VDMWEB: A WEB-BASED VDM ANIMATION TOOL

PRESENTATION BASED ON *A Literate Programming and Animation Environment for VDM*

Harry Hughes and Leo Freitas

Newcastle University

Introduction

Aims/Objectives

To provide tools to accomplish two goals:

1. A literate programming environment (LPE), in the style of *Jupyter Notebook*[1] for VDM.
2. A framework to write GUI and animation system on top of VDM models.
3. An LPE enabling embedded animations/GUI linked with VDM specifications.

To assist development and exploration of VDM models.

[1] Jupyter Notebook available at https://jupyter.org/

Introduction

How this could help stakeholders and developers

Communication of technical information to non-technical audiences:

1. Raw program/specification code with comments in code make sense to programmers.
2. A console/command line based interface: avg. user rarely interacts with this kind of interface.

The proposal is to provide a solution as:

1. Literate programming allows the developer to put rich-text in-between code cells to explain functionality.
2. Animations and GUIs allow users to interact with the underlying model visually.

The design plan is:

- **A notebook extension** for Visual Studio Code (VSCode), providing the literate environment.
- A Java based web-server extension:
  - allowing the hosting of a REST API
  - model interaction (sending commands etc.)
  - static web content.

The REST API/static web server will enable the user to write simple web front-ends to be displayed for user interaction through a range of UI design options.

The decision to use a web server as a backend was in part driven by the design of VDMJ[2] and VS Code.

VSCode allows embedding of custom web output in a notebook via the Notebook API[3].

It became clear that providing web-server based hosting for VDM was a useful contribution with wide-ranging potential applications too.

[2] VDMJ is available at https://github.com/nickbattle/vdmj

[3] VSCode Notebook API documentation at https://code.visualstudio.com/api/extension-guides/notebook

Introduction

Original Design Diagram

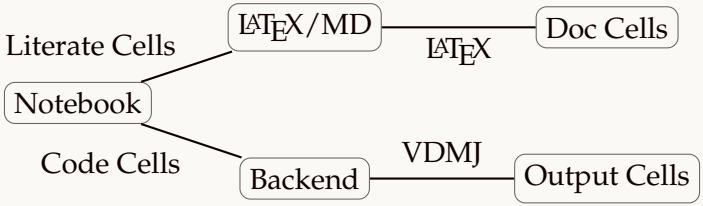


Figure 1. Equivalent of the `WEB` system from Knuth (1992) for this project

Figure 1 displays our initial design revised during later.

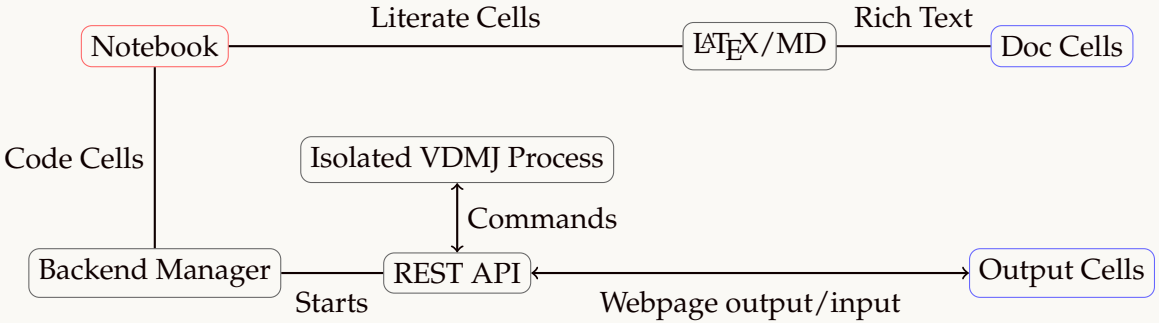


Figure 2. New version of Figure 1, reflecting design changes

Figure 2 reflects the final design, from input to output.

It does obfuscate some of the aspects of handling certain complications.

Jupyter Notebook Structure

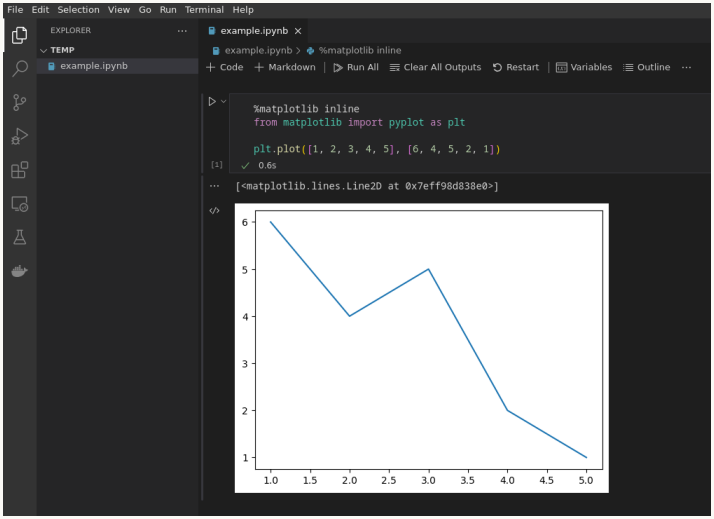


Figure 3. A Jupyter Notebook open in VSCode

VSCode running Python Jupyter Notebook window, with code output embedded after the code that produced it.

Jupyter Notebook Structure

Indicating output for Jupyter to display

The previous slide code is interpreted by Jupyter line-by-line in the Python interpreter.

```
%matplotlib inline
from matplotlib import pyplot as plt

plt.plot([1, 2, 3, 4, 5], [6, 4, 5, 2, 1])
✓ 0.6s
```

Figure 4. Code from Jupyter example

- The first line declares that `matplotlib` (a Python graphing library) should output it's plots through the Jupyter notebook interface.
- This system our inspiration for our VDM Notebook system. An equivalent system could declare how to output animations from a model and where.

We use VDM annotations as binding mechanism for REST API links. Listing 2 shows a concrete example.

The annotation points to a folder containing HTML/JS/CSS constituting a website that can send requests to `vdmj-remote` to query the model.

```
--@WebGUI("<nickname>", "<path to static web folder>")
module Conway
...
functions
    -- Perform one generation
    generation: Population -> Population
    generation(pop) ==
        (pop \ deadCells(pop)) union newCells(pop);
...
end Conway
```

Listing 2. Example of annotation webpage binding

The sample is an implementation of Conway's Game of Life, used as the test model.

We are working on moving VDMJ-remote calls as VDM annotations to simplify the HTML/JS/CSS setup.

Part I

Approach

VDM Notebook

Our working prototype (Figure 5) shows the idea works effectively.

It replicates the functionality of the *ConwayNB*[4] animation example through VDMJ's `RemoteControl` mechanism.

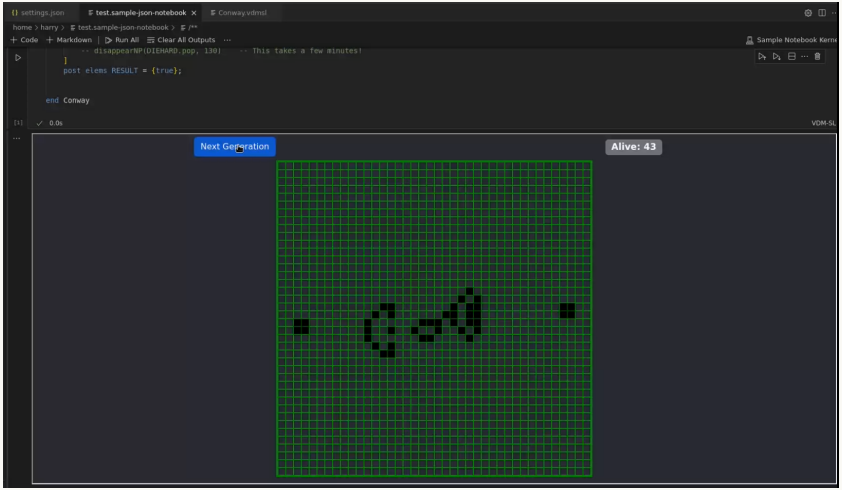


Figure 5. The original working prototype

[4]ConwayNB example available at https://github.com/leouk/VDM_Toolkit/tree/main/experiments/vdm/Games/ConwayNB

VDM Notebook

Issues with notebooks and VDM models

Jupyter Notebook is designed to develop Python, which is interpreted line-by-line.

During development, it became clear that using VDMJ for the interpreter meant that all code for a model had to be loaded at once.

This meant the primary purpose of splitting up code into blocks in a notebook was somewhat pointless.

We pivoted to web-based VDMJ. This brought advantages of VDM models being hosted as web servers and allowing animations to be built atop them.

VDMJ Remote

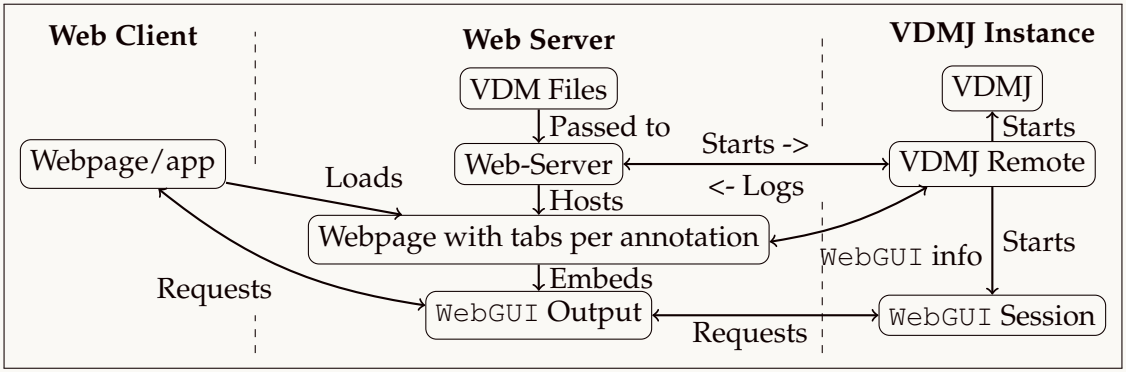
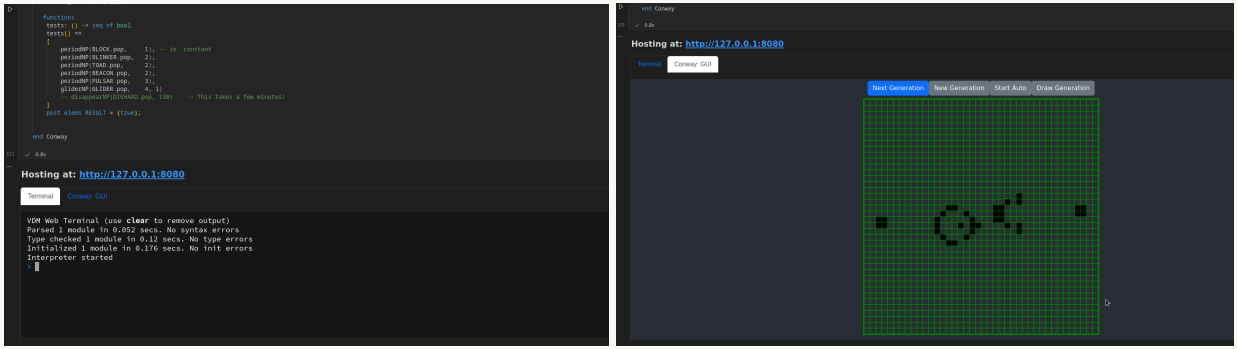


Figure 6. Broad systems diagram

Figure 6 shows a diagram of how VDMJ Remote works.

Each `WebGUI` annotation starts a separate web server to host it's specific output content.

WORKING VERSION DEMO



(a) Console **(b)** GUI

Figure 7. Prototype of VDM Notebook embedding VDMJ Remote in VS Code

Figure 7 shows screenshots of the working prototype, the output always has a terminal tab, but can also have as many annotation-specified `@WebGUI(...)` outputs in separate tabs.

Video Demo

Below is a link[5] to a video demo of a webpage running atop a model (Conway's GoL) running in VDMJ Remote. The webpage is developed in React and calls the REST API provided by VDMJ Remote.

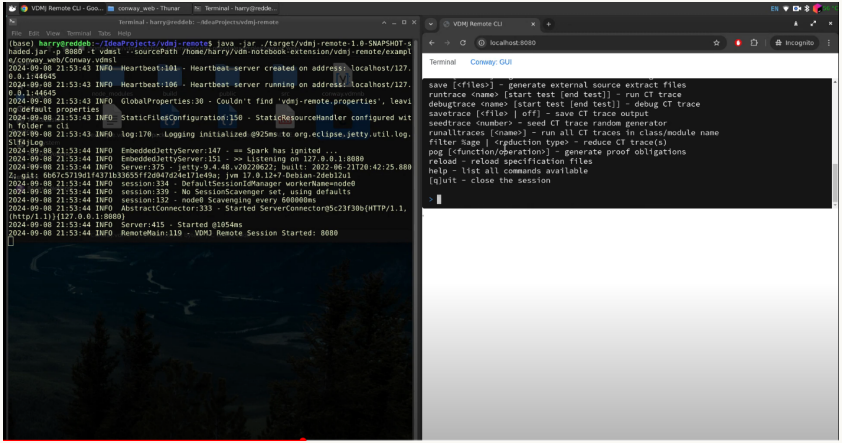


Figure 8. Demo video link (click on image)

[5] Video url: https://youtu.be/0OYVLfd1g0o

Outcome

The primary output of this project is the VDMJ Remote[6].

- Provides regulated (but not yet secured) access to a VDM model via HTTP requests.
- Can host static web content dynamically through the `@WebGUI` annotation in the model given at runtime.
- Includes a limited VDM console and tabs system on the root page.

[6] VDMJ Remote available at https://github.com/pointerless/vdmj-remote

Part II

Work yet to be done

Potential Extensions/Modifications

Improvements

Some options to improve VDMJ Remote are available:

- Adapting VDM objects to be serialized as JSON - This would improve compatability with other technologies.
- API for directly interacting with internal VDMJ functionality rather than just via a simulated command line.
- Optimisation improvements to allow fast, automated model testing.

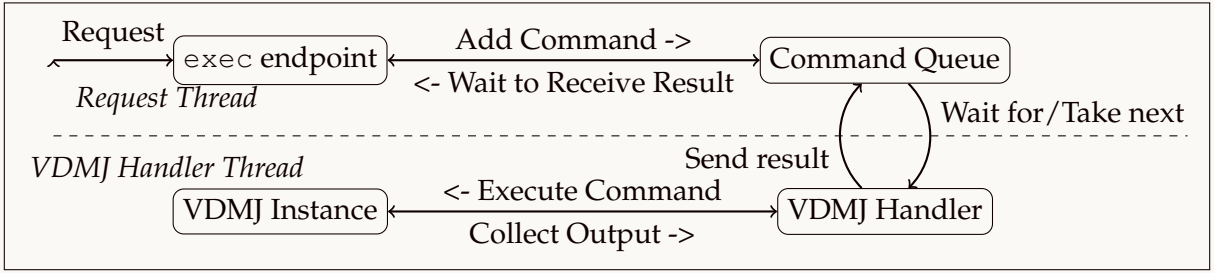


Figure 9. Parallel system for command execution, preventing race conditions

Figure 9 shows the system for queuing up commands for the VDMJ terminal, executing them, and sending back the result. This had to be implemented to prevent race conditions on simultaneous requests. While this has been tested, it has not been measured for high-throughput situations.

Potential Extensions/Modifications

Further Projects

A variety of options are available for futher projects:

- Adapting VDMJ-Remote to work as a Jupyter Kernel - This would require modification of both VDMJ-Remote and VDMJ to allow piecemeal parsing to a certain degree but would allow a VDM Notebook to run wherever Jupyter can.
- Develop Python VDMJ Bindings - This could provide an equivalent to the above, but with the additional benefit of easily embedding VDM models in a wide set of applications. An issue is that there is no Java-Python engine available that is up to date with current Python versions (at time of writing).