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 Cyber-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 Cyberphysical 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] https://www.youtube.com/watch?v=8HLJFlncMls good reference

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 in 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 [WIP]

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.

[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.

[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.

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

[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.