Model and Verify CPS using the Vienna Development Method

Shehroz Bashir Malik[1]

Contents

[1] shehroz-bashir.malik@stud.hshl.de

Abstract: In this paper, I will formally describe the Vienna Method. I will explain its theory and background. I will find and describe case studies. I will find tools that support it.

1 Motivation

The Vienna Development Method will be introduced and explored in this paper. Cyber-Physical Systems will be touched on briefly. The aim is to Validate and Verify a Cypber-Physical System using the Vienna Development Method. Certain use cases will be put forward to show the effects and efficiencies of VDM in tandem with Cyber-Physical Systems in industrial scenarios.

2 Foundation

3 What is a Cyber-Physical System?

In this section, I am going to define CPS, give examples of it, why do we need it, why is it the future [WIP]

A Cyber-Physical System is defined as the hybrid networked integration of Computation and Physical Processes which include human input. Physical Processes are controlled and monitored by Embedded Systems and their corresponding networks. A closed feedback loop is created that enables Computation Processes to adjust the Physical Processes as well as the other way around. This is a multi-disciplinary approach that encompasses many branches of Engineering and is inherently complex. [BG11] By definition, this system is heterogeneous, concurrent and deterministic **IS IT DETERMINISTIC? I THINK IT CAN BE NON AS WELL**. [DLV11] It can not be modeled in the eyes of Software Processes as they are procedural and performance dependent[ASS96]. In a Cyber-Physical System, the most important constraints to meet are Correctness and Timing. Processes must be executed simultaneously while ensuring certain time constraints are met.

3.1 Challenges of Modelling Cyber-Physical Systems

Models need to be created that provide well-defined clear semantics. They must be able to address certain challenges [Le08]. Patricia Derler et al. present the challenges as follows [DLV11]:

1. Modelling Various Behaviours - Models must be created that can handle a myriad of various behaviours - these must also include non-determinate and zeno behaviours. While modelling various Discrete Events executing simultaneously, it is possible for such behaviours to exist if the Modelling Language does not provide the appropriate semantics to model each individual event.
2. Consistency - As a system grows and becomes more complex, it is important to maintain consistency between components. As each component in a Cyber-Physical

System is in a feedback loop, any one change can cause a butterfly effect. It is important to ensure that errors are not carried forward.

3. Correctness - Correctness must be maintained throughout the system. Different components will accept different sorts of inputs. It is vital that these inputs are in the correct formats and changed as needed.
4. Functionality and Implementation - [WIP]
5. Distributed Behaviour and System Heterogeneity - Models should be capable of understanding the distributed nature of Cyber-Physical Systems while providing semantics that can handle disparities that exist in a Distributed System. The Model should be capable of providing a comprehensive overview of the entire system while respecting the domain-specific nature of certain components.

4 Why do we have to verify it?

self explanatory, perhaps I can combine these two sections and phrase it better[WIP]

5 Why do we have to validate it?

self explanatory[WIP]

6 Vienna Development Method

6.1 Introduction

The Vienna Development Method was developed in the IBM Laboratory Vienna in the 1970s to model and develop sequential software systems. [AP98] This specification language was standardized by the International Standards Organization (ISO) in 1996. [ISO96]
In order to specify, develop and verify computer systems - Formal Methods need to be employed. These involve the use of mathematical logic to perform rigorous analysis to ensure the reliability and robustness of a system.[La22].
VDM allows you to examine regions of incompleteness and vagueness in system specifications thus ensuring your final output is a valid implementation of your initial design. It is able to maintain flexibility is a landscape of ever-changing requirements. Multiple tools have been developed to employ VDM. The ones I will be exploring in this paper are Overture [Ov22] and VDM Tools [VD16].
An extension of the VDM Specification Language was developed to embrace object-orientation and concurrency. It is known as VDM++. It allows one to describe real-time distributed and embedded systems.[VLH06]

6.2 VDM-Specification Language [WIP]

1. Types - There are two types namely Primitive Types and Quote Types. Primitive Types are similar to declaring a variable. All sorts of arithmetic and logical operations can be carried out on these types. It includes high-level support for Mathematical Set Notation. These can include

$$\begin{aligned} \mathbb{N} &- Integers \\ \mathbb{Z} &- NaturalNumbers \\ \mathbb{R} &- RealNumbers \\ \mathbb{Q} &- RationalNumbers \\ \mathbb{B} &- Boolean \end{aligned} \tag{1}$$

2. Composite Types

6.3 VDM++ Notation

VDM++ employs a lot of concepts from Object Orientation. It can contain a group of class specifications. These can be categorized into two types, namely Active and Passive Classes. Active Classes can run on their own without an external stimulant whereas Passive Classes can be activated from an Active Class.
Each individual class has its own set of components:

1. Class Header - Class Name Declaration with support for single or multiple inheritance
2. Instance Variables - Variables along with their types are declared in this section. An invariant and the initial state can also be defined.
3. Functions - Can modify the state using implicit and explicit expressions but they can not refer to instance variables
4. Operations - Can modify the state using implicit pre/post condition expressions or statements. Considered to be more powerful than functions.
5. Threads - Threads enable concurrent behaviour. It is a stream of actions that need to be performed while the class is active. It can also be a unlimited stream of actions or in other words: an infinite loop.
6. Synchronization - Certain Boolean conditions need to be set in order to maintain mutual exclusion and enable concurrency.

VMD++ also includes basic concepts from C++ such as Constructors and Access Modifiers.

7 Case Studies of VDM

INCLUDE AN EXAMPLE OF CODE GENERATION TO C++/JAVA

7.1 Self-Balancing Scooter

In this paper, a Self-Balancing scooter is modeled. Their goal was to model and simulate from the inital stage of the project. A heterogeneous co-model was created that enabled co-simulation. This co-simulation was split in two components, namely Discrete-Event (DE) Models and Continuous-Time (CT) Models. The Vienna Development Method was used to express the DE Model. On the other hand, the 20-sim tool was used to model the CT side. These two models were executed simultaneously using the DESTECS co-simulation framework.[Fi13] For this paper, I will only be focusing on the Discrete Event Model. [FPG12]
The ChessWay scooter is balanced by an embedded controller connected to gyroscopes and accelerometers which prevent it from falling over. There are two controllers that control the left and right side respectively. They communicate over a data connection to access sensor data and synchronize their behaviour. They isolated the core components and variables of the scooter. Then they implemented them using VDM to model their Controller.

```
class Controller

instance variables
  -- sensors
  private angle: real;
  private velocity: real;
  -- actuators
  private acc_out: real;
  private vel_out: real;
  -- PID controllers
  private pid1: PID;
  private pid2: PID;

operations
  public Step : () ==> ()
  Step() == duration(20) (
    dcl err: real := velocity - angle;
    vel_out.Write(pid2.Out(err));
    acc_out.Write(pid1.Out(angle));
  );

  public GoSafe : () ==> ()
  GoSafe() == (
    vel_out.Write(0);
    acc_out.Write(0);
  );

thread
  periodic(1E6,0,0,0)(Step); -- 1kHz

end Controller
```

Fig. 1: Controller Modelled in VDM

Taken directly from the paper, this figure states the behavior of the Controller. It has instance variables that take values from sensors and actuators. These are then calculated using the PID objects to control necessary actions. The thread runs at a rate of 1000 Hz and initiates the Step operation.
Using this technique of co-simulation, they were able to easily explore different designs

and analyze any changes in their outputs. This enabled flexibility in the creative process. It guarantees robustness and reliability by allowing the user to see the effects of changes in the DE side or the CT side almost immediately.

7.2 Modelling an Injection Moulding Machine using the Vienna Development Method [WIP]

[Bö21]

8 Tools

Overture and VDM-Tools are explored due to their plethora of features. Features like Code Generation

8.1 VDMTools

In this paper John Fitzgerald et al. discuss the various features available in VDMTools [VD08]. VDMTools enables Syntax and Type checking. Type Checking also includes checking if the model follows correct composition of the VDM types. VDM++ classes and VDM-SL Modules can be analyzed using the State Semantic Analyzer. VDMTools checks whether the correct data types are modelled according to their respective inputs and outputs thus enabling integrity.
All VDM Models created can be executed. The built-in debugger allows breakpoints at functions and operations and investigating which stack is being called. One can also verify type invariants and the pre- and post- conditions on functions and operations.
An external logfile can be created by the interpreter to validate the model. This logfile includes every event that was run during execution as while as the time it is executed in. It introduces an additional section for Distributed System models where it also tracks the resource accessed by the event.
Scripts can be written to run tests of individual components in a module. These can be engineered to run sequentially or in large batches. This ensures correctness of a model.
VDM Models can be automatically generated into C++ and Java code. This is usually the last step in the development cycle. It is carried out after the system is verified and validated using the built-in tools.

8.2 Overture

9 Future Works

maybe I can add a section that discusses more about what can be done [WIP]

10 Conclusion

Bibliography

[AP98] Alagar, V. S.; Periyasamy, K.: Vienna Development Method. In: Specification of Software Systems, pp. 219–279. Springer New York, New York, NY, 1998.

[ASS96] Abelson, H.; Sussman, G.J.; Sussman, J.: Structure and Interpretation of Computer Programs. Electrical engineering and computer science series. MIT Press, 1996.

[BG11] Baheti, Radhakisan; Gill, Helen: Cyber-physical systems. The impact of control technology, 12(1):161–166, 2011.

[Bö21] Böttjer, Till; Sandberg, Michael; Larsen, Peter Gorm; Macedo, Hugo Daniel: Modelling an Injection Moulding Machine using the Vienna Development Method. Hugo Daniel Macedo, Casper Thule, and Ken Pierce (Editors), p. 51, 2021.

[DLV11] Derler, Patricia; Lee, Edward A; Vincentelli, Alberto Sangiovanni: Modeling cyber–physical systems. Proceedings of the IEEE, 100(1):13–28, 2011.

[Fi13] Fitzgerald, John S; Larsen, Peter Gorm; Pierce, Ken G; Verhoef, Marcel Henri Gerard: A formal approach to collaborative modelling and co-simulation for embedded systems. Mathematical Structures in Computer Science, 23(4):726–750, 2013.

[FPG12] Fitzgerald, John; Pierce, Ken; Gamble, Carl: A rigorous approach to the design of resilient cyber-physical systems through co-simulation. In: IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN 2012). pp. 1–6, 2012.

[ISO96] : Information technology — Programming languages, their environments and system software interfaces — Vienna Development Method — Specification Language — Part 1: Base language. Standard, International Organization for Standardization, Geneva, CH, December 1996.

[La22] Langley Formal Methods Program • What is Formal Methods, 06-Oct-22.

[Le08] Lee, Edward: Cyber Physical Systems: Design Challenges. pp. 363–369, 06 2008.

[Ov22] Overture Tool, open-source integrated development environment (IDE) for developing and analysing VDM models., 28-Aug-22.

[VD08] : VDMTools: Advances in Support for Formal Modeling in VDM. 43(2), 2008.

[VD16] VDMTools v9.0.6 by Kyushu Univeristy, 25-Oct-16.

[VLH06] Verhoef, Marcel; Larsen, Peter Gorm; Hooman, Jozef: Modeling and Validating Distributed Embedded Real-Time Systems with VDM++. In (Misra, Jayadev; Nipkow, Tobias; Sekerinski, Emil, eds): FM 2006: Formal Methods. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 147–162, 2006.