Proc. of the Second Intl. Conf. on Advances In Computing, Communication and Information Technology- CCIT 2014.

ISBN: 978-1-63248-051-4 doi: 10.15224/ 978-1-63248-051-4-133

Modelling of Cyber Physical Systems using AADL and Event-B

Manoranjan Satpathy

Abstract— Given the architecture of a Cyber Physical System (CPS) in AADL (Architecture Analysis and Design Language), we aim to use a model based development approach for system construction. We use the Event-B formalism for modelling and analysis. The model decomposition mechanism in Event-B helps us in establishing a one to one correspondence between the AADL components and the Event-B sub-models. Using the example of an Adaptive Cruise Controller, we show how interesting architectural properties could be verified within the Event-B modelling framework.

***Keywords*—Model based development, Cyber physical systems, Architectural Properties**

I. Introduction

Cyber physical systems integrate computation, networking and physical processes. Since there could be a signal propagation loop between the physical plant and the computational elements, it means that the plant affects the computation, and computation in turn affects the plant [6]. Some of the prominent examples of CPS are (a) Adaptive Cruise Controller for a vehicle, (b) Aircraft fuel tank controller, (c) Lane centring controller for a vehicle, and (d) Remote healthcare monitoring system. Modelling of a CPS has many challenges [6] in that (a) a CPS model needs to integrate different models of computation (MOC) -- plants need continuous-time models whereas the rest of the system is discrete models, (b) modelling needs to consider the dynamics of communication and software, and (c) time is critical to operation.

AADL is a standard language for specifying the architecture of embedded control systems [5]. AADL is currently being extensively used to specify architectures of embedded software in avionics, automotive, transportation and robotics domains. An important concept of AADL is refinement, which is the focus of this paper. An AADL model can be associated with a set of architectural properties and it is expected that these properties are preserved under refinement. Thus, an abstract model can be gradually refined to introduce sub-components, ports, threads, devices, buses etc. However, the model specification and the refinement mechanism in AADL are of informal nature; therefore, there is a need to assign formal semantics to AADL entities that captures architectural properties.

Event-B [1] is an independent formal modelling notation for rigorous development of software systems. The two primary characteristics of Event-B are the notion of formal refinement and model decomposition [10]. Consistency of and refinement relationship between Event-B models can be formally verified. As the model grows bigger and bigger, the proof effort of showing the correctness of the model with respect to its specification grows with the size off the model. In order to keep the proof effort within bounds, Event-B has the decomposition mechanism which means a model can be decomposed to component models and the component models can be further refined independently.

In this paper, we use AADL for specifying the architecture of a CPS and its architectural properties. We then use the Event-B method for formally modelling the requirements while respecting the architecture and the architectural properties as specified in the AADL. Model decomposition technique is used to decompose an Event-B model so that the sub-models represent the AADL components. Architectural properties are modelled as invariants which are verified within the Event-B method using necessary tool support [2]. We illustrate our approach by considering the case study of a simplified Adaptive Cruise Controller.

The organization of the paper is as follows. Section II discusses our case study. Section III presents the Event-B method and the model decomposition mechanism in greater detail. In Section IV, we show how a one to one correspondence can be established between AADL components and the Event-B components. Section V presents the related work, and Section VI concludes.

II. Our Case Study

We will illustrate our method in relation to the example of an Adaptive Cruise Controller (ACC). An ACC keeps the host vehicle at a certain gap with respect to the front vehicle. When there is no front vehicle, the ACC system behaves like a Cruise Controller meaning that it maintains the speed of the host vehicle at a certain speed – called the cruise speed. When a front vehicle appears, the ACC system controls the speed such that a minimum gap – or more – is maintained between the two vehicles.

The mode behaviour of the ACC system is as follows. Initially the ACC is in an off mode. When the driver presses the on button, the system moves to the ready mode. The ACC operates beyond a minimum speed. When the ACC is at ready mode and the minimum speed condition is satisfied then if the driver presses a set button then the ACC transits to CC mode.

Author’s Name: Manoranjan Satpathy
Indian Institute of Technology (IIT), Bhubaneswar
Bhubaneswar – 751013; India

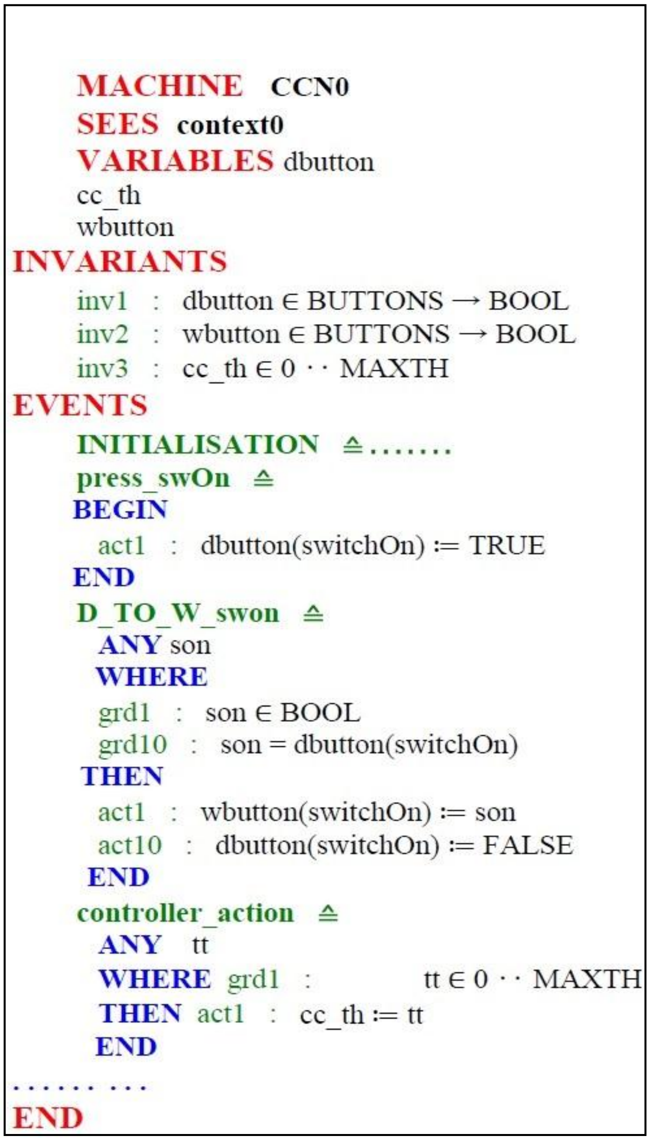
When in CC mode and there is no front vehicle, then ACC drives the vehicle at the cruise speed. As soon as a front vehicle appears close to the host vehicle, then state control transits to the ACC mode in which a minimum gap is maintained between the two vehicles. At any time the driver can take control of the vehicle by pressing either the brake or the throttle pedal. The driver can give control back to the ACC by pressing a resume button. Note that whenever the system is either in CC or ACC mode, then the throttle value is determined by the system; otherwise the throttle value is determined by the driver.

III. The Event-B Method

Event-B is a formal modelling notation based on first order logic and Set Theory [1]. The basic unit of modelling in Event-B is an Event-B machine. Figure 1 shows a truncated machine -- called CCN0 -- of an abstract Adaptive Cruise Controller (ACC). All constant and set declarations are stored in a context machine called **context0** in the figure -- details of this machine has not been shown. Machine CCN0 has a set of state variables declared under the **VARIABLES** clause. These variables must satisfy some constraints which are declared under the heading **INVARIANTS**. Such invariants need to be satisfied at all points when the machine is executed. In the figure we have shown only some of the variables and invariants. The events under the heading **EVENTS** define the dynamic behaviour of the machine. An event has a guard and an action part. When the guard is true the event is enabled. At any time one of the enabled events is non-deterministically selected for execution. Then the individual actions of the selected event are executed in parallel. When there are no more enabled events, the machine halts. A special event called **INITIALISATION** initialises the machine variables; this is executed exactly once at the beginning (we have omitted the initialisations). Details about Event-B machines and its execution can be found in [1].

Figure 1 shows only some of the events. The set BUTTONS and the value of the constant MAXTH -- for maximum throttle -- are defined in **context0**. In the event *controller_action*, a non-deterministic throttle value is selected and it is given to the variable *cc_th*, for the throttle value computed by the Controller. In event *D_TO_W_swOn*, the value of the function dbutton(swOn) is copied to dbutton(swOn); this models sending of the event from the Driver component to the bus. In event *pr_swOn* -- this models the driver pressing the ON button -- dbutton(swOn) is given the **true** value.

The *Rodin Platform* [2] provides the tool support for consistency and refinement checking of Event-B models. This toolset auto-generates a set of proof obligations which can be discharged automatically or by interactive theorem proving. Rodin toolset has necessary provers for theorem proving. Once the proof obligations are discharged, it would mean that the Event-B models are correct and refinement relationship holds between respective models.



```
        MACHINE  CCN0
        SEES  context0
        VARIABLES dbutton
        cc_th
        wbutton
INVARIANTS
        inv1  :  dbutton ∈ BUTTONS → BOOL
        inv2  :  wbutton ∈ BUTTONS → BOOL
        inv3  :  cc_th ∈ 0 ‥ MAXTH
EVENTS
        INITIALISATION  ≙.......
        press_swOn  ≙
        BEGIN
          act1  :  dbutton(switchOn) := TRUE
        END
        D_TO_W_swon  ≙
          ANY son
          WHERE
          grd1  :  son ∈ BOOL
          grd10  :  son = dbutton(switchOn)
         THEN
          act1  :  wbutton(switchOn) := son
          act10  :  dbutton(switchOn) := FALSE
         END
        controller_action  ≙
          ANY   tt
          WHERE grd1  :          tt ∈ 0 ‥ MAXTH
          THEN act1  :  cc_th := tt
          END
...... ...
END
```

Figure 1. Structure of an Event-B model

Model Decomposition

We consider the shared event decomposition of an Event-B model [10]. Let us ignore the controller_action event and the variable cc_th in the machine of Figure 1. What the truncated model represents is that --after due initialization -- the driver presses the switch button to make it on. As a consequence of this event, the functional value dbutton(swOn) gets a true value. The event D_TO_W_swOn then copies the value of dbutton(swOn) to wbutton(swOn) -- this means this event puts the value of the driver press into the bus. We will now decompose the current machine into a Driver machine and a Bus machine. We distribute the variables of the original machine into the component machines: variable dbutton belongs to the Driver machine and the variable wbutton belongs to Bus.

Figure [2] shows the component machines. Observe that events pr_swOn and D_TO_W_swOn are in the Driver machine; event D_TO_W_swOn is also in the Bus machine. What it means is that the two events D_TO_W_swOn in both the machines are synchronized which results in the copying of the value of dbutton(swOn) to wbutton(swOn). Also the dbutton(swOn) next becomes false meaning that the corresponding button is ready to be pressed again.

Rodin toolset has a decomposition plugin which can perform model decomposition into component models. That the composition of the component models are equivalent to the original model is based on the sound theory of model decomposition in Event-B [10].

Overview of AADL

AADL provides textual and graphical formats to specify architectures and is supported by well-structured modelling environments and robust underlying syntax [5]. AADL follows a *component-style* architecture framework, components are central to an AADL architecture. Component templates are available to model run-time entities of application software (like system, process, thread, data, sub-program etc.) and for hardware and platform specific entities (like processor, bus, memory, device etc.). Such a categorization of components make AADL suitable for specifying embedded software architectures. In addition, AADL has several features for modular definition of architectures: pre-defined refinement templates, packages, abstract components, arrays of sub-components etc.

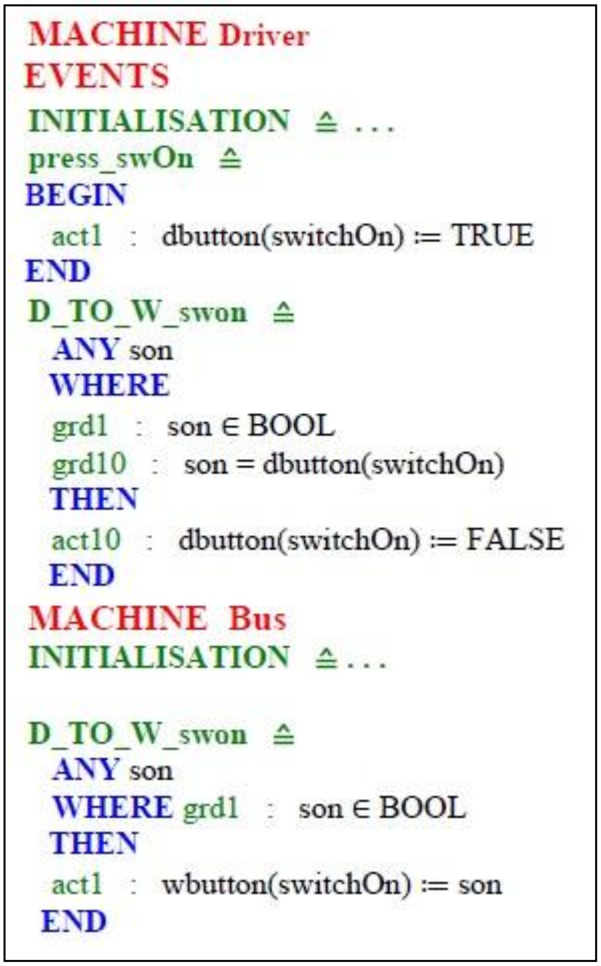
Components in AADL interact with each other through *features* which are interfaces to exchange data or events. Components can also communicate with other components using synchronous call and return and through shared data access. Each component (feature) has a set of properties associated with it that describe the component (feature). In addition, an AADL model can have global properties like end-to-end latency.

Figure 3 shows the various components of ACC: Driver, Controller, Plant, Gap Sensor, Plant and the Bus. Each component has threads, and the threads have their periodicities. AADL can specify all these architectural elements and their properties like the thread periodicities and the end-to-end latency.

IV. AADL to Event-B

AADL and Event-B both support refinement mechanism. Syntactic constructs are provided to transform an AADL model to a more detailed one. However, the refinement mechanism in AADL are informal in nature.

Event-B supports formal refinement. It can also specify and prove timing properties. Refer to Figure 3. A thread – call it th1 -- puts the values of the switch presses into the bus. A thread (th3) in the Gap Sensor component also puts the value of the gap in the bus. Thread th2 puts the values in the bus into the controller component which is consumed by another thread in the Controller (th4) which in turn computes the throttle value. Another thread (th5) puts the throttle value into the bus. Thread th6 puts the throttle value in the bus into the Plant component. The plant thread th7 uses this throttle value to produce the engine speed. Thread th8 senses the speed value. Th9 puts the speed value into the bus. Th10 puts this speed value into the controller which is consumed by th4.



```
MACHINE Driver
EVENTS
INITIALISATION ≙ . . .
press_swOn ≙
BEGIN
  act1  :  dbutton(switchOn) := TRUE
END
D_TO_W_swon ≙
  ANY son
  WHERE
  grd1  :  son ∈ BOOL
  grd10  :  son = dbutton(switchOn)
  THEN
  act10  :  dbutton(switchOn) := FALSE
  END
MACHINE  Bus
INITIALISATION ≙ . . .

D_TO_W_swon ≙
  ANY son
  WHERE grd1  :  son ∈ BOOL
  THEN
  act1  :  wbutton(switchOn) := son
  END
```

Figure 2. Event-B decomposition

We claim that for an AADL model at any level of abstraction, there can be a corresponding Event-B model, and the Event-B model captures all the architectural properties. It is easy to put the above AADL model in Event-B. Each thread behaviour can be captured as an event in event-B.

Figure 1 shows a portion of the Event-B model. Refer to the events *press_Swon* and *D_TO_W_swOn*. When driver presses the switch-on button, then dbutton(swOn) becomes true. When the second event executes the value of dbutton(swOn) is copied to wbutton(swOn), the latter is a variable of the bus. Once the value is copied the value of dbutton(swOn) is reset to mean that the variable is ready for another switch press.

In the next refinement, task periodicities are assigned to each process thread in the AADL model, and some timing properties. To model this in the Event-B, we introduce a timer.

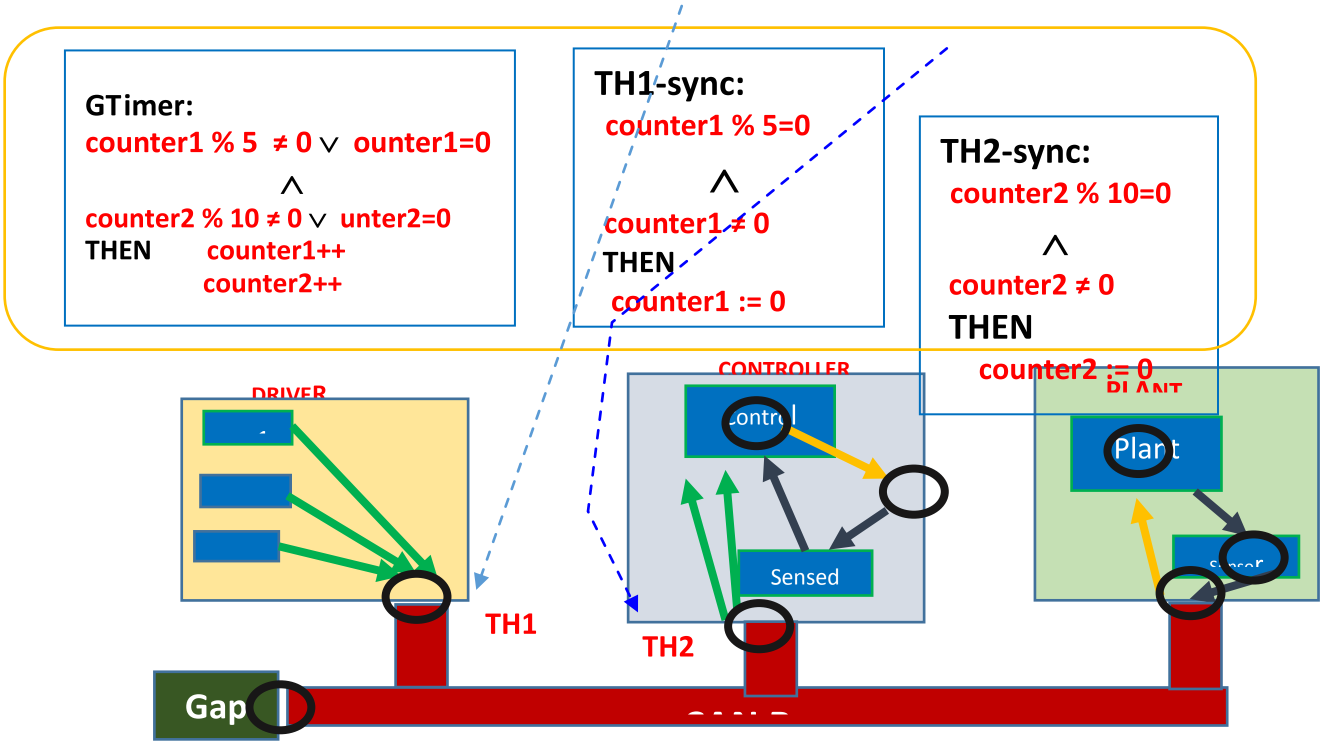


Figure 3. Event-B components and timing synchronizationn

In Figure 3, let TH1 represent a thread which puts some value of the driver component into the bus; and it be of periodicity 5ms. Let TH2 be another thread which sends a value in the bus to the Controller, and it be of periodicity 10 ms. Let counter1 and counter2 be two counters which represents the passing of time in both the threads. Event GTimer updates both the counters at the same time. Only when the counter values match any of the thread periodicities, then that thread executes -- not shown in figure -- and the corresponding counter values are re-initialized to zero. It can be seen that the periodicities of the threads are met.. Similar approach is used to satisfy the periodicities of all threads.

The refined AADL model has 5 components. However, we introduced a Timer module into the Event-B model CCN0_timer for modelling of timing properties; thus, the Event-B model has 6 modules. The AADL model has components, but the corresponding Event-B model is still monolithic though it has 6 modules within it; however, structurally, it is as given in Figure 3. Next we use shared event model decomposition [10] on the Event-B model to obtain 6 components. This decomposition has been performed by using the decomposition plugin [10] of the Rodin toolset.

By the theory of shared event decomposition, the original Event-B model is equivalent to the synchronous decomposition of the 6 decomposed Event-B models, as shown in Figure 4. The individual Event-B components can be independently refined. For instance, mode behaviour can be introduced into the Controller component. This refinement – because of the sound logic of model decomposition – will not violate the compositional property of the whole model.

The decomposed Event-B model, which has a close relationship with the corresponding AADL model, is correct by construction. We omit here the remaining refinements of the ACC model.At this stage, the decomposed Event-B model preserves the architectural semantics of the current AADL model; in addition, they have the same component structure. The only difference is that in the AADL model, the global notion of time is outside of the system; in Event-B, it is within the system.

In [8] and [9], we have developed a method and tool to automatically translate Event-B models into Simulink models [7]. Following this method, we can obtain Simulink models for the Driver, Controller and the Plant components. However the operations involving the Bus could be translated to the send() and receive() commands of a standard bus protocol.

V. The Related Work

Even though AADL is extensively used to model architecture of many embedded software [5], it lacks an exhaustive framework for formally verifying AADL models against its requirements. By using the Event-B method and model decomposition, we address this deficiency.

In [12], the authors take a synchronous scheduler specified in AADL and model it using Event-B. Their Event-B model is similar to the one discussed here in terms of a step by step translation and successive refinements. For the considered problem, the authors verify architectural properties involving schedulability. However, they consider only a synchronous subset of AADL where only thread components are modelled and translated to event-B. This paper does not discuss about model decomposition and the refinement of component models.

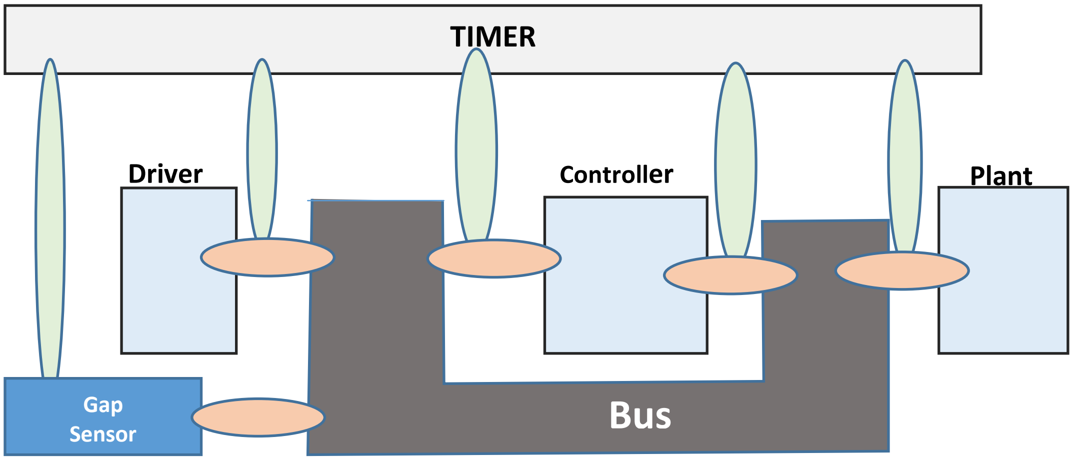


Figure 4. Communication synchronization between the Timer, the Bus and the individual components.

In [12], the authors take a synchronous scheduler specified in AADL and model it using Event-B. Their Event-B model is similar to the one discussed here in terms of a step by step translation and successive refinements. For the considered problem, the authors verify architectural properties involving schedulability. However, they consider only a synchronous subset of AADL where only thread components are modelled and translated to event-B. This paper does not discuss about model decomposition and the refinement of component models.

VI. Conclusion

The results presented in this paper are preliminary results towards use of AADL and AADL refinements – and their formalization – in the context of cyber physical systems. Our proposed framework uses a series of refinements and decompositions in Event-B to capture semantics of AADL models defined through successive refinements. Using the translation of AADL components here, we are able to formally prove architectural requirements of CPS. Our framework can also prove timing properties like end-to-end latency.

We can translate the Bus component into the commands of a bus protocol like the CAN bus. For the remaining components, we can derive Simulink models as discussed in [5].

References

[1] J.R-.Abrial, Modeling in Event-B: System and Software Engineering, Cambridge University Press, 2010.

[2] J.R-.Abrial, M. Butler, S. Hallerstede, T.S. Hoang, F. Mehta, L. Voisin, “Rodin: an open toolset for modelling and reasoning in Event-B,”, STTT, Vol. 12, pp 447-466, Novemner 2010.

[3] J.-P. Bodeveis, M. Filali, “Event B development of a synchronous AADL scheduler,” ENTCS Vol. 280, pp 23-33,2011.

[4] S.Corrigan, “Introduction to the Controller Area Network (CAN),” Application Report, Texas Instruments, 2008.

[5] P.H. Feiler, D.P. Gluch, Model-Based Engineering with AADL: An Introduction to the SAE Architecture Analysis & Design Language, Addison Wesley, 2012.

[6] E.A. Lee & S.A. Seshia, Introduction to Embedded Systems: a Cyber Physical Systems Approach, Lulu.com publishers, 2013.

[7] Mathworks, “Simulink – Simulation and model based design,” www.mathworks.com

[8] M. Satpathy, S. Ramesh, C. Snook, N.K. Singh, M. Butler, “A mixed approach to development of control designs,” 2013 IEEE CCSD-SU, Hyderabad, August 2013.

[9] M.Satpathy, C. Snook, S. Arora, S. Ramesh, M. Butler, “Systematic Development of Control Designs via Formal Refinement,” Intl. Conf. on Model Driven Engineering and Software Development, Barcelona, Feb 2013.

[10] R. Silva, C. Pascal, T.S. Hoang, M. Butler. Decomposition tool for Event-B, Software: Practice and Experience, Vol. 41(2), 2011.

[11] M. D’Souza, S. Ramesh, M. Satpathy, Architectural Semantics of AADL using Event-B, to be presented at the IEEE IC3I, November 2014.

[12] J.-P. Bodevis, “Event-B development of a synchronous AADL scheduler,” ENTCS, Volume 280, pp. 23-33, 2001.

About Author:

Dr M. Satpathy is an Associate Professor in Computer Science at IIT Bhubaneswae, India. His research interests are Software Design, Testing and Verification.

Event-B modelling and the model decomposition mechanism can address the informality that is inherent in an AADL model.