim U, Uddin A, Campean F and Gheorghe M (2017) Towards an Integrated Approach to Verification and Model-Based Testing in System Engineering. Presented at: The International Workshop on Engineering Data- & Model-driven Applications (EDMA-2017) within the IEEE International Conference on Cyber, Physical and Social Computing (CPSCom) June 21-23, 2017, Exeter UK.

Copyright statement: © 2017 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

Towards an Integrated Approach to Verification and Model-Based Testing in System Engineering

Raluca Lefticaru[1], Savas Konur[1], Unal Yildirim[2], Amad Uddin[2], Felician Campean[2], Marian Gheorghe[1]
[1] School of Electrical Engineering and Computer Science, University of Bradford, UK
[2] School of Engineering, University of Bradford, UK
{R.Lefticaru, S.Konur, U.Yildirim1, A.Uddin3, F.Campean, M.Gheorghe}@bradford.ac.uk

***Abstract*— Engineering design in general and system design of embedded software have a direct impact on the final engineering product and the software implementation, respectively. Guaranteeing that the models utilised meet the specified requirements is beneficial in detecting misbehaviour and software flaws. This requires an integrated approach, combining verification and model-based testing methodology and notations and methods from system engineering and software engineering. In this paper, we propose a model-based approach integrating various notations utilised in the functional design of complex systems with formal verification and testing. We illustrate our approach on the cruise control system of an e-Bike case study.**

***Keywords*— design engineering, system engineering, verification, model checking, model-based testing, electrical bike, cruise control**

I. Introduction

Evolving customer need and trends, with emphasis towards servitisation and increased levels of autonomy of technical systems, raise new multidisciplinary complexity challenges that need to be addressed in an integrated way [15]. Companies are striving to develop highly interactive engineered systems by shifting and enhancing their product development process from monodisciplinary to multidisciplinary and interdisciplinary working environments [32]. Such an approach is required to integrate diverse technological systems embedded with finite multidisciplinary features that work together to deliver expected requirements and functions articulated by designers for the whole system [24].

Interdisciplinary system design and development require coherent harmonization of existing functional and structural methods [34], and information technology solutions that are often shared across multi-disciplinary teams ranging from conceptual design to design verification and testing phases [10]. To achieve this, product development organisations are striving to (1) transition from classic-documentation based practices to model-based (i.e. object-oriented based) practices in order to minimize re-work and to establish structured formalism on system modelling concepts between multidisciplinary teams in design phase; (2) conduct systematic verification and testing of system requirements captured through the design phase in the context of appropriate system level.

The challenge of interdisciplinary system development is further raised when the system verification is expected to cover evolving real world operational uses. This requires design and analysis of new features along with already existing and embedded features in the legacy system and its architecture. This enforces the OEMs (Original Equipment Manufacturers) to test and verify the system in a transdisciplinary environment.

Companies are constantly striving towards the establishment of such interdisciplinary system development environment that could work top-down and left-right across system's decomposition and integration levels (i.e., system, subsystem and component levels), as illustrated in Fig. 1 in the context of a systems engineering V-Model, which also sets the context for the work presented in this paper.

The problem this paper is specifically aiming to address is the lack of integration between disciplinary approaches to model based design verification and testing.

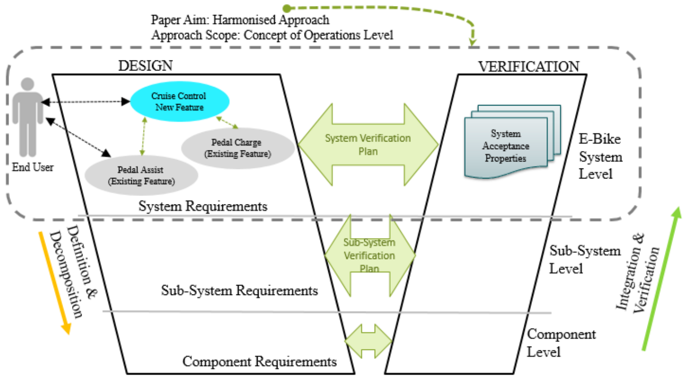
Paper Aim: Harmonised Approach
Approach Scope: Concept of Operations Level
DESIGN
VERIFICATION
End User
Cruise Control New Feature
Pedal Charge (Existing Feature)
Pedal Assist (Existing Feature)
System Verification Plan
System Acceptance Properties
E-Bike System Level
System Requirements
Definition & Decomposition
Sub-System Verification Plan
Sub-System Level
Sub-System Requirements
Integration & Verification
Component Level
Component Requirements

Fig. 1. Paper scope in the context of V-model

From the perspective of the electro-mechanical domain, the robust engineering design framework employed by the automotive industry [16] provides the framework for developing verification and testing methods and carrying out design verification. This methodology is underpinned by the philosophy of including operational noise factors into the physical testing at component, subsystem and system level – on the right side of the systems engineering V (illustrated in Fig. 1). While this methodology discusses the opportunity to use models of the system for the purpose of design verification, this largely relates to physics based simulation models of components or subsystems, and it does not address the verification of the control logic of the systems or the software components or subsystems.

On the other hand, rigorous methodologies for software verification and validation, underpinned by formal methods [39] have been developed and extensively used for software systems within an industrial context and even applied to earlier stages of specification and design. Formal methods are used in all stages of the software project development underlying verification of the models used and formal aspects of software testing [17]. A challenge remains the integration of formal methods with various other methods and notations utilized. There are initiatives in various application to achieve certain level of integration – avionics [30], automotive [22].

The design paradigm common in industries, such as the automotive, is that advanced control features (such as Advanced Driver Assist Systems – ADAS) are introduced to enhance comfort, safety and efficiency of the overall system – including the user and the machine. Such features are delivered by advanced sensing technologies deployed within the system, and software control systems to integrate the functionality of the feature within the whole system, delivering the customer expected functionality. A systems engineering approach is required to deliver the integration of the new feature functionality within the whole system, which includes many legacy physical systems features. Of particular interest is the verification and validation of the new features – focusing on both logic (to deliver the required user interaction) and behaviour in the context of the real world operation of the system, which might include significant uncertainty.

The research behind this paper reflects an interdisciplinary effort to address this problem, by combining engineering design analysis with formal methods for system verification and model checking / testing, within a systems engineering environment. This paper provides our initial approach towards an integrated method that uses verification and model-based testing. Our approach is illustrated by a case study based on the design analysis of an electric bicycle (e-Bike), which introduces new ADAS features, as described in [4]. The analysis is in particular exemplified in relation to a cruise control feature (CC) for the e-Bike system. This case study presents a coarse grain view of the system that does not include low-level components with their behaviour. This is in accordance with the verification and testing method proposed in this paper.

Using the cruise control system of our e-Bike case study, we present a high level approach that verifies the system's expected behaviour captured through current design methods in operational context with the end user (such as use case and state diagrams), as depicted in Fig. 1. The verification is done through the use of verification tools that check and identify *all possible system behaviours* (both expected and unexpected, i.e., escaped properties). The validation of any implementation is obtained through model-based testing and reveals both structural and component behaviour errors.

The article is organized as follows. Section II provides a brief overview of the case study, design methods for system model development and system verification and testing methods. We then describe the generic steps of the developed approach in Section III, which is followed by verification and testing analysis, in Sections IV and V, respectively. Discussion and conclusions are presented in the end.

II. Background / Context

A. *E-Bike System with Rider Assist Features*

The e-Bike system considered as case study in this paper, described in [4], is based on brushless hub motor that has the capability to work as a generator as well as a motor. The propulsion system of e-Bike combines an electric drive system with a conventional pedal drive. This affords a variety of ways of controlling the propulsion of the e-Bike, delivering enhanced options for the rider to interact with and control the system, enabling a variety of user activities to be supported. As an example, a cruise control feature can be introduced. A cruise control is an ADAS technology that automatically controls the speed of a transportation system (such as motor vehicle or electric-bike) set by the user [26]. This feature adds value (in relation to comfort and safety) to the user by simplifying the driving task in relation to the longitudinal dynamics, so that the driver has more time to concentrate on traffic and other aspects of the navigation. For an e-Bike system a cruise control feature would provide the user with enhanced control of the journey time, while also controlling the level of exercise undertaken. To deliver the CC feature, the electric drive system needs to adjust the torque input such that the velocity of the bicycle is maintained, regardless of the effort input from the rider.

From a system design point of view, adding new functionality and validating the operational safety of new technology in the legacy system (e.g., motor vehicle or e-Bike) increases development time [35]. Thus validation of additional functionalities of a new technology and its impact across other technologies of an existing system via system behavioural modelling would be time and cost saving for an organization.

In this paper, we will model cruise control technology for the e-Bike system via system modelling methods and verify its modelled behaviour via system verification and testing methods.

B. *System Modelling Methods*

SysML sequence diagrams are widely used in the representation of scenarios of interaction for a system use case. The use of UML/SysML sequence diagrams on test case generation and software reliability [9, 29, 36, 37, 43] has been the subject of intensive research efforts in recent years. For example, [43] used sequence diagram in the modelling of a test case for a use case of a hybrid powertrain, while [26] represents interactions between elements of cruise control system of an automobile by UML sequence diagrams. Conventional sequence diagrams are more suitable for functional requirements capture for systems based on signal or information exchanges. However, they are insufficient in the extraction of exchange based functional requirements for the analysis of multidisciplinary systems. Both textual and graphical diagrams have been developed to overcome such limitations. Zheng et al. [41] have discussed a comprehensive methodology for multidisciplinary design of mechatronic systems based on an interface model, while [33] has introduced a textual template for enhanced interface analysis that integrates operational-based and exchange-based interactions for comprehensive capture of functional and non-functional requirements of a system in its external environment.

The common graphical modelling languages UML and SysML state machines, commonly employed for the model-based representation of multidisciplinary systems [1], are based on Statecharts [13]. [35] uses SysML state machine diagrams in the representation of functional state transitions of the cruise control of a vehicle. As noted by [3], semantics and syntax of statecharts and thus UML/SysML state machines are limited in the modelling of complex systems.

Other related recent work includes the development of an Enhanced Sequence Diagram [5] and the System State Flow Diagram [40] function modelling framework. Both methodologies provide strong support for flow based function modelling of complex systems with multiple operational model, with rigorous representations that facilitate further development of semi-formal specifications for the system analysed.

C. System Modelling Methods

Systems are becoming more integrated and complex. Engineers deal with such complexity by building and integrating smaller devices / subsystems that are interoperable within a larger system. This hierarchical approach is increasingly being used in engineering and technology applications, including aviation, automotive, mobile phones, security systems, medical devices, nuclear power systems, manufacturing systems, etc.

It is imperative to detect any design or software flaws in the early stages of system development as failure to do so may lead to the collapse of the entire system. One way of doing that is guaranteeing that the design/system meets the desired requirements. In this respect, verification and testing haven been used extensively in the last two decades in various systems.

Here, due to space limitation, we only report some similar work applying verification and testing in a similar context: [22, 25] provide an integrated methodology for formal verification and model-based testing of automotive embedded systems. The authors introduce an architectural description language that is supported by timed-automata based verification and model-based testing against functional requirements. [28] presents a toolset developed for modelling, verifying and testing of software specifications written in a dedicated language. The toolset employs several third-party verification tools. The approach has been applied to a simple cruise control system case study.

Our integrated approach starts from the V model, mentioned earlier in the paper, and shows how various notations used in system engineering, such as UML/SysML [1]. In our investigation we refer to use case and state diagrams (statecharts) – capturing high level specification of the system and design structure matrix [11] providing key relations amongst various functions, are complemented by a formal model, called X-machine [18], describing in a rigorous way the behaviour captured by the previous diagrams and textual descriptions. A verification mechanism is then developed to formally verify high-level description of the system, as presented by the X-machine model. This rather coarse grain verification mechanism, based on the use of a verification tool (NuSMV [6]), helps checking that key requirements are present in the model, which is rather generic and ignores low-level details, e.g. time, continuous behaviour and component details. The validation aspects are covered by a model-based testing approach, directly derived from the X-machine model [18, 19]. This testing methodology allows to check that every component is implemented according to the model provided based on certain input and internal values associated with it and generates a test sequence that reveals implementation errors, exemplified by using a tool [23] generating JUnit test cases.

III. Our Methodology For Integrated Approach – E-Bike Case Study

The methodology presented in this paper is coherent with the structure systems engineering of V-Model in Fig. 1 with explicit relationships between the process steps (left side) and the validation steps (right side). The first step of the approach is to document system functional requirements as a state transition diagram in respect of relevant system use case. The second step focuses on the verification and validation of these requirements and introduces a formal model facilitating this process.

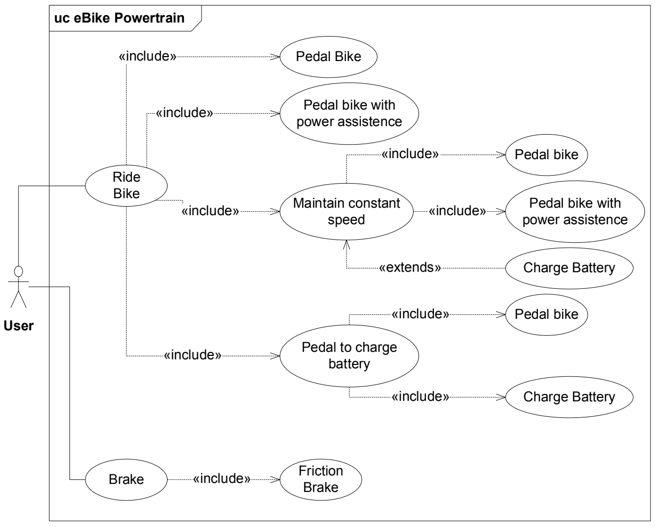

Fig. 2. Use cases in relation to the control of the e-Bike Powertrain by the user

This paper will explain the principles of the integrated approach on the basis of a generic electric bicycle (e-Bike) with a strong focus on the cruise control feature (CC). An electric bicycle can be used just like a regular bicycle by pedalling only. The rider can also take advantage of combined human plus electric power by controlling the level of assistance received from the bicycle [27]. The battery of the bike can be recharged whilst pedalling or riding downhill. While riding the bike, the user can maintain constant velocity of travel, regardless of his / her pedalling input and road conditions by using cruise control feature. Fig. 2 represents these use cases in a Use Case Diagram.

Fig. 2 represents five use cases of the e-Bike as (i) pedal bike (Pedal Only – PO, for short), (ii) pedal bike with power assistance (Pedal Assist - PA), (iii) maintain constant speed (Cruise Control - CC), (iv) pedal to charge battery (Pedal Charge - PC) and (v) brake (Brake - Br). In the sequel, for various models and descriptions, we will use either full names or abbreviations of these use cases.

The cruise control use case "scenario" (which details the process view of the user interaction with the e-Bike) is summarised as follows:

1. When the e-Bike has reached a velocity that the user wants to maintain, the user activates the CC feature (switches to CC ON).

2. The CC feature acquires and stores the current velocity of the e-Bike.

3. The CC feature confirms to the user the activation status (CC "ON").

4. The CC feature continuously acquires the current velocity of the e-Bike, calculates the deviation from the target, and generates and transmits a control signal for the e-Bike (EDS) to correct the propulsion system torque output, generating an acceleration that will bring / keep the velocity within the target range.

5. The user requests to terminate the cruise control feature (switch to CC OFF).

6. The CC feature confirms the status (CC "OFF").

Various dependencies can be captured between the e-Bike use cases in Fig. 2. For example, the bicycle cannot go straight to CC mode from Br mode and CC mode can be initiated from PO mode. Fig. 3 represents possible linkages between e-Bike use cases in Fig. 2 as a Design Structure Matrix (DSM) [11]. A DSM is a matrix representation associated with a network graph of the design, which is often used by engineering design teams to capture dependencies or linkages between the design elements of the system (which would be represented as vertices in a network representation). In this case, the DSM shows the possible transitions between the modes of operation, which are systematically considered in a pairwise (row to column) manner; e.g. from Cruise Control transitions are only possible to Pedal Only and Pedal Assist modes.

	Pedal Only	**Pedal Assist**	**Cruise Control**	**Pedal Charge**	**Brake**
Pedal Only	▨	X	X	X	X
Pedal Assist	X	▨	X		X
Cruise Control	X	X	▨		
Pedal Charge	X			▨	X
Brake					▨

Fig. 3. Dependencies between e-Bike use cases via Design Structure Matrix

The strength of the DSM method is that it systematically evaluates all possible combinations of operations in order to capture the logic of the control system for the e-Bike. Fig. 4 represents multiple possible state transitions for the e-Bike ride mode as a statechart [14], a state diagram that provides the basis of UML and SysML state machines [1].

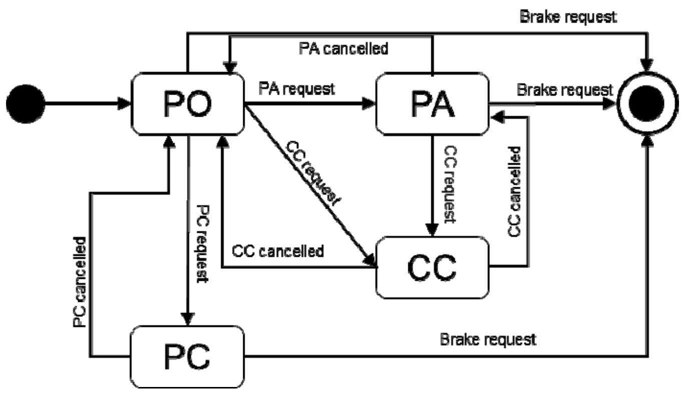

Fig. 4. Statechart of the e-Bike

From the Cruise Control point of view, Fig. 4 illustrates three key state transitions:

- The user should be able to request / activate CC from PO or PA;
- If the CC is cancelled, the system should normally return to the state from where it was activated – i.e., PO or PA, respectively;
- The system should not transit from CC to the output state directly; e.g., when the user brakes, the system returns to PA/PO before jumping to the output state (also identified as Br).

Table I summarizes state transitions in Fig. 4 for four distinct cases as a table: PA request, CC request, CC cancel and Brake request in CC mode. For example, the third row shows that if the CC is cancelled, the bike returns to the PA state.

TABLE I. CC FEATURE STATE TRANSITION TABLE

User action/ request	**Current State**	**Input**	**Previous State**	**Next State**	**EDS Actions**
PA request	PO	PA	PO	PA	EDS in PA mode
CC request	PA	CC (On)	PO	CC (On)	EDS in CC mode
CC cancel	CC (On)	CC (Off)	PA	PA	EDS returns to PA mode
Brake request (BR) in CC mode	CC (On)	BR	PO/PA	PO/PA	EDS off

The diagrammatic and matrix notations introduced above for specifying the problem are complemented by a formal model allowing to rigorously define the behaviour of the state based model introduced above (Fig. 4).

The formal model used in this paper has been extensively applied in software engineering and model-based testing [18].

This model, called X-machine, allows to formally define functions associated to events and actions attached to the state machine transitions. This approach is very suitable for a state based model [40] where pieces of functionality have been already identified. These functions appear as

f: In x Mem → Out x Mem

where In and Out are the sets of inputs and outputs, respectively; Mem is a set of internal values, called memory values. The set of inputs is

In = {"pa", "cc", "pc", "br", "pac", "ccc", "pcc"}

representing the possible events or requests, e.g., "pa" = PA request, "br" = Brake request, "ccc" = CC cancel etc. The set Out is not utilized in this paper, hence it won't be present in the subsequent description. Memory values will be aggregated out of three distinct elements, defining the current state (set denoted by State), the state preceding CC activation (StateBeforeCC), and the EngRun set indicating whether the engine is off or running – with values False, when in states Br (final state) or PO, and True otherwise. Hence, we have

Mem = State x StateBeforeCC x EngRun.

When some of the component values are not necessary then this will be denoted by '_'. We illustrate the use of the model for specifying the behaviour associated with two of the transitions in Figure 4. The transitions from PA and PO to CC are both denoted by CCrequest. This is triggered when a "cc" input (a CC request) is received. The context of triggering CCrequest is given by the current internal values (memory values), consisting of the current state, either PA or PO, and the status of the engine, True in PA and False in PO. When the transition is executed it affects the memory values by indicating the arrival state, CC, the preceding one (either PA or PO) and the status of the engine. Formally, we define

CCrequest("cc", (PA, _, True)) = (CC, PA, True)

CCrequest("cc", (PO, _, False)) =(CC, PO, True).

When CCcancelled is triggered, by an event "ccc" or "br", i.e., a request for cancelling the Cruise Control or for Brake, then the control is returned to either PO or PA depending on the memory value associated with StateBeforeCC and changing adequately EngRun. CCcancelled will be replaced by CCcancelledtoPO and CCcancelledtoPA as the current state machine is a non-deterministic automaton and for testing purposes we need a deterministic one [18, 19], unless we change it to an equivalent deterministic machine [20]. The same deterministic automaton will be used for the formal verification. These will be discussed in Sections IV and V. The complete definition of the X-machine model can be found at [38].

IV. VERIFICATION

We can analyse the system to check if it works according to the design requirements using simulation and testing. Both approaches, however, can only analyse a limited subset of *all possible behaviours*, and hence cannot provide an *assurance* that the system in question works *correctly* and any undesired behaviour will *not happen*.

A typical approach to solve this issue is using *formal verification* to perform exhaustive analysis, instead of using simulation or testing. One particular formal verification method is *model-checking* [8], which is an algorithmic approach to verification. Model checking receives a mathematical model of the system and a requirement, expressed in a suitable formal logic, and checks if this formal requirement is verified by *exhaustively* exploring *all system behaviours*, captured by the mathematical model. If the answer is '*yes*', then we can say that the requirement is verified in all possible situations. Otherwise, the so-called "*counter example*" is generated, which shows some possible behaviour of the system that does not satisfy the given requirement. This allows the designer/modeller to change the system accordingly to fix the error.

In model checking, requirements are specified in suitable formal logics, in particular *temporal logics*. Two well-known and most widely used temporal logics are *Linear Temporal Logic (LTL)* [31] and *Computational Tree Logic (CTL)* [7]. Both logics can capture dynamic behaviour of systems evolving over time. LTL does that by adopting linear interpretation of time (i.e. one possible behaviour at a time); whereas CTL adopts branching semantics (i.e. multiple behaviours that can occur at the same time).

Various tools have been developed to support model checking, which allows the automatic verification of properties. NuSMV [6] is one of the most used model checking tools, which was originally designed to verify reactive systems. NuSMV has its own modelling language, which is based on finite state machines. The tool supports the verification of both LTL and CTL properties. NuSMV employs *symboli*c methods, allowing a compact representation of the state space to increase the efficiency and performance.

In order to proceed with the formal verification stage we need first to map the formal X-machine model, a state based approach, onto a NuSMV representation. Table II shows how functions CCrequest and CCcancelled (with its variants CCcancelledtoPA and CCcancelledtoPO) are translated into NuSMV code. The code lines corresponding to them are 15, 20 (CC request from PO / PA), and 21-24 (CCcancelled to PA or to PO, for both events, "ccc" or "br").

For the NuSMV specification briefly described in Table II, we have verified some properties. Table III is presenting some of the properties that were verified, expressed using the Linear Temporal Logic (LTL). Some of the operators used are:

- *Globally* `(G)`: `G p` meaning that the property `p` holds in any state
- *NeXt* (`X`): `X p` meaning that `p` holds in the next state
- *Implies* (`->`), and (`&`), or (`|`), negation (`!`) having the expected meaning, e.g. `p -> q (p` implies `q)`

For example, property P1 from the table says that globally the following holds: if current state is PC or Brake, then the next state will never be CC. P2 states that globally, if the current state is CC and the brake event is received, then the next state will not be Brake (the system should not transit to Brake directly, it will first return to PO or PA).

TABLE II. NuSMV CODE EXCERPT FOR THE EBIKE MODEL

Line	Code excerpts
12	`next(event):= {pa, pc, cc, br, ccc, pac, pcc};`
13	`next(State):= case`
14	`(State = PO & event = pa) : PA;`
15	`(State = PO & event = cc) : CC;`
16	`(State = PO & event = pc) : PC;`
17	`(State = PO & event = br) : Brake;`
18	`(State = PA & event = br) : Brake;`
19	`(State = PA & event = pac): PO;`
20	`(State = PA & event = cc) : CC;`
21	`(State = CC & event = ccc & StateBeforeCC = PO):PO;`
22	`(State = CC & event = ccc & StateBeforeCC = PA):PA;`
23	`(State = CC & event = br & StateBeforeCC = PO): PO;`
24	`(State = CC & event = br & StateBeforeCC = PA): PA;`
25	`(State = PC & event = pcc) : PO;`
26	`(State = PC & event = br) : Brake;`
27	`TRUE : State;`
28	`esac;`
29	`next(StateBeforeCC) :=  case`
30	`(State = PO & next(State) = CC)  : PO;`
31	`(State = PA & next(State) = CC)  : PA;`
32	`TRUE : StateBeforeCC;`
33	`esac;`
34	`next(EngRun) := case`
35	`(next(State) = PO) : FALSE;`
36	`(next(State) = Brake) : FALSE;`
37	`TRUE : TRUE;`
28	`esac;`

The properties listed in Table III either match the behaviour corresponding to key transitions illustrated in Fig. 4 and discussed in Section III or concern some generic behaviour, describing it as an invariant of the system. Indeed, the first transition corresponds to P1, second transition is described by P3 and the third one by P2. Properties P4 and P5 describe the behaviour of the system by identifying its states corresponding to the engine (usage of the battery) being on or off.

TABLE III. NuSMV CODE EXCERPT FOR THE EBIKE MODEL

Prop. ID	Properties	
	Informal query **`Formal query LTL`**	***Result***
P1	*The user should be able to request / activate Cruise Control only from PO or PA* `LTLSPEC G ( (State=PC | State= Brake) ->  !(X(State=CC)) )`	True
P2	*The system should not transit directly from CC to Brake directly* `LTLSPEC G ( (State=CC & event=br) -> !(X (State=Brake)))`	True
P3	*When brake is requested in CC the system returns to PA or PO* `LTLSPEC G (( State=CC & event= br)->(X(State=PO | State=PA)))`	True
P4	*When system is in CC, PA or PC state, the Engine is running (EngRun is True)* `LTLSPEC ((State=CC | State=PA | State=PC) -> (EngRun=TRUE))`	True
P5	*When system is in PO or Brake state, the EngRun is False* `LTLSPEC G ( (State=Brake | State= PO) -> (EngRun=FALSE))`	True

Other properties can be considered and associated systematically to the requirements listed in Section III and described by various notations. This will allow us to validate the system requirements with a formal approach. The formal verification is further complemented by a testing approach in the following Section. The NuSMV specification of the model and the properties verified are available from [38].

V. TESTING

Testing is largely used in software industry for validating the software products, but it is also used in system engineering, as shown in the introductory section presenting the V model, and in verifying cyber physical systems [2, 42].

The testing method used in this approach, called X-machine based testing, shows a tight coupling with the formal model previously defined [18, 19]. In this section, we will present the constraints imposed on the model used and the way a test set is generated for a specific testing criterion.

The testing approach presented in [19] (given that certain constraints are imposed to the model, called design for testing conditions [18]) creates a one-to-one relationship between the input values and the functions defined in the model and provides test set for various testing criteria – state cover, transition cover, etc. These constraints refer to the type of associated automaton (this should be minimal, hence deterministic) and the functions that appear in the definition of the X-machine [19]. Each function, *f*, must be test complete, which means that for every memory value, *m*, there must be an input, *in*, such that (*in*, *m*) is in the domain of *f*. Also the X-machine must be output distinguishable, i.e., any two distinct functions must return distinct outputs when the same input and memory values are accepted [19].

In our case we are dealing, as it comes out of the model description [38], with an X-machine where functions are partially defined, but their domains are disjoint. This means that through the tuples (*inp*, *mem*), where *inp* is an input and *mem* a memory value we uniquely identify functions. For this reason, we do not need outputs. When the same input might be accepted by multiple functions, such as "*br*" by both CCrequest and Brakerequest then distinct memory values are matched up with these functions as it follows from [38]. According to these observations a function wrongly triggered in certain circumstances, for instance wrongly associated with a state of the model, could be discovered with an appropriate test case.

This X-machine based testing approach is supported by tools verifying the design for testing conditions and generating test sets. Some of them [21, 23] use a high level specification of the X-machine in the form of an XML file, describing the functions, memory, input domains for the functions etc. For the e-Bike case study we have specified the X-machine model from Section III using the Broker@Cloud Verification, Validation and Testing Tool Suite [23]. This tool suite enhances the specification writing with:

- A validation module, which checks the associated automaton, signalling the existence of non-reachable states, events (or inputs) ignored in certain states (this could be a design choice, but the tool detects them and gives a warning).
- A verification module, which checks the functions, e.g. would produce a warning message when several

functions processing the same input combination (non-determinism) or if there is no function processing a certain input (the system will block or halt).

- A test generation module able to produce abstract test suites, i.e., sequences of functions with their input values, realising different levels of coverage of the state machine (e.g. state cover, transition cover, or even more, by providing the maximum path length to be explored from each state)
- A test grounding module which maps the abstract test suites into concrete tests for a programming language (or technology), e.g., producing JUnit test cases for a Java implementation (or more specific test classes for JAX-WS, RESTful web services).

We have specified, verified and validated our e-Bike specification using the Broker@Cloud tool suite and also run the test generator, obtaining complete test suites, according to the X-machine testing methodology [18, 19]. We consider that it is mandatory to achieve at least state and transition cover. Furthermore, as the whole test suits are generated automatically, we set the length of the path to be explored after reaching each state to 2. It is worth mentioning that according to this test methodology the generated test set contains valid and invalid inputs, which are not accepted by the X-machine. This aims to check the conformance of the implementation with respect to the specification, because the implementation might be accepting inputs which are ignored by specification.

For example, the sequence of 3 function calls: PArequest("pa"); CCrequest("cc"); PArequest("pa"); contains 2 valid transitions in the X-machine (PO to PA, then PA to CC), but the last one should be ignored, as the PArequest is not a valid transition from CC. If the implementation contains an error, e.g., accepting a PArequest on any circumstances, then this test case will spot it.

Another example of problematic transitions is represented by the case when some functions could accept the same input, but depending on the current memory or state of the machine, only one of them could be triggered. For example, the sequence of calls CCrequest("cc"); CCcancelled("br"); is processing the inputs “cc” with CCrequest (system will move from PO to CC) and “br” with CCcancelled function (because the current state is CC). However, if an incorrect implementation is processing “br” with the Brake function, the system would jump from CC to Br, instead of triggering the CCcancelledtoPO function which would transit from CC to PO. Checking after applying this sequence the current state (Br or PO) can show if the implementation conforms to the specification. More details and the complete generated test suites as XML file and JUnit class can be found in [38].

VI. Conclusions

In this paper, we have presented our initial approach towards an integrated methodology to verify and test the desired behaviours of engineered systems. This approach helps system designers alleviate issues and shortcomings emerging from the lack of harmonization of verification and testing related solutions against each level of system design and decomposition analysis.

We have illustrated our approach in a case study, the cruise control system of an e-Bike system, through the use of verification and model-based testing techniques. To this end, we defined a formal X-machine model capturing the dynamics of the cruise control system, and mapped it to a state based a representation. Using the NuSMV model checking tool, we formally verified the system requirements (that were expressed in a formal logic) against the formal model.

The validation of the implementation is obtained by using a model-based testing approach. To achieve that, we have specified, verified and validated our e-Bike specification using the Broker@Cloud tool suite and also run the test generator, obtaining complete test suites, according to the X-machine testing methodology.

In this paper, we mainly focused on the feature interactions of the cruise control system (i.e. CC, PA, PC, Brake, PO) at only system level within a V-model (as aimed in this paper, see Fig. 1). It is envisioned that same approach can be deployed across other system decomposition levels such as subsystem and component levels within V-model where each subsystem (e.g. Electric Control Unit) and component (e.g. micro-controller) on their own possess states and behaviours with other interacting subsystems and components. After this, it would then be explored how the integrated approach can be deployed and enhanced to support V-model philosophy thereby making sure of traceability from top-down requirements definition and decomposition (from system to component level) to bottom-up integration and verification among levels (from component to system level).

We will further study the dynamic behaviour of the system analysed in this paper such that we can identify the conditions that cause the system to be unable to meet its requirements. In this respect, we will use the *noise factor*, which has long been used in engineering, to assess *modelling* uncertainty, and study the effect of noise factors through *(stochastic/probabilistic) verification* and *testing*.

This approach could be further complemented by an agent-based simulation of the whole system to identify conditions where the system fails to achieve its function – and thus derive system function failure modes. For simulation purposes, especially for more complex models, e.g. having communication and also parallel evolution, specialized software like FLAME [12], which is an agent-based simulation framework based on X-machine formalism, is an ideal simulation environment.

This envisaged holistic approach, integrating these different aspects, can be carried out at a very early stage to support identification and validation of system function failure modes.

References

[1] A. Albers and C. Zingel, “Challenges of model-based systems engineering: a study towards unified term understanding and the state of usage of SysML,” Smart Product Engineering, Proc. of the 23rd CIRP Design Conference, LNPE, pp. 83-92, 2013.

[2] S.A. Asadollah, R. Inam, and H. Hansson, "A survey on testing cyber physical systems," in IFIP International Conference on Testing Software and Systems, LNCS, vol. 9447, pp. 194-207, 2015.

[3] D.M. Buede, The Engineering Design of Systems: Models and Methods, 2nd ed. New Jersey: Wiley, 2009.

[4] F. Campean, H. Williams, A.Uddin, U. Yildirim, "Engineering systems analysis through failure mode avoidance", Lecture Notes ENG4111M, University of Bradford, 2017.

[5] F. Campean, U. Yildirim, "Enhanced sequence diagram for function modelling of complex systems", in Procedia CIRP, 27th CIRP Design Conference, 2017.

[6] A. Cimatti, E. Clarke, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella, "NuSMV 2: an OpenSource tool for symbolic model checking," in Proc. of International Conference on Computer-Aided Verification (CAV 2002), LNCS, vol. 2404, pp 359-364, 2002.

[7] E. M. Clarke and E.A. Emerson, "Design and synthesis of synchronization skeletons using branching-time temporal logic," in Logic of Programs, Workshop, pp. 52–71, 1982.

[8] E. Clarke, O. Grumberg, and D. Peled, Model Checking. Cambridge, MA: MIT Press, 1999.

[9] M. Dhineshkumar and Galeebathullah, "An approach to generate test cases from sequence diagram," in Proc. of International Conference on Intelligent Computing Applications, pp. 345-349, 2014.

[10] M. Eigner, T. Dickopf, C. Huwig, "An interdisciplinary model-based design approach for developing cybertronic systems," in Proc. of International Design Conference, pp. 1647-1656, 2016.

[11] S. D. Eppinger and T.R. Browning, Design Structure Matrix Methods and Applications. Cambridge, MA: The MIT Press, 2012.

[12] FLAME web site. [Online]. Available: http://flame.ac.uk/.

[13] S. Friedenthal, A. Moore, and R. Steiner, A Practical Guide to SysML: The Systems Modeling Language, 2nd edition. Morgan Kaufmann, 2014.

[14] D. Harel. "Statecharts: a visual formalism for complex systems," Science of Computer Programming, vol. 8, pp. 231-274, 1987.

[15] P. Hehenberger, B. Vogel-Heuser, D. Bradley, B. Eynard, T. Tomiyama, and S. Achiche, "Design, modelling, simulation and integration of cyber physical systems: Methods and applications," Computers in Industry, vol. 82, pp.273–289, 2016.

[16] E. Henshall and F. Campean, "Design verification as a key deliverable of function failure avoidance," SAE Int. J. Mater. Manuf., vol 3(1), pp. 445-453, 2010.

[17] R.M. Hierons, K. Bogdanov, J.P. Bowen, R. Cleaveland, J. Derrick, J. Dick, M. Gheorghe, M. Harman, K. Kapoor, P. Krause, G. Lüttgen, A.J.H. Simons, S. Vilkomir, M. R. Woodward, and H. Zedan, "Using formal specifications to support testing," ACM Computing Surveys, vol. 41 (2), Article No. 9, 2009.

[18] M. Holcombe and F. Ipate, Correct Systems: Building a Business Process Solution, Berlin: Springer Verlag, 1998.

[19] F. Ipate and M. Holcombe, "An integration testing method that is proved to find all faults," International Journal of Computer Mathematics, vol. 63, pp. 159-178, 1997.

[20] F. Ipate and M. Holcome, "Generating test sets from non-deteministic stream X-machines," Formal Aspects of Computing, vol. 12, pp. 443-458, 2000.

[21] F. Ipate and D. Dranidis, "A unified integration and component testing approach from deterministic stream X-machine specifications," Formal Aspects of Computing, vol. 28, pp. 1-20, 2016.

[22] E.-Y. Kang, E.P. Enoiu, R. Marinescu, C. Seceleanu, P-Y. Schobbens, and P. Pettesson, "A methodology for formal analysis and verification of EAST-ADL models", Reliability Engineering and System Safety, vol. 120, pp. 127-138, 2013.

[23] R. Lefticaru, A.J.H. Simons, "X-machine based testing for cloud services," in Proc. of ESOCC 2014 Workshops, CCIS, vol. 508, pp. 175-189, 2014.

[24] C. Liu, H.P. Hildre, H. Zhang, and T. Rølvåg, "Conceptual design of multi-modal products," Research in Engineering Design, vol. 26(3), pp. 219-234, 2015.

[25] R. Marinescu, "Model-checking and model-based testing of automotive embedded systems", Ph.D. Thesis, Mälardalen University, 2014.

[26] J. Mitschang, "Evaluation of a model-based development process for automotive embedded systems," Diploma Thesis, Technische Universität Kaiserslautern, 2009.

[27] A. Muetze and Y.C. Tan, "Electric bikes - a performance evaluation," IEEE Industry Applications Magazine, vol. 13, pp. 12-21, 2007.

[28] D. Owen, "Combining complementary formal verification strategies to improve performance and accuracy," Ph.D. Thesis, West Virginia University, 2007.

[29] T. Peng and G. Ding, "Formal specification and automated verification of UML2.0 sequence diagrams," in Proc. of IEEE International Conference on Granular Computing, pp. 370-375, 2012.

[30] I. Perseil and L. Pautet, "Formal methods integration in software engineering", Innovations in Systems and Software Engineering", Vol. 6 (1), pp. 5 – 11, 2010.

[31] A. Pnueli, "The temporal logic of programs" in Proc. of the 18th Annual IEEE Symposium on Foundations of Computer Science, pp. 46–57, 1977.

[32] T. Tomiyama, "Architecture-centric model-based product development," in Mechatronics. Linz, Austria, 2012. in Proc. of Mechatronics, pp. 434–443, 2012.

[33] A. Uddin, F. Campean, and M.K. Khan, "Application of the interface analysis template for deriving system requirements," in Proc. of International Design Conference, pp. 543-552, 2016.

[34] A. Uddin, M.K. Khan, F. Campean, and M. Masood, "A framework for complex product architecture analysis using an integrated approach," *Concurrent Engineering: Reasearch and Applications*, vol. 24 (3), pp. 195-210, 2016.

[35] A.V. Varadarajan, M. Romijn, B. Ossthoek, J. van de Mortel-Fronczak, and J. Beijer, "Development and validation of functional model of a cruise control system," in Proc. of Formal Engineering Approaches to Software Components and Architecture (FESCA), pp. 45-58, 2016.

[36] S. Vasilache, "DePAT: a framework for specification-based automatic test generation," in Proc. of the International MultiConference of Engineers and Computer Scientists (IMECS), vol 1, pp. 517-522, 2015.

[37] S. Vasilache, "Specification-based test case generation using dependency diagrams," in Proc. of the World Congress on Engineering and Computer Science, vol 1, pp. 185-189, 2016.

[38] Web page with the models presented in this paper. [Online] Available: http://www.inf.brad.ac.uk/~skonur/models/ebike/.

[39] J. Woodcock, P. G. Larsen, J. Bicarregui, and J. Fitzgerald, "Formal methods: practice and experience," ACM Computing Surveys, vol 41 (4), Article No. 19, 2009.

[40] U. Yildirim, F. Campean, and H. Williams, "Function modelling using the system state flow diagram," AIEDAM Special Issue: Function Modelling: Benchmark Models, Problems, and Approaches. vol.31 (3), in press, 2017.

[41] C. Zheng, P. Hehenberger, J. Le Duigou, M. Bricogne, B. Eynard, "Multidisciplinary design methodology for mechatronic systems based on interface model", Research in Engineering Design, DOI 10.1007/s00163-016-0243-2, 2016.

[42] X. Zheng, C. Julien, M. Kim, and S. Khurshid, "Perceptions on the state of the art in verification and validation in cyber physical systems", IEEE Systems Journal, DOI: 10.1109/JSYST.2015.2496293, 2015.

[43] C. Zingel, A. Albers, M. Matthiesen, and M. Maletz, "Experiences and advancements from one year of explorative application of an integrated model-based development technique using C&C^2-A in SysML," IAENG International Journal of Computer Science, vol. 39 (2), pp. 165-181, 2012.