# Viral Agent-Based Model Interface
# Software Development Plan

**Version 1.0**

# Revision History

| Date | Version | Description | Author |
|---|---|---|---|
| 01/30/2026 | 1.0 | Filled out all desired material | Norwood, Ellion |
| | | | |
| | | | |
| | | | |

# Table of Contents

# 1. Introduction

## 1.1 Purpose

This Software Development Document (SDD) describes the design, architecture, and implementation details of the Viral Agent-Based Model Interface. The document provides developers, maintainers, and researchers with a detailed understanding of how the system is constructed, how its components interact, and how future modifications or extensions should be implemented.

The document complements the Software Requirements Specification (SRS) by explaining the internal design decisions used to implement the requirements defined in that document.

## 1.2 Scope

The Viral Agent-Based Model Interface is a desktop application that provides a graphical interface for configuring and executing an existing CUDA-based viral agent-based simulation model. The system allows users to configure simulation parameters, validate inputs, launch the simulation engine, and visualize simulation outputs without requiring direct interaction with the underlying CUDA source code.

The application integrates three major components:

- A **PySide6-based graphical user interface**
- A **CUDA-based simulation engine**
- A **file-based data exchange system for simulation parameters and outputs**

The system is designed to run on Linux laboratory workstations equipped with NVIDIA GPUs capable of executing CUDA simulations.

## 1.3 Intended Audience

This document is intended for:

- Software developers maintaining or extending the system
- Researchers running simulations using the interface
- Future contributors responsible for modifying the GUI or validation logic
- Lab administrators responsible for deployment and maintenance

## 2. System Overview

The Viral Agent-Based Model Interface provides a graphical environment for configuring and executing viral infection simulations using an agent-based model. Previously, users were required to manually modify C/CUDA source code and execute simulations through command-line tools. The GUI simplifies this process by providing structured input fields, automated validation, and visualization of simulation outputs.

Users interact with the GUI to configure simulation parameters such as cell counts, transmission probabilities, infection rates, and time settings. Once the user initiates a simulation run, the GUI generates a configuration file containing the selected parameters and launches the CUDA simulation engine as a separate process.

The simulation engine reads the parameter file, executes the viral agent-based model using GPU acceleration, and writes simulation results to output files stored on the local file system. The GUI then reads these files and presents the results in graphical and tabular formats.

This architecture ensures that the computational model remains separate from the user interface while enabling researchers to run simulations efficiently.

## 3. System Architecture

The Viral Agent-Based Model Interface follows a modular architecture consisting of three primary components:

1. PySide6 Desktop GUI
2. CUDA Simulation Engine
3. File-Based Results Store

### 3.1 PySide6 Desktop GUI

The graphical user interface is implemented in Python using the PySide6 framework (Qt for Python). The GUI is responsible for user interaction, parameter configuration, validation, and visualization of simulation outputs.

The GUI provides multiple pages for configuring simulation settings, monitoring simulation progress, and viewing results. Input parameters are collected through structured interface elements such as numeric input fields, sliders, and toggle controls.

Before a simulation is executed, the GUI validates all user inputs using the parameter validation module to ensure that values fall within biologically reasonable ranges.

The GUI also handles launching the simulation engine as a separate operating system process and monitoring its execution status.

### 3.2 Simulation Engine

The simulation engine is an existing C++/CUDA implementation of a viral agent-based model. It performs computationally intensive calculations using GPU acceleration provided by the NVIDIA CUDA framework.

The simulation engine is compiled using the NVIDIA CUDA toolkit during development and distributed as a precompiled executable for deployment.

The simulation engine reads configuration parameters from a text file generated by the GUI and produces output files containing simulation results.

Because the simulation engine is executed as a separate process, the GUI does not directly interact with CUDA code or GPU hardware.

### 3.3 File-Based Results Store

The file system serves as the primary communication layer between the GUI and the simulation engine.

Before launching a simulation, the GUI writes all parameter values to a configuration file (Parameters.txt). This file contains key-value pairs corresponding to the variables expected by the simulation engine.

After execution begins, the simulation engine reads the parameter file and writes simulation outputs to a designated results directory. These outputs typically include time-series data files, optional spatial snapshots, and log files.

The GUI reads these files after the simulation completes and uses them to generate visualizations such as plots and tables. All simulation inputs and outputs remain stored on disk for future analysis and reproducibility.

## 4. Parameter Validation System

To ensure simulation stability and prevent invalid model configurations, the system includes a parameter validation module implemented in Python.

The validation module verifies that all parameters fall within predefined numerical ranges and that required conditions are satisfied before the simulation is executed. These rules are based on biological constraints and client-provided parameter limits.

If invalid input values are detected, the GUI displays clear error messages and prevents the simulation from running until the issue is corrected.

Examples of validated parameters include:

- Number of layers
- Time step
- End time
- Probability of cell-to-cell infection
- Eclipse phase duration
- Infection phase duration
- Viral production rate
- Multiplicity of infection (MOI)

Validation also ensures that values are of the correct data type and that dependent parameters satisfy logical constraints.

## 5. Simulation Execution Workflow

The following sequence describes the typical execution process for a simulation run:

1. The user launches the Viral Agent-Based Model Interface.
2. The user configures simulation parameters using the GUI.
3. The GUI validates all parameter inputs using the validation module.
4. If validation succeeds, the GUI generates a configuration file containing all simulation parameters.
5. The GUI launches the CUDA simulation engine as a separate process.
6. The simulation engine reads the parameter file and executes the viral agent-based model using GPU acceleration.
7. During execution, the simulation engine writes output files to a results directory.
8. After the simulation completes, the GUI reads the output files and generates visualizations such as time-series plots.

This process allows the computational model to remain independent from the graphical interface while enabling efficient data exchange between components.

## 6. Project Structure

The project is organized into multiple directories that separate the GUI implementation, simulation models, and output data.

A simplified project structure is shown below:

```
Viral_Agent_Model_Interface/
│
├── View/
│   ├── RunPage.py
│   ├── paramValidation.py
│   └── GUI components
│
├── ABM_GERG/
│   └── Syncytia_Formation.cu
│
├── test/
│   └── Viral_Transmission.cu
│
├── runs/
│   └── simulation outputs
│
└── snapshots/
    └── spatial simulation images
```

This structure separates GUI logic from simulation code and ensures that simulation outputs are stored independently of the application code.

## 7. Dependencies

The system relies on several software dependencies required for both the GUI and simulation engine.

Required dependencies include:

- Python 3.x
- PySide6 (Qt for Python)
- NumPy
- Pandas
- Matplotlib

- NVIDIA CUDA Toolkit
- NVCC compiler
- Linux operating system with NVIDIA GPU drivers

These dependencies must be installed on the lab workstations before the application can run correctly.

## 8. Error Handling and Robustness

The system is designed to handle runtime errors gracefully to ensure reliability during simulation execution.

Potential error scenarios include:

- Invalid user input parameters
- Missing or corrupted parameter files
- Failed simulation execution
- Incomplete output files
- File system access errors

When such errors occur, the GUI displays descriptive error messages and preserves user input whenever possible. The system also prevents simulations from executing when required inputs are invalid.

## 9. Future Development Considerations

Several improvements may be implemented in future versions of the system, including:

- Real-time visualization of simulation progress
- Expanded support for additional viral models
- Improved graphical plotting capabilities
- Additional validation rules for new biological parameters
- Enhanced configuration management for saved simulation presets

These enhancements can be integrated without major architectural changes due to the modular design of the system.

## 10. Conclusion

The Viral Agent-Based Model Interface provides a user-friendly platform for configuring and executing GPU-accelerated viral infection simulations. By separating the graphical interface from the simulation engine and using a file-based communication mechanism, the system ensures flexibility, maintainability, and ease of use for researchers.

The architecture allows users to perform complex simulations without interacting directly with CUDA code while maintaining compatibility with the existing agent-based modeling framework.