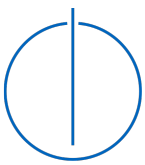
DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Flexible Visualization for Data Analytics on the Virtual Mobility World (ViM) Experimentation Platform

Syed Hassaan Tauqeer



DEPARTMENT OF INFORMATICS

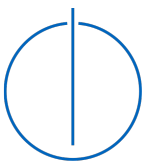
TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Flexible Visualization for Data Analytics on the Virtual Mobility World (ViM) Experimentation Platform

Flexible Visualisierung für Datenanalyse in der Virtuellen Mobilitätswelt (ViM) Experimentierplattform

Author:	Syed Hassaan Tauqeer
Supervisor:	Prof. Dr.-Ing. Jörg Ott
Advisor:	Dr. Ljubica Pajevic Kärkkäinen
Submission Date:	15.04.2020



I confirm that this master's thesis in informatics is my own work and I have documented all sources and material used.

Munich, 15.04.2020 Syed Hassaan Tauqeer

Acknowledgments

I would like to express my deepest gratitude to my advisor, Dr. Ljubica Pajevic Kärkkäinen, for her consistent guidance and mentoring throughout the project and thesis, specially during the meticulous review of the drafts of this document.

I would also like to thank Dr. Ilias Gerostathopoulos, for taking time out of his busy schedule to provide guidance and help with the CrowdNav platform.

Dedication

To my parents, Munazza Syeda and Tauqeer Abid, without whom none of my achievements would ever be possible.

Abstract

In a bid to enhance user productivity, this thesis aims to augment the analytics part of an existing experimentation platform, in order to make its workings more understandable and add avenues of customization. The experimentation platform currently deals with, and can perform statistical tests and run optimization processes on, a stand-alone, independent traffic simulation system to assess the performance of vehicle trips, customer complaints, and fuel consumption among other measurable and optimizable metrics. Modeling the experimentation platform's analytics module's core functions in an environment that is familiar to the users, we provide them with the ability to create experiments and view their results. We also provide users with the flexibility to add modules (specifically custom visualization plug-ins) to this project, if they find the need or if future iterations of the experimentation platform provide support for newer features. We do this by keeping this project modular and independent of the experimentation platform that it works in tandem with. By choosing the development environment that we built this project in, we have also achieved a more compute-resource friendly tool than the analytics part that was built within the experimentation platform.

Contents

1 Introduction

Data and its significance have recently taken the world by storm. The Economist went as far as calling data a new lucrative commodity [5]. However, hoarding a heap of data and storing it in a database, thinking it will bring more profit is as useless as not having it in the first place. What brings value to that data is the process of converting it into intelligible and actionable information.

As [7] notes, technologically-enabled companies are **26%** more profitable than their competitors. They also state that despite having data and a large number of data scientists, the number of companies successfully transforming data to effective insights is low. One reason for this is, data scientists and analysts spending more time extracting data and then porting it over to tools that they generally practice with, creating a rift between engineering and analysis and also unproductively spending valuable time.

With the advent of the Internet of Things (IoT) and connected devices, the amount of data generated has been exponentially increasing. To keep up with the pace of extracting, processing and inferring value worthy information from raw data, the tools that aid field-specific analysts have to be tailor-made to their specifications and according to their strengths [4]. This ensures that these data scientists and analysts spend more time contributing to plug-ins that they can write and implement within their comfort zone and on analysis rather than spending time in equipping themselves with other technologies to provide their expertise to the overall development and analysis ecosystem.

Porting these tools to environments that data scientists are comfortable with helps to make their work more focused and increases the output productivity of these individuals. They no longer have to focus on stressful installations and environment setups, nor do they need to go through huge chunks of full-stack code to determine how certain functions are written and operating. Another benefit of this, is the ability of analysts to run multiple experiments on these tools with slight variations to their parameters to see a possible effect on the results which might give them an idea on how their hypotheses tally in differing scenarios.

1.1 Problem and Motivation

This thesis is designed around, and is a continuation of, an on-going project that deals with experimentation performed on black-box software systems connected to the Virtual Mobility World (ViM) experimentation platform [2]. ViM is a web-based tool capable of handling automated experimentation and running optimization processes. The main application-related focus of this tool is the experimentation with simulations (and possibly real-world data) of traffic scenarios, which include routing, customer-satisfaction, overhead of trips that are running late and so on. The ultimate vision for this framework is to monitor and enable the decision-making processes for efficient traffic management, accident-avoidance and the management of ride-sharing, eventually resulting in smart connected vehicles acting as transportation fleets.

The platform is capable of running statistical significance tests like "Factorial Analysis of Variance (ANOVA)" and independent "T-test" on streams of data coming in via user-defined data connectors. It treats the test environment as a complete black-box model. However, the structure of how the black-box model will ingest experiment, settings and how it will output resulting data values, is controlled via defined instructions by the initiator of the experiment.

While the ViM platform checks most boxes on the smooth-sailing full-stack solution list, it shows a slight hint of uneasiness at being a flexible analysis tool. Its frontend can handle well developed and precise experiment orders but it is not made for an ad-hoc A/B comparison, where one just wants to change one slight parameter and compare the outcome with a previous result. Retrieving other results is also a step that requires navigating back to all experiments and then choosing the required one. Moving back and forth between tests makes a quick result comparison quite frustrating. Furthermore, the frontend is designed and developed in "TypeScript"[18], a language that is probably quite well known within the web-development community but not so much in the data science community. This creates the possibility of delays when a data scientist/analyst needs to add in custom visualization options to represent the experiment results in a newer form. Not to mention the arduous process of installing the frontend (while managing all its dependencies) and the toll it takes on the computing resources every time it is run.

1.2 Goal

The goal of this thesis is to make the analytics part of the experimentation platform more accessible to data scientists by re-designing and re-implementing it in a more

familiar setting to said, scientists. This would then enable them to run experiments on the simulation environment and aid in analyzing data from the simulation/real-world environment.

Concisely, that would entail:

1. Familiarizing with the ViM platform prototype.
2. Performing background research of open-source and proprietary tools and frameworks for experimentation analytics.
3. Conceptualizing the process of run-time and post-experiment interaction with the platform, as well as the functionalities for analyzing experiment results
4. Designing the frontend data analytics module.
5. Implementing the devised solution and evaluating it via use-case scenarios.

1.3 Overview of Approach and Contributions

Since the existing code base of the project was extremely large, and understanding the core functionality by solely reviewing the code was not a feasible option, the only solution available was to analyze the communication of the front and back ends of the platform to figure out the structure of instructions passed to and fro.

After the instruction templates for defining "Target Systems" and "Experiments" were saved, forms for querying user input were created and linked to the instruction templates. Instructions could then be sent directly to the backend for either creating a new target system or experiment (on a selected target system). Since target system creation is the most basic step, there is no prerequisite for it, apart from the fact that the target system (black-box simulation environment) should be running in the background (so when the backend sends a configuration, there is a system listening to it) before it is defined using this solution. For experiment creation, the prerequisite is the selection of an already defined and running target system (so that the experiment instructions are sent to a simulation that is listening to the backend).

Although all three modules, target system creation, experiment creation and result retrieval work independently, the functionality of the result retrieval is kept separate from the other two. That is because one could always choose to analyze previously

run experiments without creating any newer ones. The prerequisite for result retrieval does not require any black-box system running in the background (since successful experiment results are saved in a database). However, it does require selecting an already completed experiment. The key contributions of this thesis, in essence then,

are as follows:

1. Implementing the ViM frontend of the tool in a Jupyter Notebook environment.
2. Functionality for creating "Target Systems" (the structure which defines how the ViM backend will communicate with the black-box simulation environment).
3. Functionality for creating new experiments that run on target systems defined in the same session or at an earlier stage (the experiments are also defined in a set structure that is understandable by the black-box simulation environment).
4. Functionality for retrieving and displaying experiment results from the platform via queries to the backend (current iteration supports the results for experiments that are already done running).

1.4 Road-map of the Thesis

So far, this thesis has tried to cover the ground for its need. The chapter following this one, **Background and Related Work**, gives the background of the project, including the ViM platform. In addition, it will give a brief overview of work, done by the community, that is relevant to this project and from which the project draws inspiration. The chapter **Exploring ViM** will then give a detailed view of the ViM platform and the CrowdNav simulation platform. Afterward, the chapter on **System Design** shall describe how this project came together, including its design and architectural aspects. In **Evaluation**, the study of the use-cases of the project shall show the workings of the newly developed prototype and benchmark their performance while the final chapter will mark the conclusion and the closure of this thesis.

2 Background and Related Work

2.1 Project Background

The Virtual Mobility World (ViM) Experimentation Platform is a web-based application that acts specifically as a statistical significance testing and optimization tool. The tool, itself, has no data generation sources, nor does it have the capabilities of running simulations in and of itself [10]. It merely acts as a platform that can stage experiments on simulations that are running independently of the tool.

One could think of ViM as an abstract control deck for setting up and monitoring various simulations and experiments on multiple simulation environments, each having an array of different control mechanisms. ViM accomplishes this generalization of interaction by treating each simulation environment as a black-box [10].

The current iteration of ViM supports a traffic management simulator, called CrowdNav. Users can set up statistical significance experiments to monitor various factors of interest (in case of ride-hailing services) such as trip overheads, fuel consumption, customer satisfaction, etc. and then optimize the parameters to minimize the negative aspects and maximize the positive ones.

The ViM client is extremely robust. However, it lacks the option of being customizable and is not very compute-resource friendly. The current technology that it was developed on (Angular and TypeScript) requires building the solution every time it has to be run. Not only does it occupy memory resources, but it also requires special expertise in the web-development domain to add custom visualization plug-ins to the tool. A regular user of the tool, a data scientist, is probably not the sort of person best equipped to handle making changes to ViM.

Greg Wilson, mentions in [25] that most scientists are not very adept at industry practices of software engineering, which makes them very likely to spend a lot of time debugging software systems to get them running. This, in turn, consumes a lot of time that they could have spent on studying the data and drawing out results and conclusions, which is something that they excel at. Therefore, it is important to bring the functionality of the ViM client to a more compute-resource friendly platform which

also provides a familiar setup to the user of the tool and allows them to be as involved with its customization as they need, to be more productive. In the next section of this chapter, we look towards some industry leaders in order to obtain inspiration for the best industrial practices, which might help us achieve this goal.

2.2 Related Work

This thesis is inspired by the idea of enhancing productivity and bringing a feeling of inclusivity to widely diverse teams so that the sum of the team's achievements is effectively increased by each individual's contributions, and that is what we aim to look for in the work done by industry leaders. It is important to mention that there is not much related work in the area of enhancing the productivity of data scientists because incorporating data scientists in the folds of production-ready applications is a relatively nascent process and many companies are still trying to figure it out. However, we will investigate a couple of existing solutions, in this section.

2.2.1 Tempe

Tempe is a web-based, large-scale exploratory data analysis (EDA) tool, for temporal data [8]. It was designed by Microsoft to increase the productivity of their data scientists in addition to allowing them to collaborate on projects by easily sharing their work through project page Uniform Resource Locators (URLs).

Tempe actively computes analyses which means that it fetches query results and starts showing them instantly instead of waiting for the whole result to finish compiling. This makes it extremely responsive with large data sets. This feature also allows it to achieve its aim of eliminating the Read-Eval-Print-Loop by showing a user the result of their change instantly without having them to first print the output then make the change and print the output again to evaluate if their required function was operating.

This then, is the perfect EDA tool (just like ViM, in our case, is a great experimentation platform), except that it shows the same limitation that ViM does. The frontend client is implemented in JavaScript (a web-based language) and to add more functionality to the visualizations, an analyst would have to know JavaScript or spend time learning it.

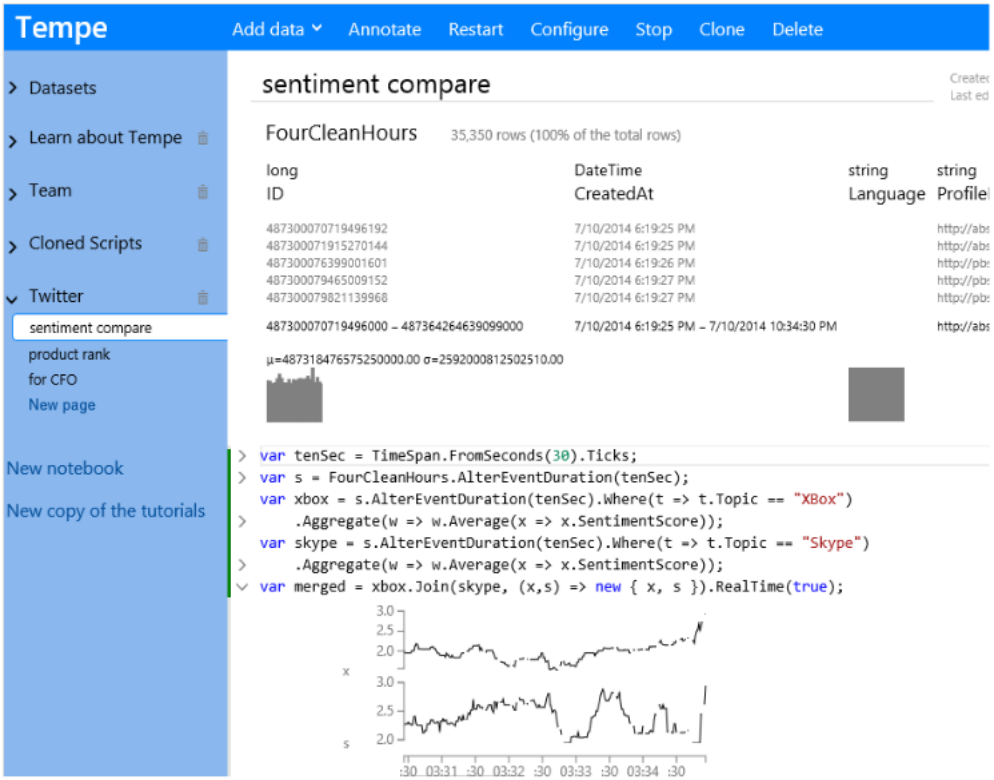


Figure 2.1: Tempe showing an analysis notebook (Figure from [8])

2.2.2 Netflix XP

Netflix XP is Netflix's experimentation platform. In [4], the research team describes the idea of re-designing the platform around the convenience of Netflix's data scientists to give them the freedom to work with the tools that they are familiar with. It is a tool that tries to bring production and ad-hoc analyses in the same environment to avoid the use of two different environments for two different analysis types. The benefit of this approach is the feature to push successful analyses to production work-flows and similarly move production setups to the drawing board, if there is a need to change them.

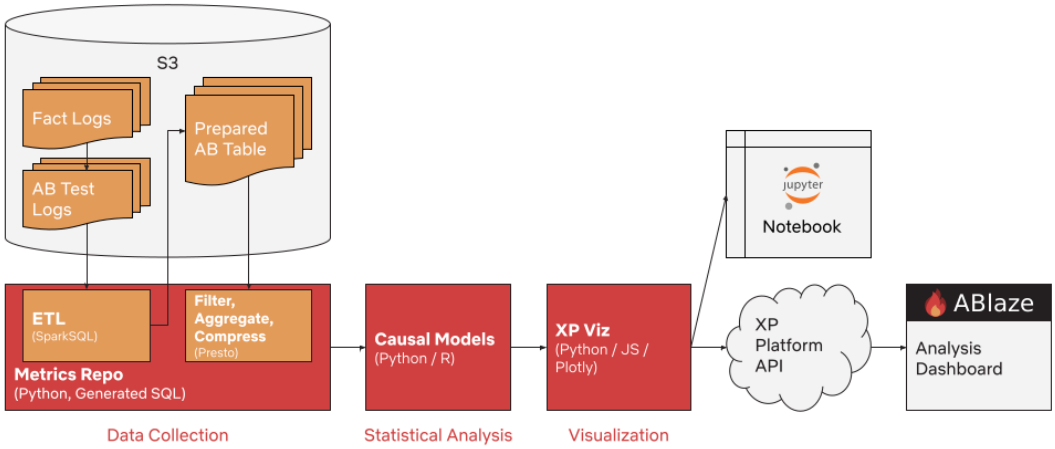


Figure 2.2: Netflix XP Experiment Analysis Flow (Figure from [4])

Figure 2.2 shows the experimental analysis flow for Netflix XP which has three main parts of design; "Data Collection", "Statistical Analysis" and "Visualization". The data is first collected, filtered and aggregated after which it passes on to the analysis module. Here, statistical tests are performed on the data and the results are then passed to the visualization module. This part can be run as a Jupyter Notebook for an ad-hoc analysis or the results can pass through the platform API to be displayed in the production dashboard.

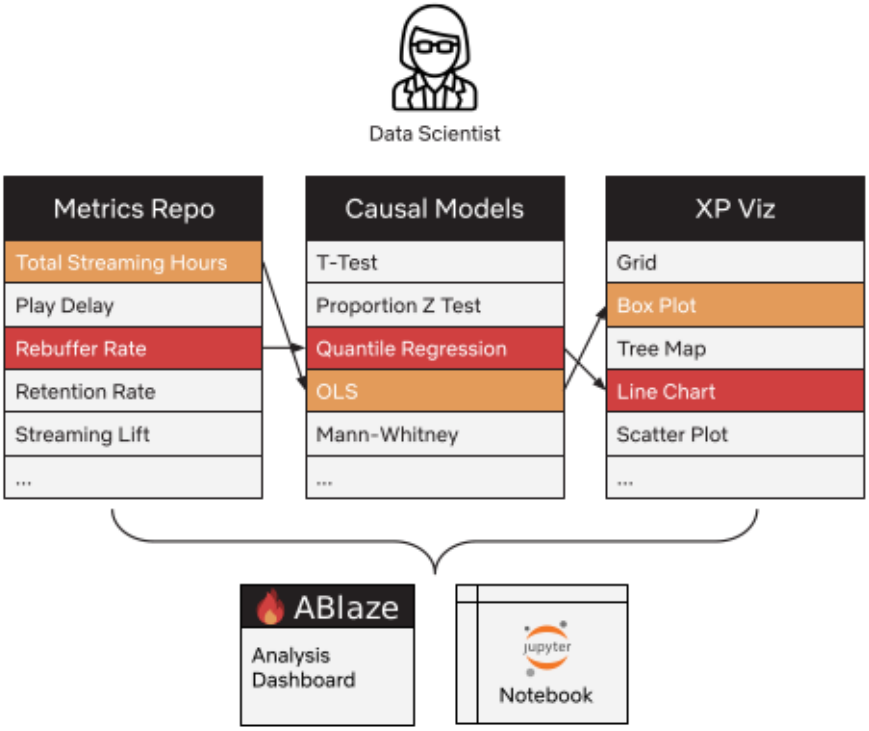


Figure 2.3: Examples of Analysis Flows in Netflix XP (Figure from [4])

The end result of this tool, is a very customizable experimentation platform. It gives the user the ability to mix and match data sources, statistical tests that are run on these sources and visualizations used to view those test results, Figure 2.3 shows this. Partly, this design for experimentation flow is the inspiration for updating ViM through the project that this thesis describes.

Netflix, being a leading online media service providing company, is one of the most vocal advocates for the need to enhance productivity through complementing various technical roles in order to obtain the most value out of them. Every day, Netflix sees 1 trillion events written to its streaming ingestion pipeline [19], upon which 150 thousand jobs are run. To gain the most insights from the treasure trove of data that they store, the data team at Netflix have brought together data engineers (responsible for extracting and transforming data), data scientists (responsible for performing statistical tests and machine learning tasks on data) and data analysts (responsible for interpreting the

results through visualizations) through a common medium; Jupyter Notebooks (see Jupyter Notebook in Appendix). At its core, Netflix has developed an ecosystem around Jupyter Notebooks. Tools developed for services can be run through scripts running on these notebooks. Interactive forms can query user input and run experiments over and over again through notebooks and instantly display meaningful visualizations. Data and tasks flow seamlessly through data pipelines in a well-synchronized dance all the while being shareable and having the ability to be applied in production once they have passed the requisite testing.

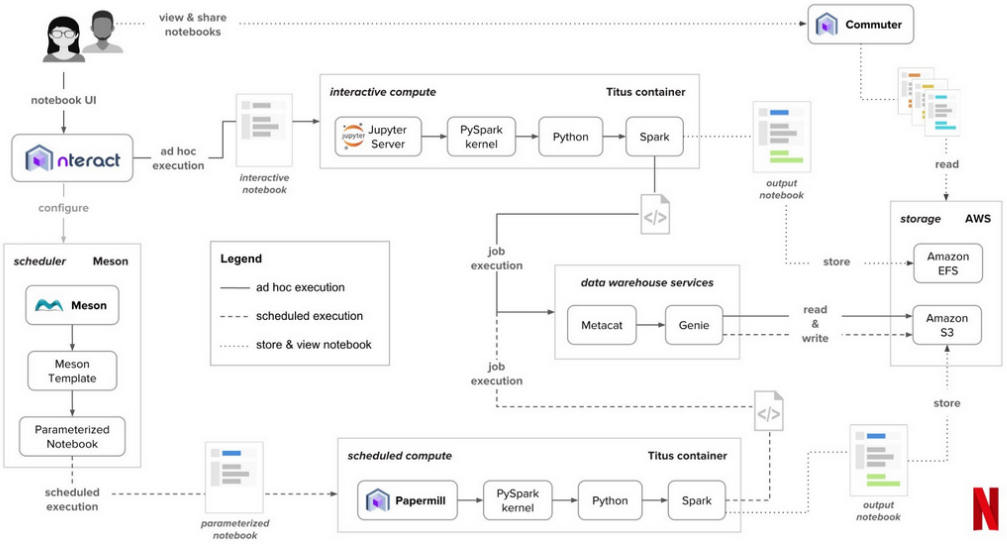


Figure 2.4: Notebook Ecosystem at Netflix [19]

3 Exploring ViM

The ViM platform is a newer iteration of "Online Experiment Driven Adaptation" (OEDA) [10], with essentially similar characteristics. This section will delve deeper into the project that this thesis is hoping to augment along with the technologies it uses.

For all intents and purposes ViM considers simulation environments as black-boxes. The simulation environment that we discuss here and the one that ViM currently supports is CrowdNav, a traffic management simulator.

3.1 Virtual Mobility World (ViM) Experimentation Platform

The black-box (simulation environment) parts visible to the ViM tool are only its data providers. The settings for simulations are also sent via the same data provider. This abstraction makes ViM a very robust experimentation tool. The component diagram (see Fig 3.1) for ViM shows how the tool is designed and structured.

The frontend "client" consists of two parts. The first part, is a collection of forms and parameter defining web-pages that are designed statically with HTML, while their functionality is written completely in Typescript. The second part is, the result visualization functionality, also written in Typescript.

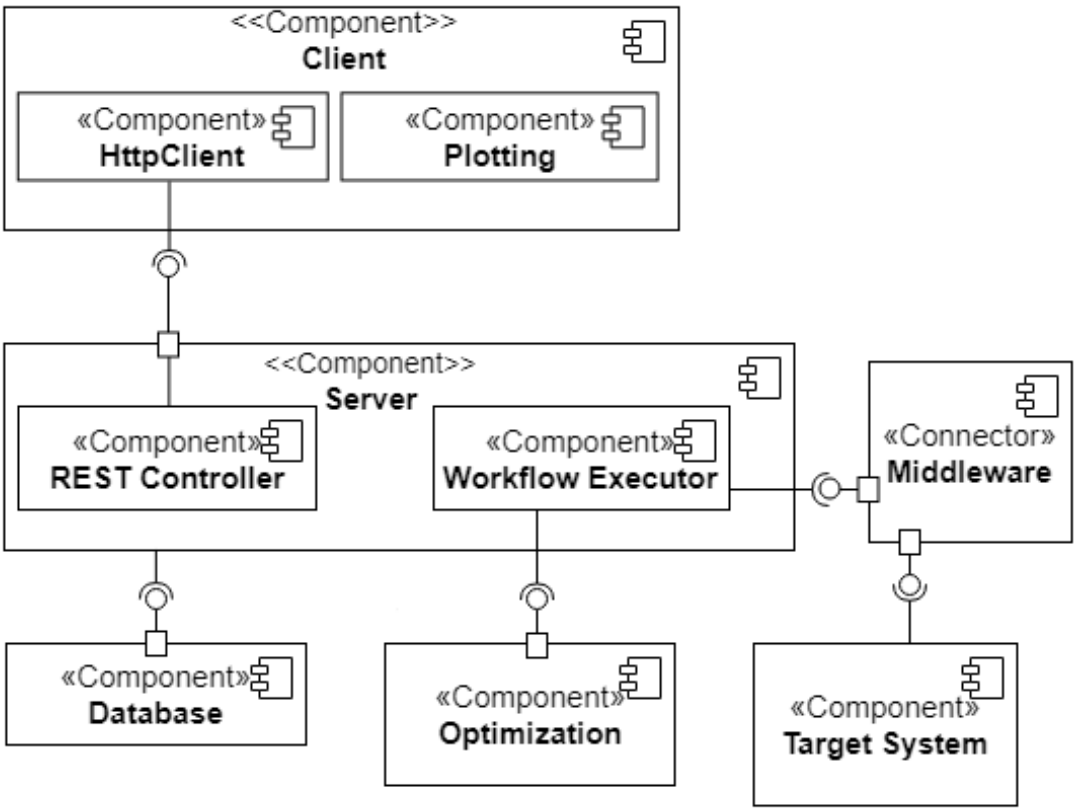


Figure 3.1: Component Diagram for ViM (Figure from [9] [21])

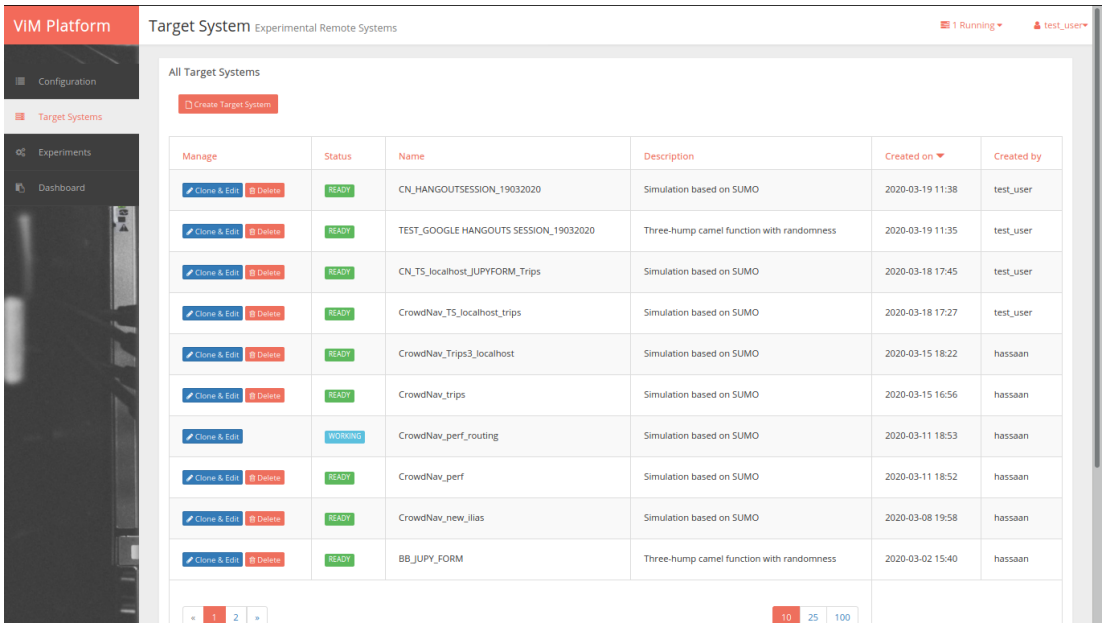


Figure 3.2: ViM Target Systems List

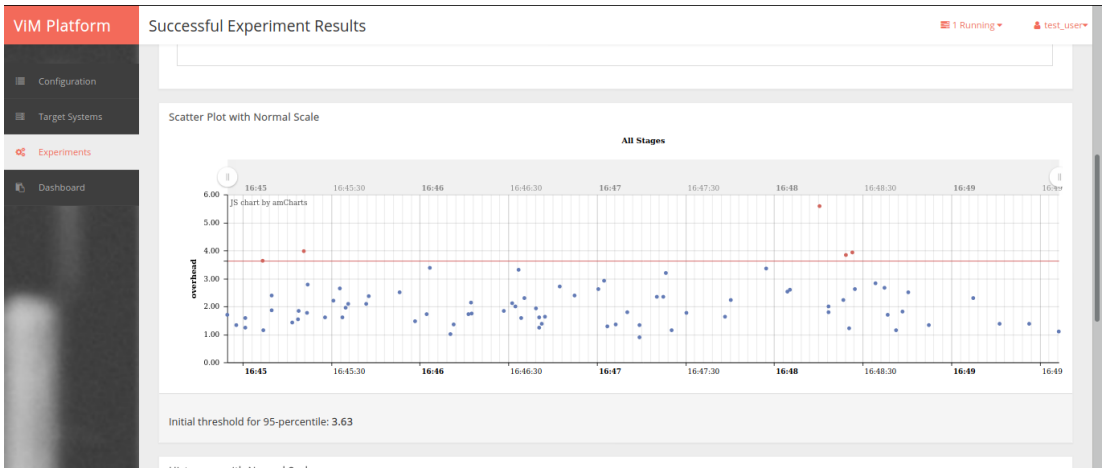


Figure 3.3: ViM Result Scatter Plot for an Experiment

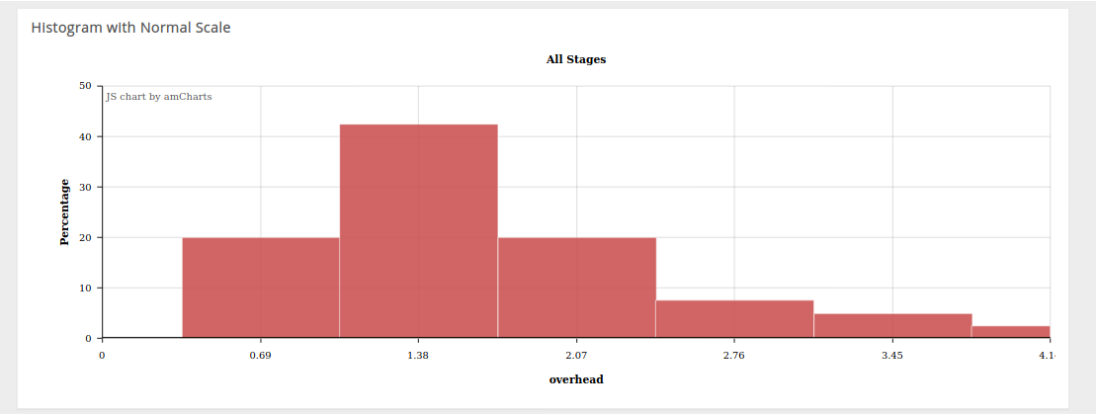


Figure 3.4: ViM Result Histogram for an Experiment

The client communicates with the backend server via a Representational State Transfer (REST) Application Programming Interface (API). REST communication is useful owing to the statelessness of the protocol. The server does not need to store client state between requests and that means it can focus on providing results requested by the client.

The server is ViM's brain and is written in Python. Having received the client's request, the server analyses what tests the client has requested and on which simulation environment. It then arranges for them to be set up on the platform. Once the tests have been set up (Fig 3.5), the server then constantly communicates with the simulation environment that was chosen for the tests and starts obtaining the results of the running simulation in real-time. Alongside, the server starts running the chosen test on the in-coming simulation data and starts gathering the result for each step (iteration) of the test. At this point, it is also streaming those results back to the client, which is continuously pinging the server via REST calls, Figure 3.6 shows this. The client then displays the results, by plotting them with interactive and meaningful visualizations, as can be seen in Figures 3.3 and 3.4.

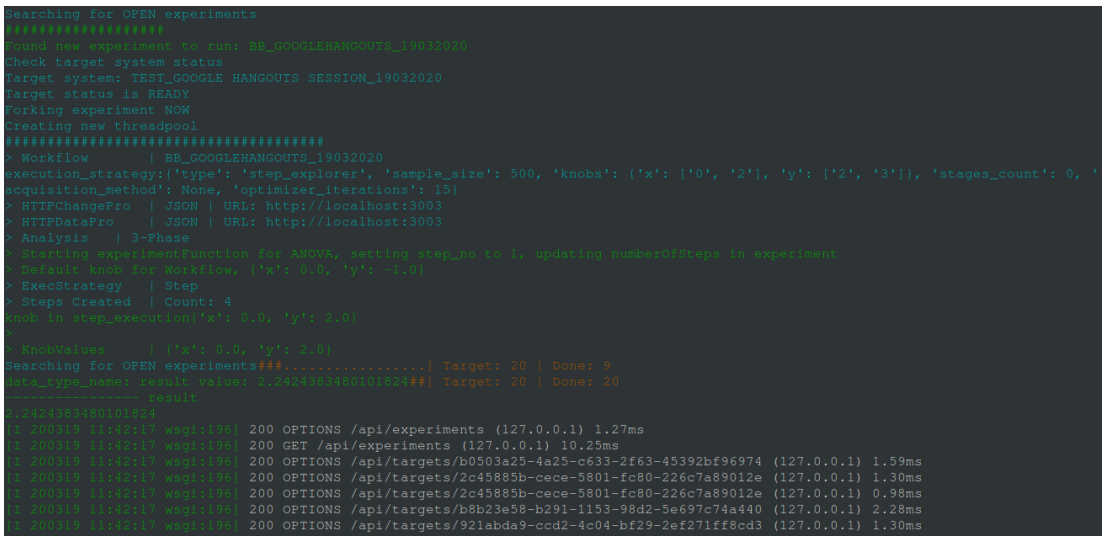


Figure 3.5: ViM Backend Experiment Settings

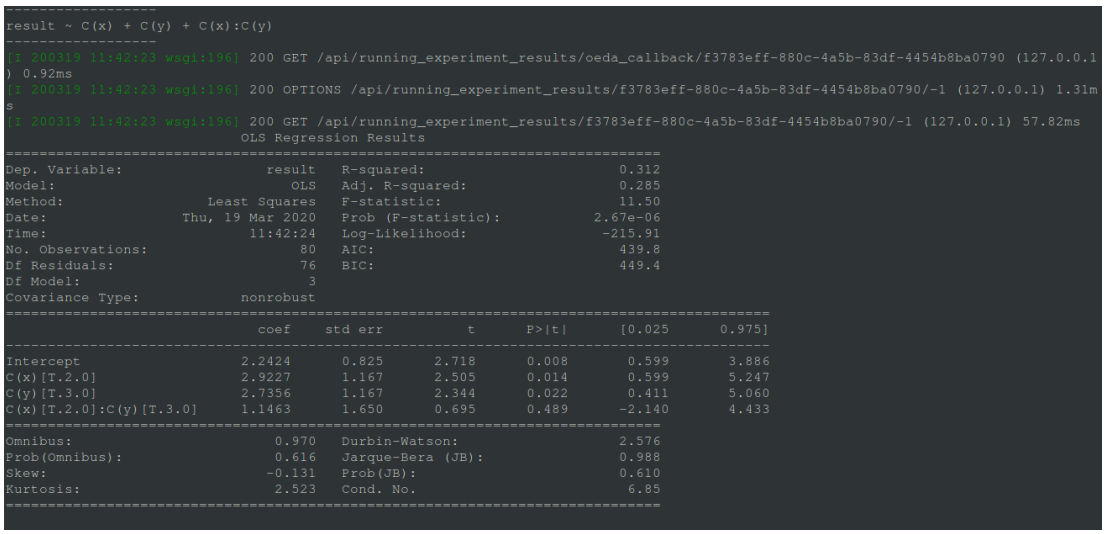


Figure 3.6: ViM Backend Result

3.1.1 Elasticsearch

Elasticsearch is a distributed, highly scalable open-source search and analytics engine used for storing, searching and analyzing large amounts of data, including textual, numerical, structured and unstructured information [6].

Data is stored as JavaScript Object Notation (JSON) documents. While storing, data is normalized and indexed so that even when it is stored in a distributed storage set up, Elacticsearch's ability to run quick text searches enable data fetching to be extremely fast and responsive. In addition to being fast, Elasticsearch is also resilient. The distributed data shards are duplicated (for redundancy) in order to provide fault tolerance in case of hardware failure.

In the ViM project, the server is connected to an Elasticsearch database that keeps track of, and stores all target system (black-box) configurations that have been defined within the ViM platform. This database also contains experiment configurations for experiments that have successfully finished along with their results.

3.2 CrowdNav

CrowdNav is a traffic routing and navigation system built on top of, Simulation of Urban Mobility (SUMO) using Traffic Control Interface (TraCI).

3.2.1 Simulation of Urban Mobility (SUMO)

Simulation of Urban Mobility (SUMO) is a free and open-source traffic simulation environment [12] [14], developed by German Aerospace Center - DLR. SUMO supports the creation and modeling of virtual world maps (road networks, signals, etc.) along with multimodal traffic including vehicles, public transport and even pedestrians. Vehicle routing within the simulation [15] follows Dijkstra's algorithm [22] by default where map intersections represent nodes and roads represent edges between those nodes. Routing also supports the A-star (A*) search algorithm [23]. Traffic lights can be interacted with using timing schedules that can be generated or imported from pre-existing configuration files. They control which vehicles are allowed to move at intersections.

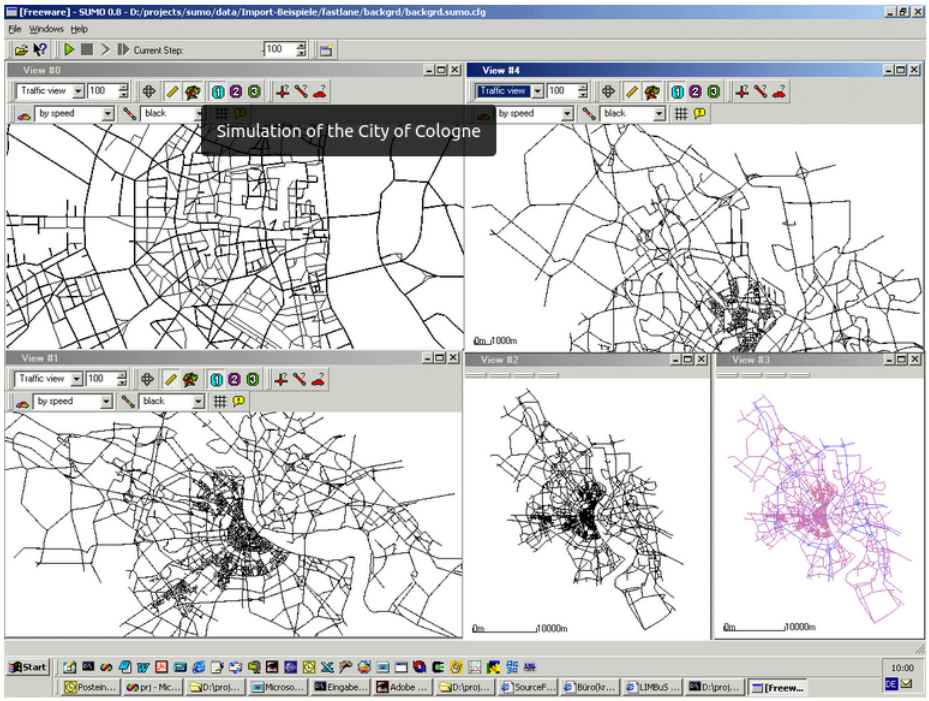


Figure 3.7: SUMO by DLR (Figure from [16])

Traffic Control Interface (TraCI)

Traffic Control Interface (TraCI) [17] is a tool that allows interaction with SUMO by allowing the interacting program to set and retrieve positional data of the simulated world and its objects. TraCI uses a Transfer Control Protocol (TCP) based architecture to access SUMO and can engage multiple clients.

Apache Kafka

Kafka, licensed under Apache, is an open-source distributed messaging system that acts as a streaming platform [13] [1]. It has two main actors; producers and consumers, and is run as a cluster on one or more servers. Clusters store records under topics. Producers can publish streams of records to one or more topics, while consumers can subscribe to one or more topics to read the published records. Kafka can reliably be used to create real-time data streaming pipelines. In CrowdNav, trip, performance and

routing data is published to Kafka under their respective topics ("CrowdNav-Trips", "CrowdNav-Performance" and "CrowdNav-Routing") and when a ViM experiment is running, it opens a consumer to read the data stream from whichever topic was specified in the experiment analysis.

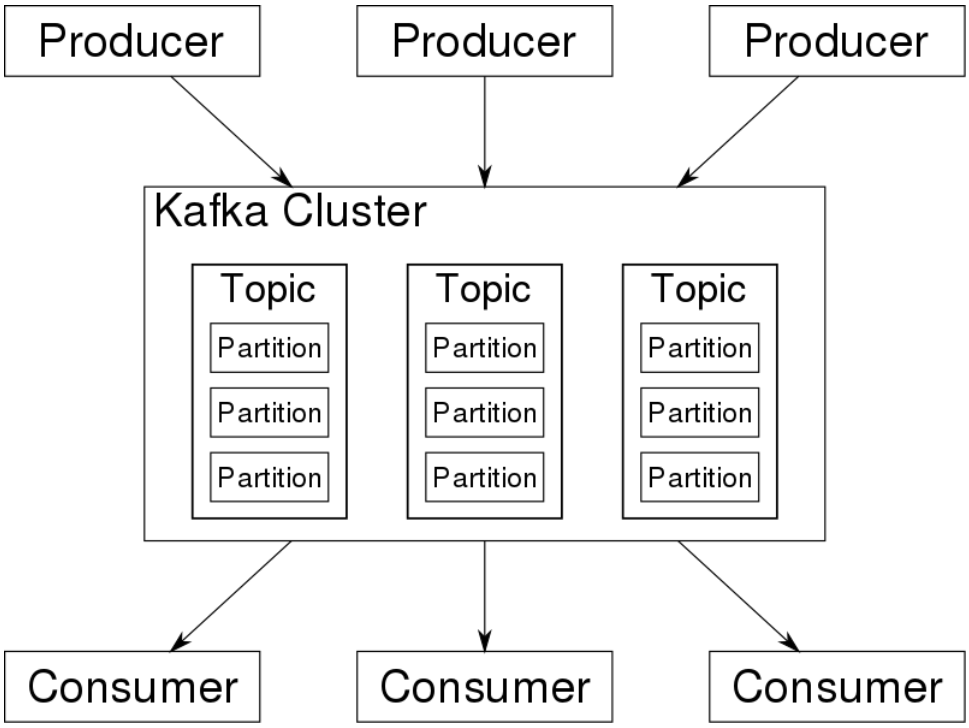


Figure 3.8: Apache Kafka Structure (Figure from [3])

4 System Design

So far, we have gone through the motivation and inspiration behind building this project. We shall now look into the important decisions that have gone into the architecture of the system and see how the design molds around the functional and non-functional requirements.

4.1 Functional Requirements

All functional requirements are recorded from a data scientist's/analyst's perspective, who shall be operating the system.

1. **Automatic credential handling**: The ViM platform has credential checks in place in order to authorize a user and issue a token for the working session. Therefore, the system should handle auto login for the user so that they can focus on their work.
2. **Creating target system**: The ViM platform recognizes simulation environments as target systems within its platform. The user should, therefore, be able to specify important inputs such as the data providers, variables involved and general settings of the black-box that ViM is to interact with.
3. **Creating experiments**: The user should be able to select what type of experiment they want to run (Factorial ANOVA or T-test). They should also be able to specify experiment details such as which variables to run the test on, which data provider to perform the aggregation on, the values for each variable that the experiment is to be run on and how many samples for each input variable combination are to be observed.
    a) **Selecting target system**: Each experiment is tied to a target system. The user should be able to select the target system upon which they would like to run the experiment, from a list of all defined target systems.
4. **Retrieving experiment results**: The user should be able to observe the results of an experiment after selecting it from a list of successful experiments.

4.2 Non-Functional Requirements

The non-functional requirements for the project are as follows:

1. Since the system aims to make life easier for the users instead of complicating it, it should also go a step further and not add any overheads in the installation of separate packages and supporting software that would hog computer resources.
2. The system shall be robust to enable using it as a stand-alone solution not tied to the entire project resources (except the ViM server)
3. The system shall hide all complexities from the user and provide a clean, simple and intuitive interface.

4.3 Architecture

The project is a Jupyter Notebook tool that is developed to act as ViM's frontend, but only for important specific functions mentioned in the functional and non-functional requirements of this chapter. We shall, for the rest of this thesis, refer to the project as "JupyViM" in order to save time and to establish a clear relationship between the two. Being written entirely in Python, JupyViM brings homogeneity in the ViM ecosystem, as the ViM server is also written in Python. Figure 4.1 shows the component diagram describing how JupyVim integrates in the ViM environment. The REST interface in JupyViM seamlessly communicates with the REST controller in the ViM server just as the ViM client does.

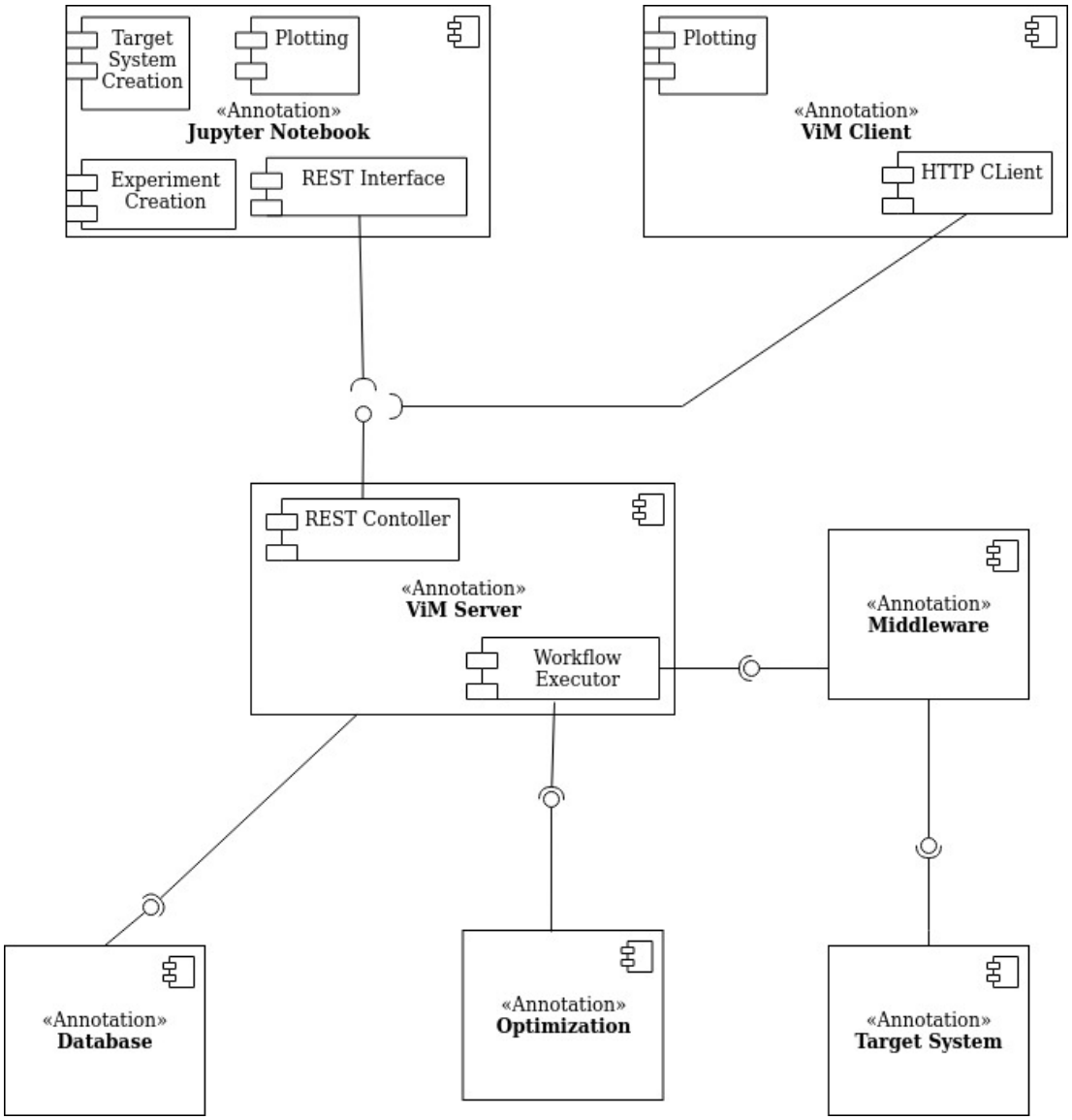


Figure 4.1: JupyViM Component Diagram

4.3.1 Design Decisions

The two main external libraries used in this project were **Ipython Widgets**[1] and **Plotly Express** [2]. Having established the work environment for the project to be Jupyter

[1] `https://ipywidgets.readthedocs.io/en/latest/`
[2] `https://plotly.com/python/plotly-express/`

Notebook, the only library that offered widget creation and event trigger support was Ipython Widgets. The library enables a low-level interaction approach, where users can define, style and create their own widgets based on a handful of starting widgets and containers. Ipython Widgets also enables users to define and tie event handlers to most of its basic widgets like buttons, drop-down menus, radio buttons, etc. and the fact that the library is written mostly in Python and JavaScript, gives users the freedom to interact with the backend Jupyter JavaScript functions and HTML page design using Python syntax which is natively more familiar to them.

While there were many choices for plotting and visualization in JupyViM, the most elegant solution was presented by Plotly Express. Plotly provides interactive visualizations where a user can comb through individual data points to find their exact coordinate values. Providing data to the plotting functions is extremely easy as it accepts DataFrames (collection of records in a table structure), while styling is also quite straight forward where the user has to bundle up all styling commands in a dictionary and pass it as an argument. Many other data visualization libraries were either too basic in function or were too complicated and verbose when it came to defining plot structures.

4.3.2 Building Blocks

JupyViM consists of several parts that come together to act as a model-view-controller (MVC) application. Figure 4.2 shows what constitutes an MVC application. The user interacts with the controller, using the controls provided in the application. The controller, relays user input to the model. The model is responsible for updating the view in the application to show the user the result of their input. This entire cycle continues if the user interacts with the application again.

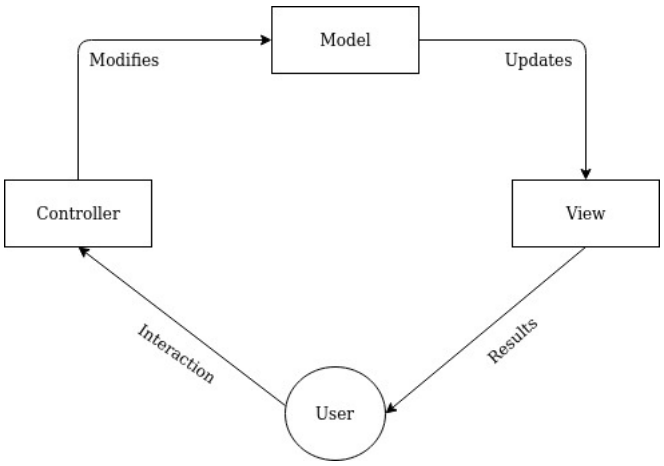


Figure 4.2: Model-View-Controller (MVC) Diagram

Forms

Forms are the backbone for requesting user input. Each form has to be designed separately and then joined with the main form container (basically an empty placeholder). Depending on their functionality and the option that the user selects, these forms need to be switched. Figure 4.3 shows a sample form for creating a 3HCF (See Three-Hump Camel Function in Appendix) target system. An important point to note here is that once the form is filled and is about to be sent to the server, the field names of each text-box are used to pull the information entered in them. Essentially, all elements in a form are independent elements that visually make up a single entity. Forms could be considered as the controller structure from the MVC framework that hides the complexities of the entire application and provides a clean interface.

Class Structures

Classes act as templates to avoid cluttering and to maximize the re-usability of code. Forms are viewed as objects of classes. Before sending the contents of an active form to the ViM server, the form content is pulled and plugged into the class structure that the form is an object member of. Classes in JupyViM can be thought of, purely, as advanced data structures, in that they just hold the data passed into them and not really perform any specific operations on that data. One very helpful function that these class structures provide is that, in Python, each class object can be cast into a dictionary object. This is very fortunate since data communication between JupyViM and the ViM server is through JSON packets (essentially dictionary structures). Figure 4.4 and Figure

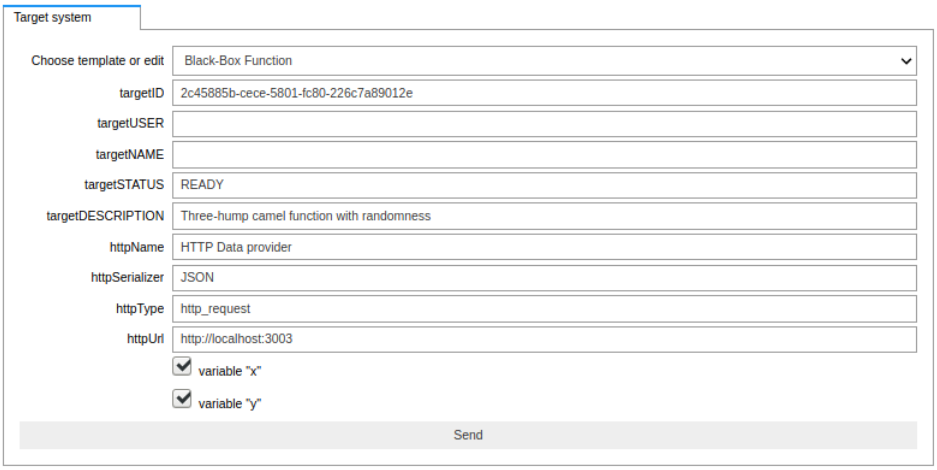


Figure 4.3: JupyViM Sample Target Form

4.5 show the class diagrams for supported black-box structure target systems, while Figure 4.6 and Figure 4.7 show the class diagrams for supported black-box experiment structures.

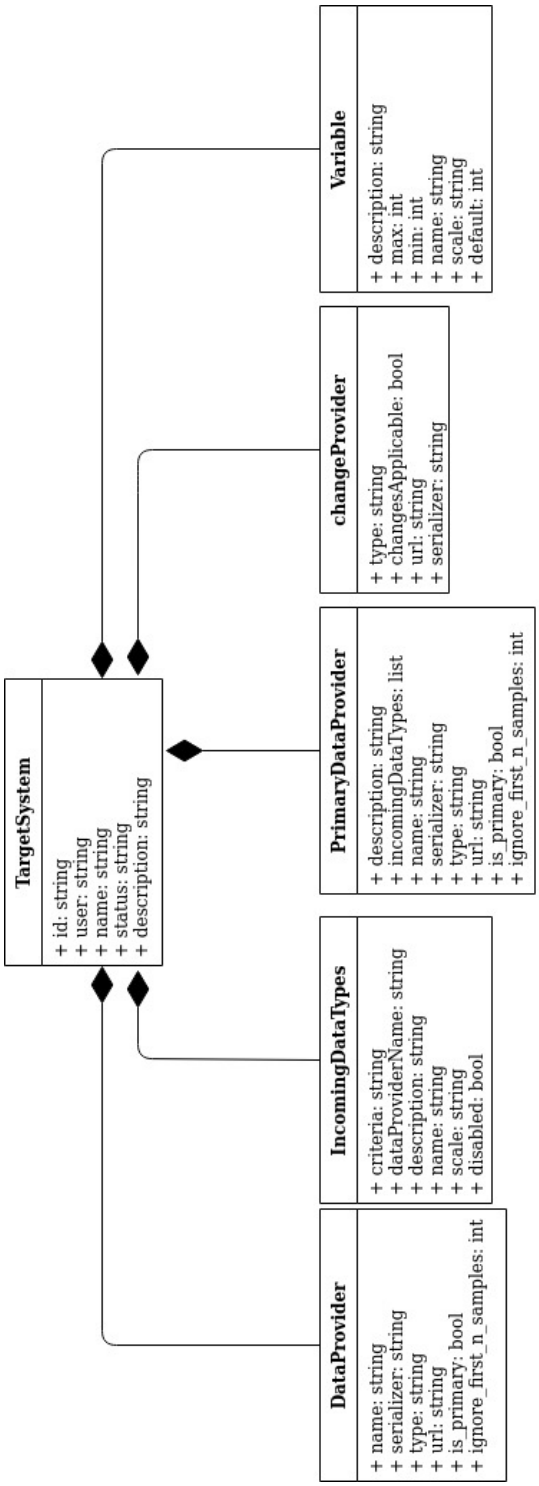


Figure 4.4: Class-Diagram For HTTP 3HCF Target System In JupyViM

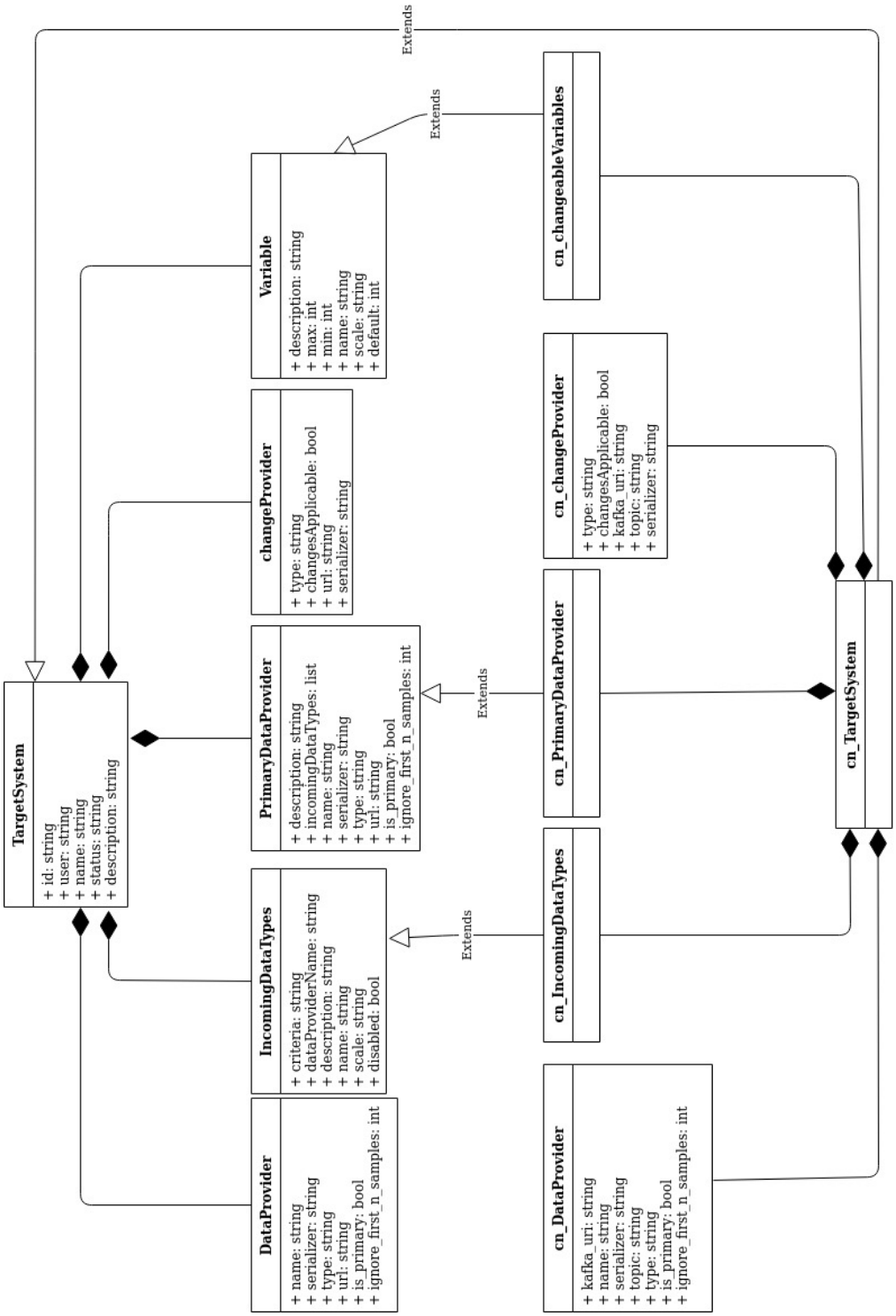


Figure 4.5: Class-Diagram For CrowdNav Target System In JupyViM

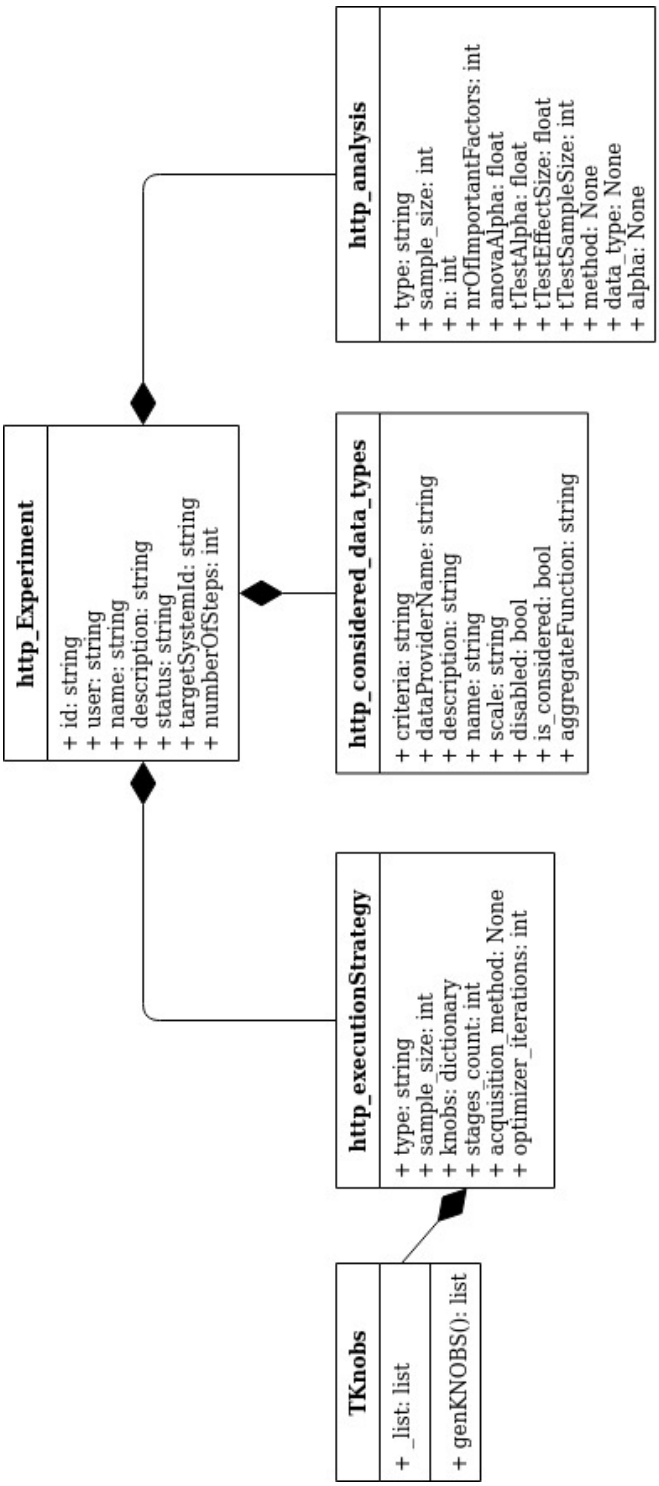


Figure 4.6: Class-Diagram For HTTP 3HCF Experiments In JupyViM

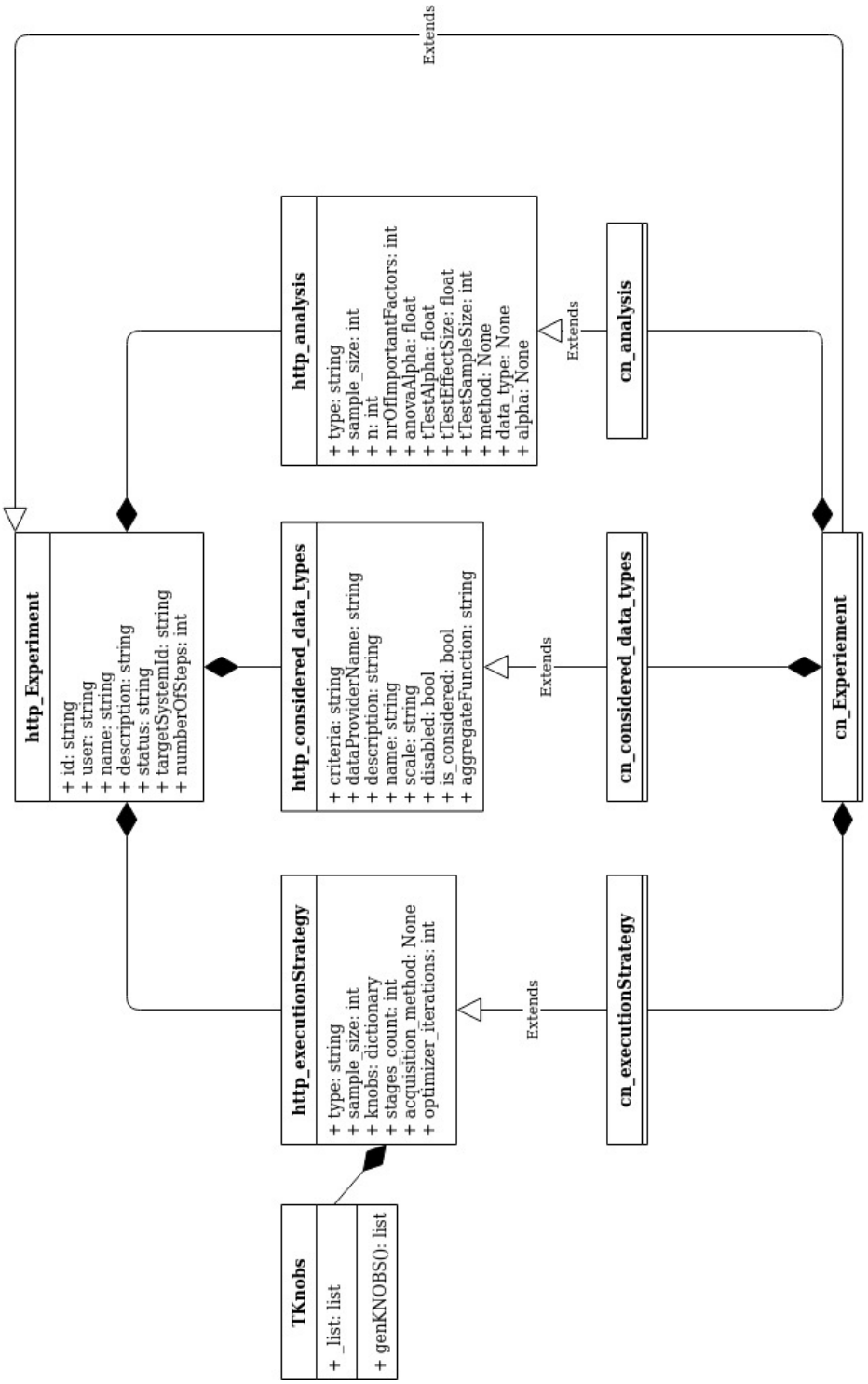


Figure 4.7: Class-Diagram For CrowdNav Experiments In JupyViM

REST API

The REST API is a communication layer that joins JupyViM and the ViM server together by allowing them to communicate back and forth. It contains all the essential call sequences that initiate server responses from ViM including the default user login (as ViM implements a user-token based session, it is imperative to first register the user to obtain the token before any further operations take place). Depending on whether the current initiated operation expects a list or result of some sort or whether a form submission is to be made, JupyVim sends GET or POST requests, respectively. Figure 4.8 shows a sequence diagram that explains how the calls from the REST API are forwarded to the server and the responses that it returns.

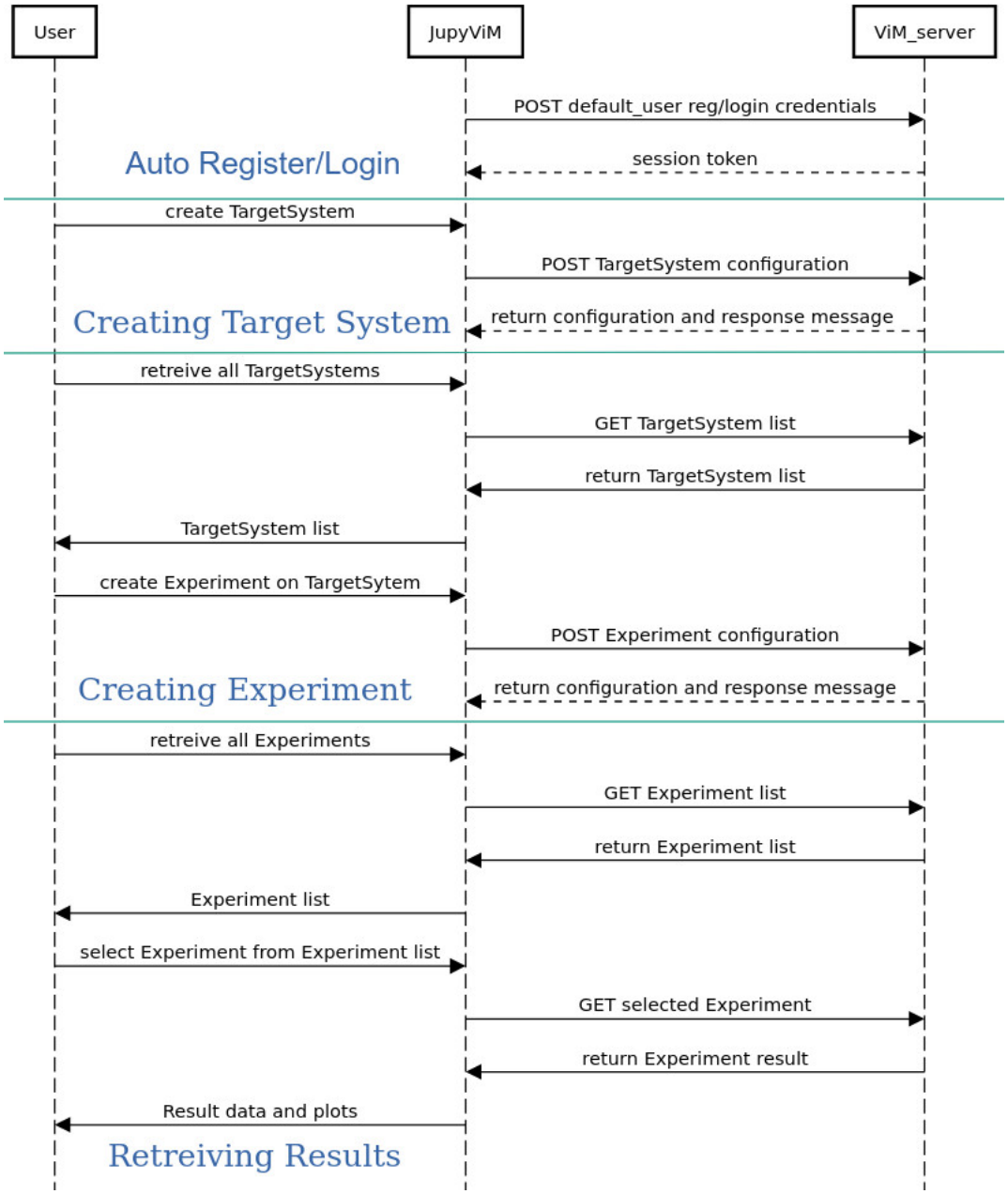


Figure 4.8: JupyViM Sequence-Diagram

Application

The application is where it all comes together. Since the view is dynamic, the forms are joined to their placeholder containers on-the-fly depending on user input. Once they are filled and the submission event has been triggered, the details in the forms are mapped to the objects of their respective classes. Object dictionaries are then cast into JSON packets and sent to the server via REST calls (according to user input —depending on target system or experiment creation). The application also has a result viewing option, where it first queries the server for all experiments that have successfully ended. As soon as the user selects an experiment, the plotting functions with-in the application un-pack the data packets received from the server and proceed to plot them. The model and view parts of the MVC are collected and managed by the application component. The sequence diagram in Figure 4.8 shows this, while Figures 4.9 and Figure 4.10 show the project file structure and how they are linked with each other.

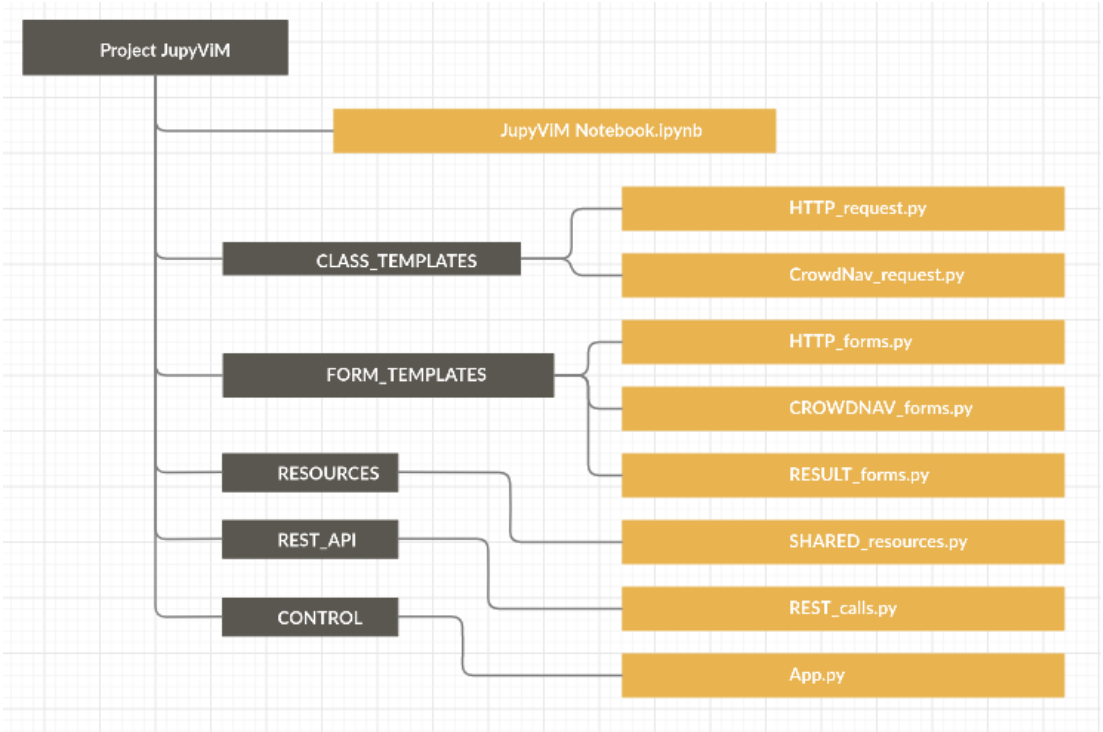


Figure 4.9: JupyViM Project Structure

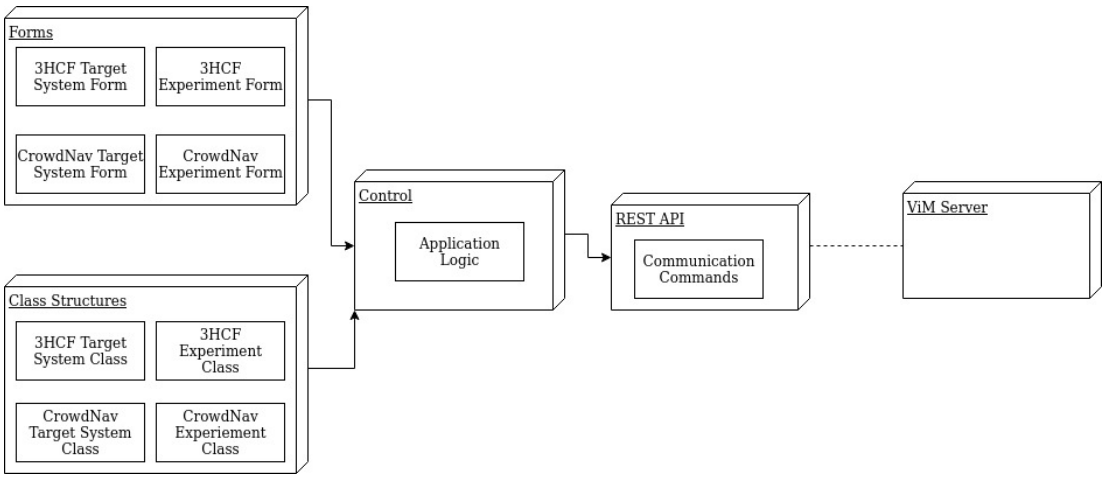


Figure 4.10: JupyViM Deployment Diagram

4.3.3 Features

The features implemented in JupyViM are as follows:

1. **Automatic credential handling**: Since the ViM platform works on a token-session approach, it was imperative to bypass the entire step in order to not hinder the analyst from carrying out their experiments. The way this works is by checking if a default user (pre-defined within JupyViM) is already in the system bypassing their credentials to the ViM server. If the server does not return a token, that would mean the default user is not registered with ViM. Therefore, the default user would have to be registered and then the log-in process would have to be repeated. If, however, a token is returned the first time around, then the JupyViM notebook loads up the application. This functionality is neatly wrapped up in a single function call that runs every time a new instance of JupyViM is instantiated

2. **Target System Creation**: If a user selects the option to create a target system, they are presented with a drop-down list of pre-existing target system templates. Selecting a template, loads its respective target system form. The user would then need to fill out the information as they see fit. Once they submit the form using the "Send Target System" button, all inserted information is mapped to class structures relevant to the target system type that they selected from the drop-down initially. This information is then mapped into a JSON packet that is sent over to the ViM server. If all configuration specifications are met, the target system is created.

3. **Experiment Creation**: The experiment tab initially only shows a "Refresh Target Systems" button and an empty drop-down list. Once the user presses the button, a REST call will fetch all available target systems. Upon selecting any target system, the user will find an experiment form associated with that target system appear. The user would then need to fill out the missing information and choose to either edit the information already present or leave it as is. When the user presses the "Send Experiment" button, the information in the forms is pulled from their fields and used to create an experiment object. The experiment object is then cast into a JSON packet and sent to the ViM server. If all configuration specifications are met, the experiment is created.

4. **Retrieve Experiment Results**: The results tab contains a "Refresh Experiments" button which, once pressed, will populate the experiment drop-down list. The user will then have to select an experiment and its results will be displayed right underneath the cell.

   For features 2, 3 and 4, the user may choose to keep on carrying the function out without refreshing anything. They may also choose to momentarily perform another function and then come back to the current operation. Either way it will work as expected because of asynchronous calls to the server.

In this chapter, we have seen, in detail, the architecture of the devised solution (JupyViM), based on the functional and non-functional requirements of the project. We have seen how the project starts from its basic building blocks, like forms and classes, and proceeds to obtain user input to assemble a payload packet for the ViM server. We have also discussed how the communication, between JupyViM and the ViM server, takes place and how JupyViM eventually receives experiment results and plots them in the notebook application. In the next chapter, we shall go over the evaluation process of the use-cases of JupyViM along with a benchmarking of the communication delays between JupyViM and the ViM server to assess JupyViM's performance.

5 Evaluation

This chapter shall focus on the operation and evaluation of the JupyViM project through its use-cases. Figure 5.1 shows the use-case diagram for the project. The user can create target systems, set up experiments and retrieve results for experiments that have successfully finished running.

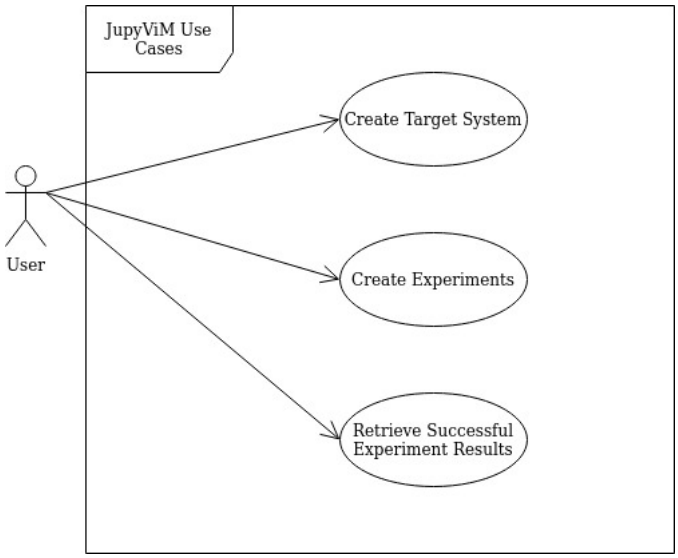


Figure 5.1: JupyViM Use-Case Diagram

5.1 Features

Each use-case is implemented as a standalone feature in JupyViM with no interdependency among them.

5.1.1 Creating Target Systems

The creation of a target system is basically the definition of a black-box simulation environment for ViM to recognize and communicate with. The "Target System" tab (Fig 5.2) in JupyVim, initially, contains a drop-down menu (Fig 5.3) with the two supported simulation frameworks, the Three-Hump Camel Function (Black-Box Function —see Three-Hump Camel Function in the Appendix) and CrowdNav. We will, however, only discuss CrowdNav as it is the main simulation environment under investigation and is a lot more interesting to observe.

Figure 5.2: JupyViM Target System Tab

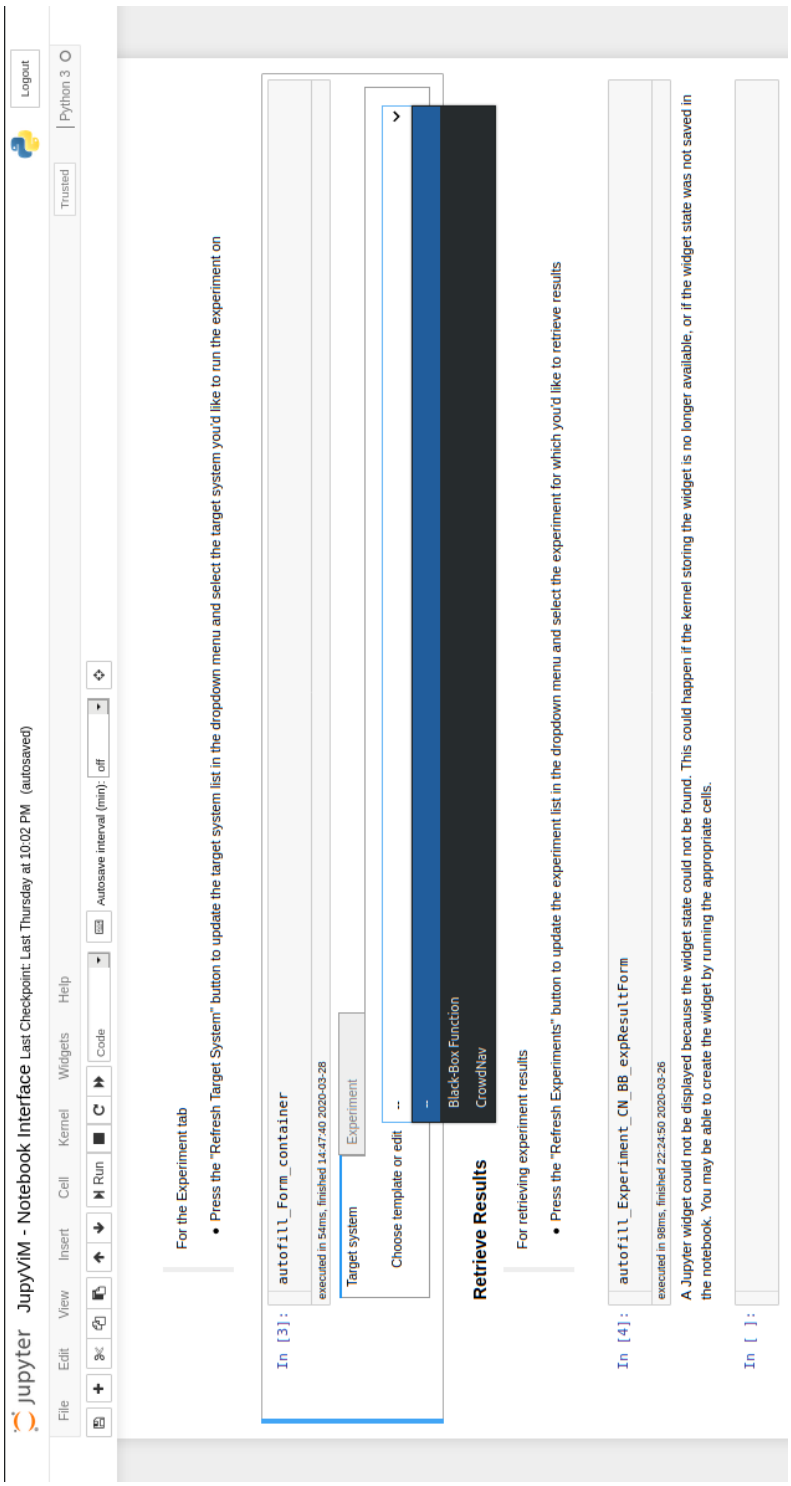


Figure 5.3: JupyViM Target System Drop-Down Options

To see the target system forms, the user needs to select either of the supported simulation environments which will display the respective target system definition forms associated with that option. If the user selects CrowdNav, they will see a form shown by Figures 5.4 and 5.5 and will need to go through the following steps:

- Once the form appears the user will have to give the target system a name in the "cntargetName" field. This will help to uniquely identify the target system when using it later on. The user may also choose to give a description in the "crowdnavtargetDESCR" field.
- Most other text boxes should ideally be left to their default settings. However, the most important text-box after the target system name is the "Kafka URI" field. This needs to be changed based on what the default Kafka URI of the system, JupyViM is running on, is. For example in Fig 5.4, this field is defined as 'localhost:9092', and that is because Kafka is locally installed on the machine that JupyViM is running on. If the user is using a Docker image for Kafka, this could possibly well be 'kafka:9092' or any other string that it was configured to run on.
- The next part that a user needs to define is the data provider section. The user may choose a single data provider by checking the check-boxes next to the names of the data providers (e.g. in Fig 5.4 only the "Trips" data provider is selected) or they may choose a combination or all of them.
- After checking the required data provider check-boxes, the user will need to press the orange "Generate Primary Data Providers" button. Directly below the button, the user will then see the options that they selected from the previous step appear in a selection-box. They would then need to click on one of the data providers in order to declare that provider as the primary data provider. In Figure 5.4, "trips" is the only data provider and is selected as the primary data provider by the user.
- The user then needs to finally select the variables that their target system would contain. These variables affect how the simulation runs. Figure 5.5 shows that only the "Route Randomization" and "Exploration Percentage" variables are selected for the target system being defined.
- Once pressed, the "Send Target System Configuration" button will send the JSON packet containing the target system details to create a new target system, through a POST call in the REST API.

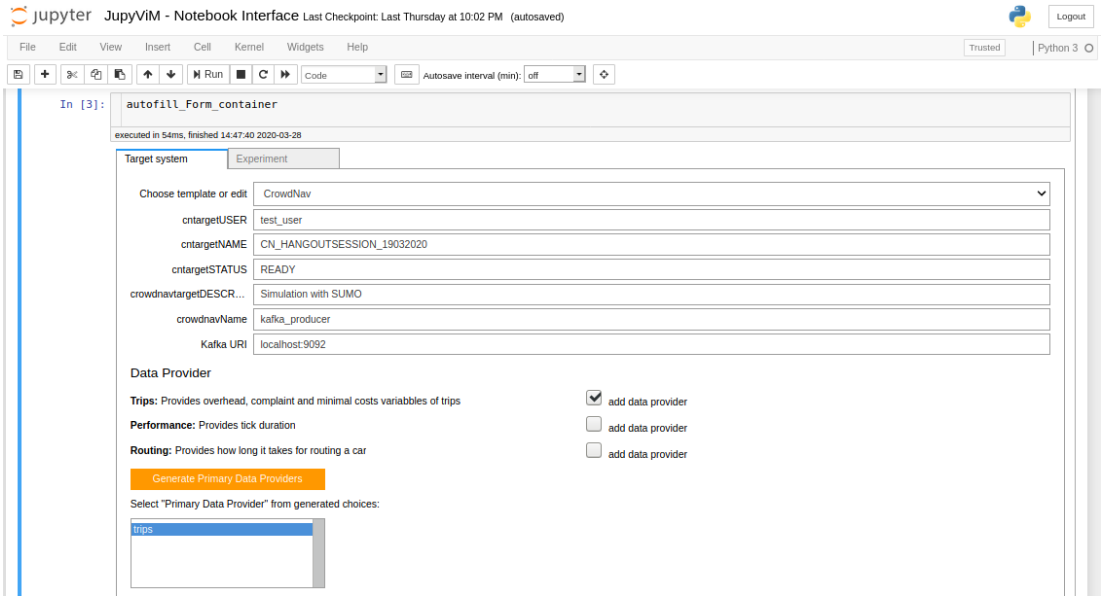


Figure 5.4: JupyViM CrowdNav Target System Creation-1

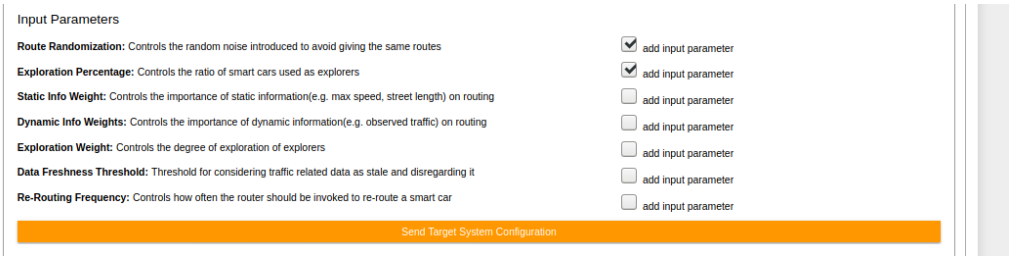


Figure 5.5: JupyViM CrowdNav Target System Creation-2

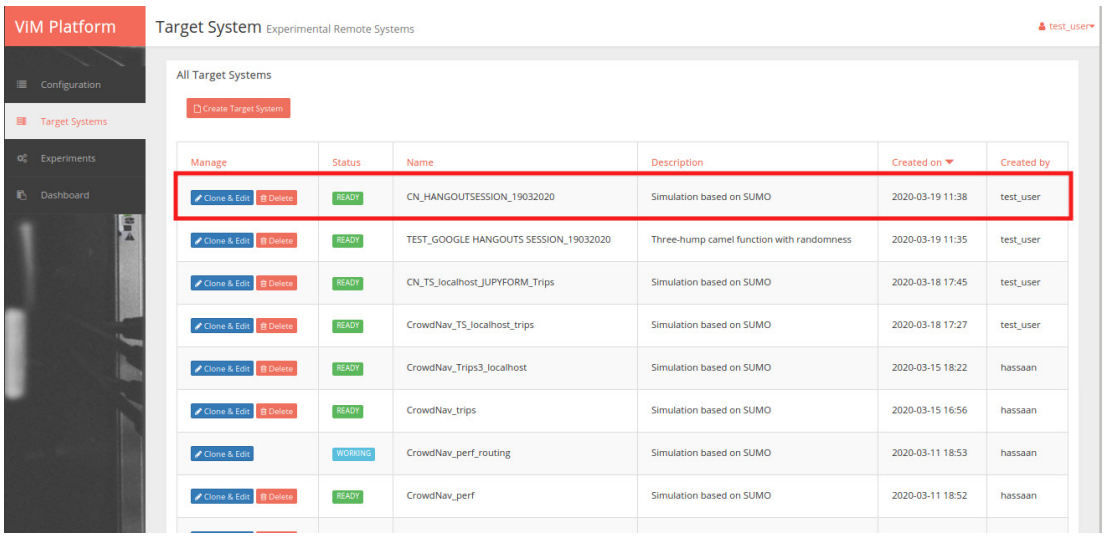


Figure 5.6: ViM Showing Newly Created Target Systems

Figure 5.6 shows the new target system creation verified by ViM.

5.1.2 Creating Experiments

The "Experiment" tab shows that the form initially only contains a "Refresh Target System" button and a drop-down menu which is empty, as can be seen in Figure 5.7. In order to populate the drop-down, the user needs to press the "Refresh Target System" button. Doing so results in a populated target system list (see Fig 5.8).

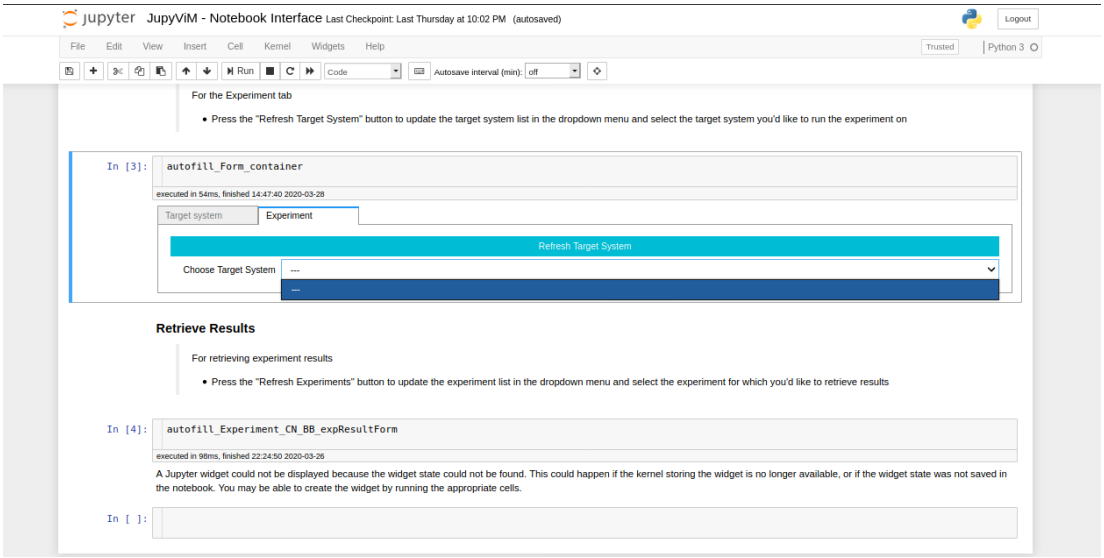


Figure 5.7: JupyViM Experiment Tab

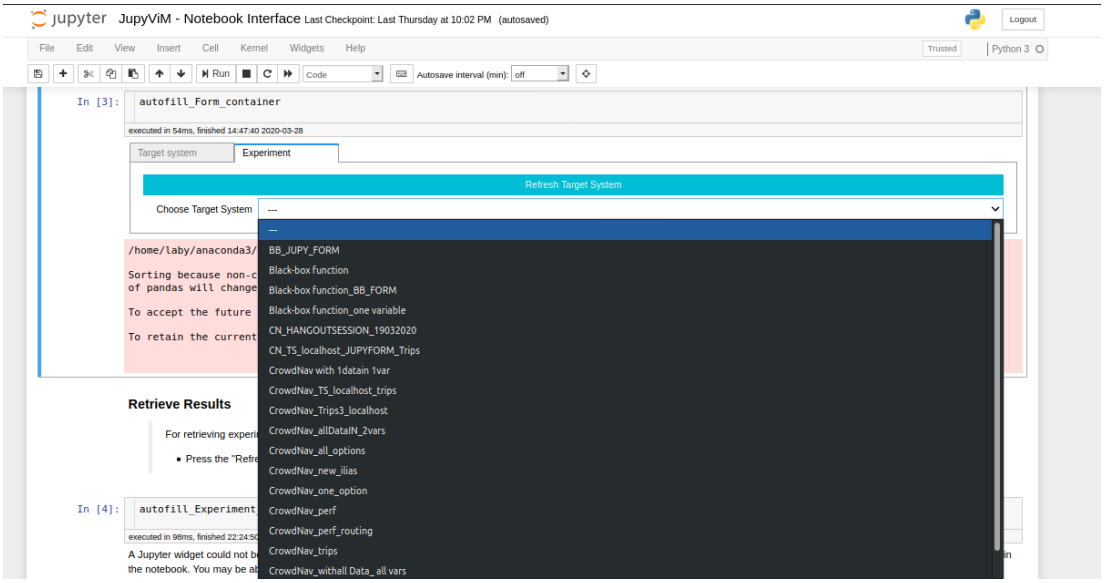


Figure 5.8: JupyViM Experiment Populated Target System List

The user would then need to select a target system from the drop-down list. If the target system that the user selects is based on the CrowdNav simulation environment,

the user will see an experiment form structure similar to Figure 5.9. After the form appears, the user will need to follow the following steps:

- The user will first need to assign the experiment a name in the "Experiment Name" field. This will help to identify the experiment later on. The user can then give a description of the experiment if they so wish, but this is not obligatory.
- The user can then select the type of test they wish to run. The current test options supported are Factorial ANOVA and T-test. Figure 5.9 shows the Factorial test option selected by the user
- Afterwards, the user needs to select an aggregation function that will be performed on the primary data provider. The options include maximum, minimum, average, sum, count and some percentile values. In case the primary data provider is "Trips", there is a further selection that needs to be made on the three attributes of this data provider. Figure 5.9 shows the aggregate function set to 'average' and the 'overhead' metric selected among the three trip metrics, since the target system selected for this experiment had its primary data provider set to "Trips".
- Next, the user needs to provide testing values for the variables displayed for the experiment. The variable options displayed might change depending on the variables that the target system supports, which in this case are "Exploration Percentage" and "Route Randomization" as can be seen in Figure 5.9
- The user then needs to set the sample size for the experiment in the "Iterations" field and the ANOVA threshold in the "Threshold" field. Default options for both these parameters are used if the user does not change them.
- Once pressed, the "Send Experiment" button will send the JSON packet containing the experiment configuration to create and start a new experiment, using a POST request from the REST API.

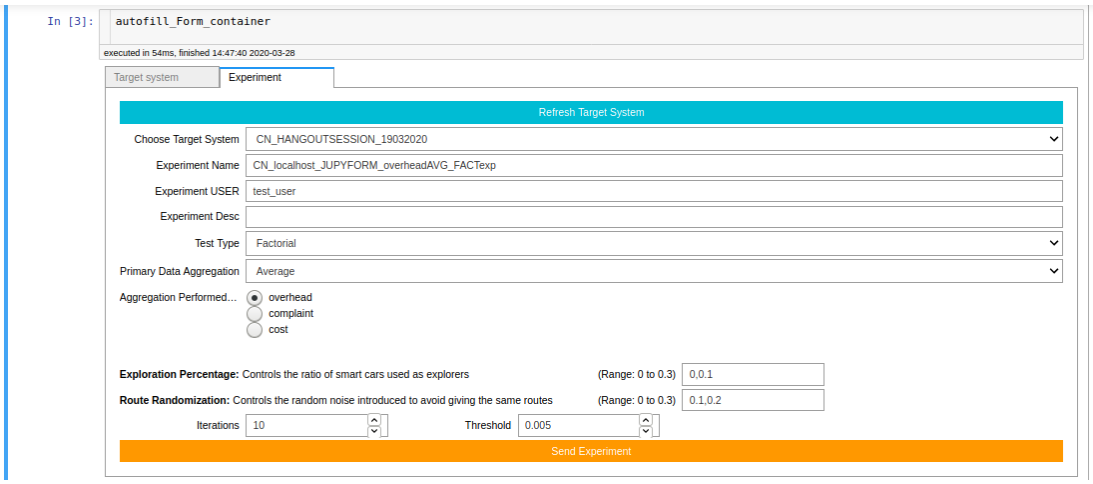


Figure 5.9: JupyViM CrowdNav Experiment Form

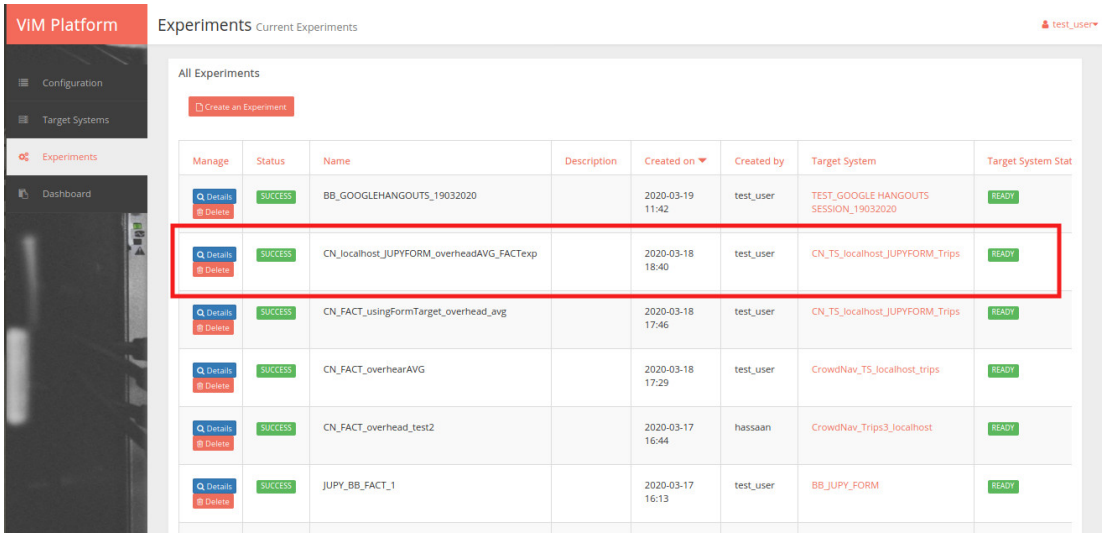


Figure 5.10: ViM Showing Newly Created Experiments

Figure 5.10 shows the new target system creation verified by ViM.

5.1.3 Retrieving Results

Results can be retrieved for experiments that have successfully finished running. When the user first goes over to the results cell, they will see a form similar to Figure 5.11.

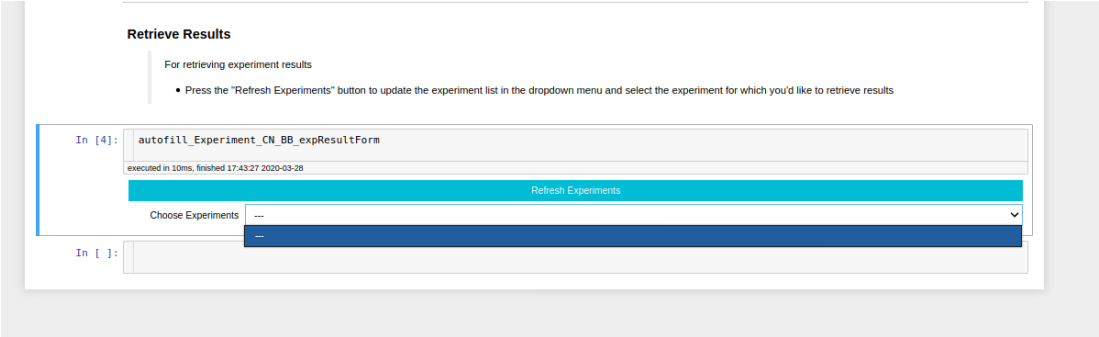


Figure 5.11: JupyViM Results Cell

The user will need to click the "Refresh Experiments" button to fetch the experiment list. Once they do so, the drop-down menu will be populated with experiments that successfully finished. This can be seen in Figure 5.12.

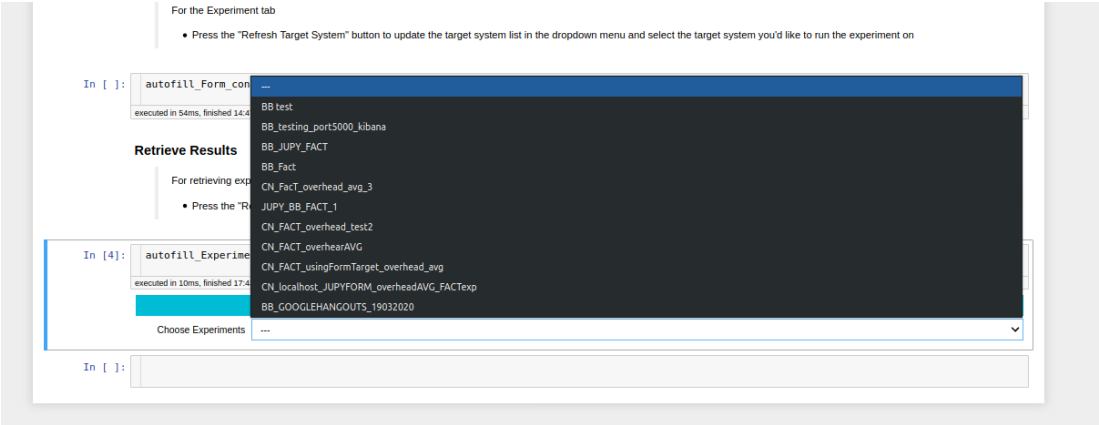


Figure 5.12: JupyViM Results —Drop-down Showing Updated Experiment List

Selecting any of the experiments from the drop-down will fetch the results of that particular experiment and display them immediately, including plotting of the graphs and figures. Figures 5.13 to 5.17 show the outputs for the experiment we created in the "Create Experiments" sub-section of this chapter.

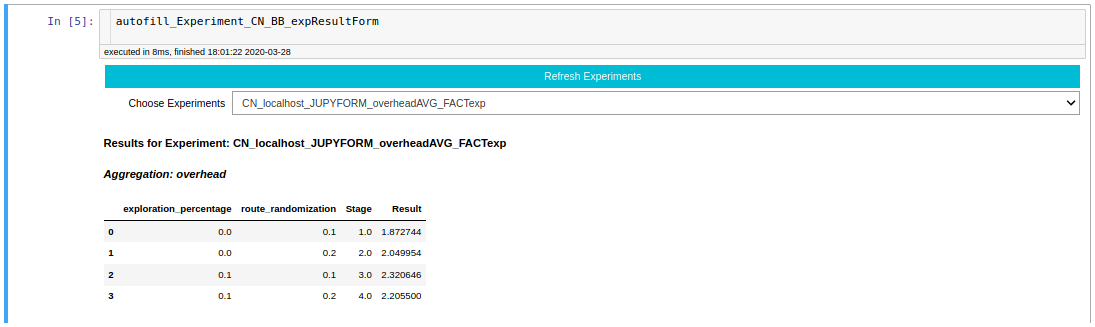


Figure 5.13: JupyViM CN Experiment Results —Stage Table

The scatter plot in Figure 5.14 shows the time-series of the "overhead" metric against the time-stamp that it occurred on along with the 95^{th} percentile threshold line drawn over the plot.

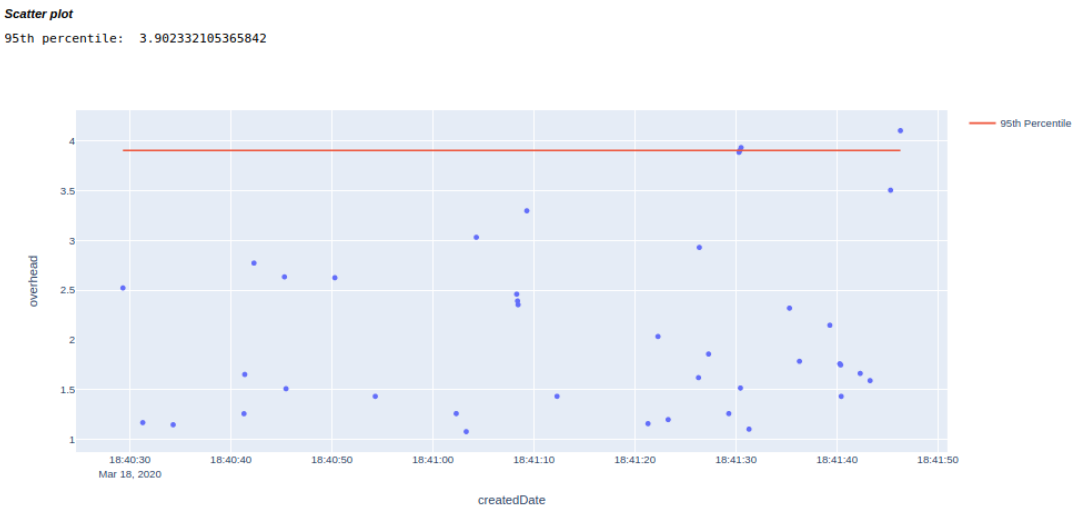


Figure 5.14: JupyViM CN Experiment Results —Scatter Plot

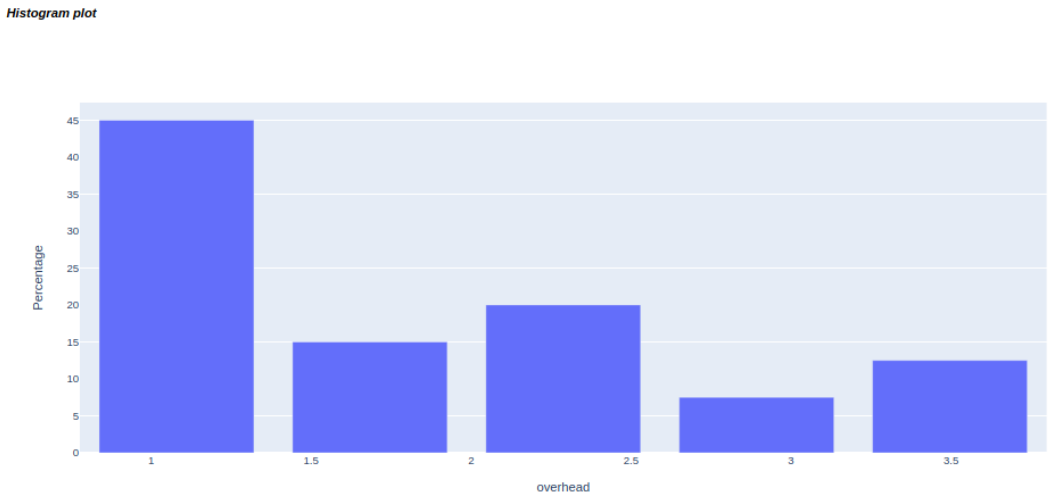


Figure 5.15: JupyViM CN Experiment Results —Histogram

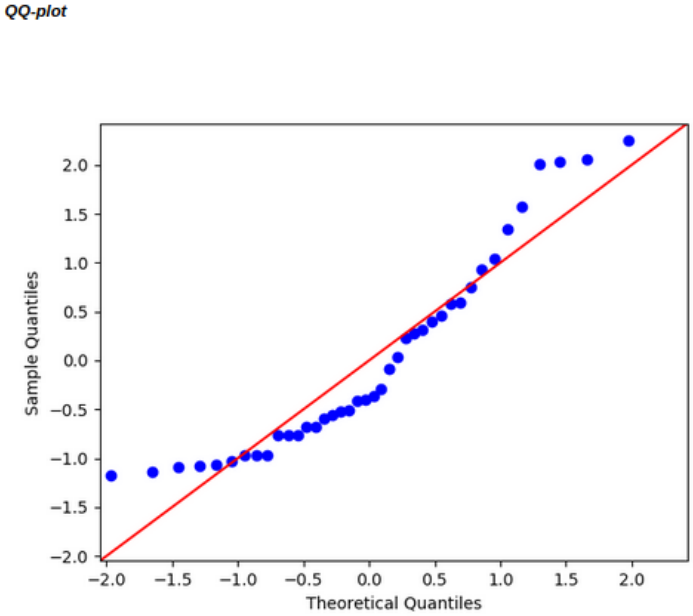


Figure 5.16: JupyViM CN Experiment Results —QQ-Plot

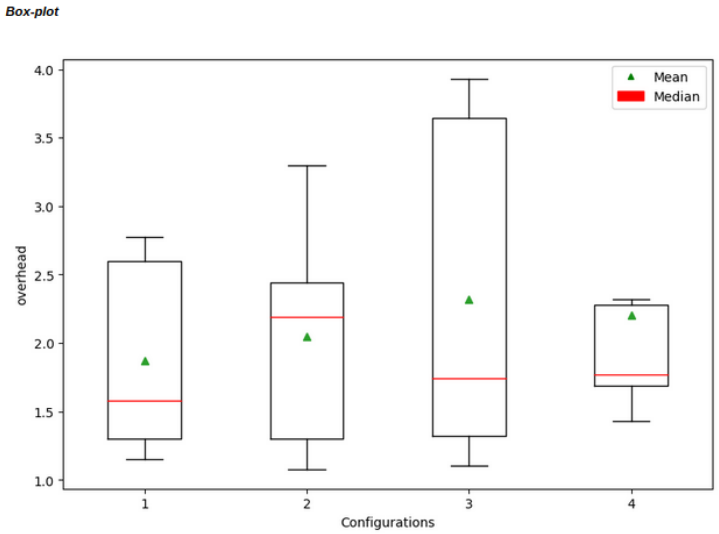


Figure 5.17: JupyViM CN Experiment Results —Box-Plot

5.1.4 Run-time Analysis

In order to assess the responsiveness of JupyViM and its communication with the ViM server, the use-cases of JupyViM were timed and charted. The purpose of these timed experiments was to find out if the sample size or the number of variables affected the time it took for JupyViM to send out a request to the ViM server and receive a response. All tests were carried out on the CrowdNav simulation platform and the primary data provider was set to "Trips" in all test cases to act as a constant.

- **Target System Creation**: CrowdNav supports a total of 7 variables that influence how the simulation is carried out. For this test, the number of variables defined within the target system configuration was varied and timed to find out if they caused any delays in the communication between JupyViM and the ViM server. Table 5.1 and Figure 5.18 show the results. From the line chart, it is quite evident that there is a slight anomaly at the 2 variable target system creation point which takes about **50%** more time than all the other target system combination's average response time. All other target systems fall in the range of about five-tenths of a second. This means the communication delay between JupyViM and the ViM server is not at all significant.

Table 5.1: Timed Target System Creation

No. of CrowdNav variables	Round trip request time (seconds)
1	0.304778
2	0.855585
3	0.296169
4	0.209463
5	0.35548
6	0.162918
7	0.244335

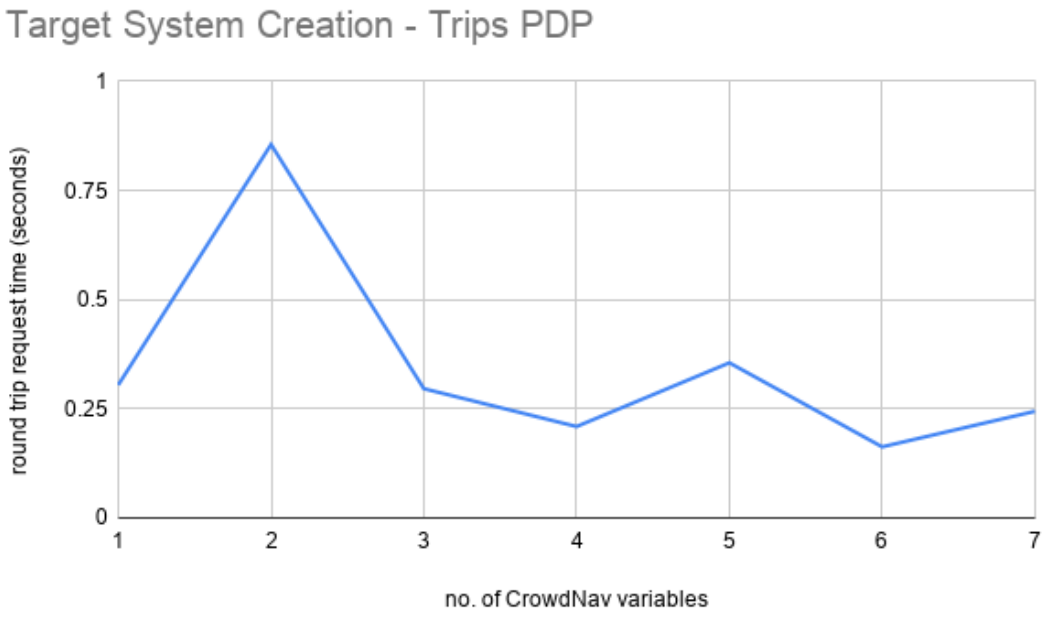


Figure 5.18: Timed Target System Creation —Line Chart

- **Experiment Creation**: Experiment creation has been dealt with in two approaches. The first approach is where the sample size of the experiment variables was kept constant at 10 but the number of variables was varied. The second approach is where the number of variables was kept constant at 2 and the sample size was varied. It is also important to note that in all iterations of both the approaches, the

test used was Factorial ANOVA and all tests were run on target systems defined in the previous (Target System Creation) step.

– *Constant Sample Size and Varied Number of Variables*: The first iteration starts with 2 variables instead of 1 because Factorial ANOVA needs at least 2 variables to run. Figure 5.19 shows that the first point, experiment definition with 2 variables, is anomalous to the trend. All other experiments fall within the range of a tenth of a second of each other, which shows that varying the number of variables used in the experiment definition does not necessarily have any effect on the response time between JupyViM and the ViM server.

Table 5.2: Experiment Creation Fixed Sample Size

Experiment No.	No. of Varying variables	Round trip request time (seconds)
1	2	0.343014
2	3	0.13274
3	4	0.174219
4	5	0.156394
5	6	0.155563
6	7	0.231348

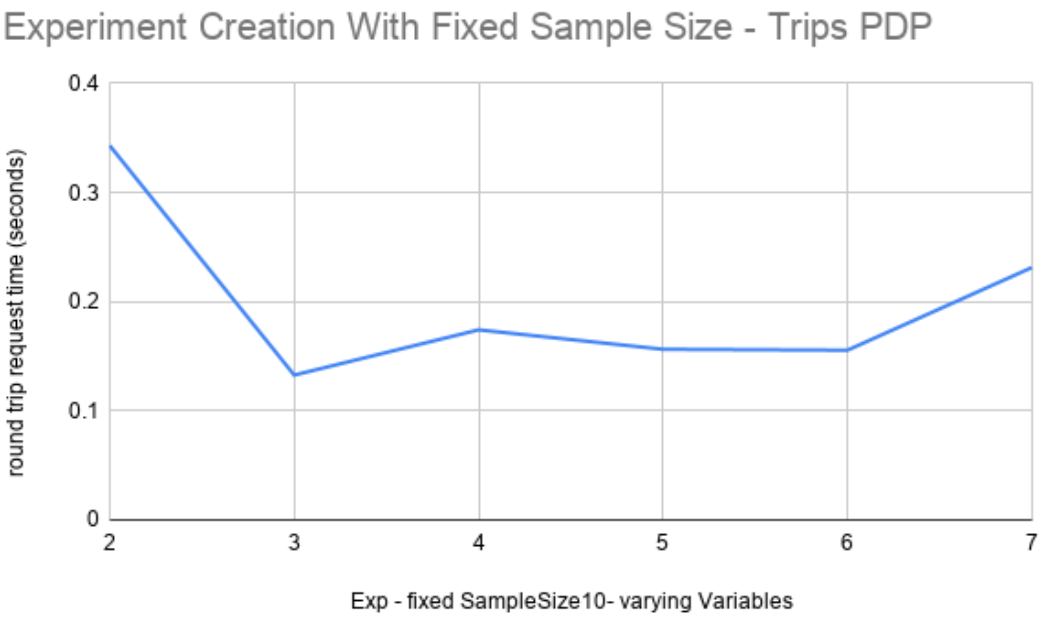


Figure 5.19: Experiment Creation Fixed Sample Size —Line Chart

– *Constant Variable Size and Varied Sample Size*: Figure 5.20 shows that all varied sample sizes have response times between JupyViM and the ViM server that fall within the range of roughly two-hundredths of a second. This means varying the sample size number in the definition of an experiment has nearly no lagging effect on JupyViM, making its communication with the ViM server quite instantaneous.

Table 5.3: Experiment Creation Fixed Number of Variables

Experiment No.	Sample size	Round trip request time (seconds)
1	10	0.07733
2	20	0.066738
3	30	0.064242
4	40	0.075621
5	50	0.057078

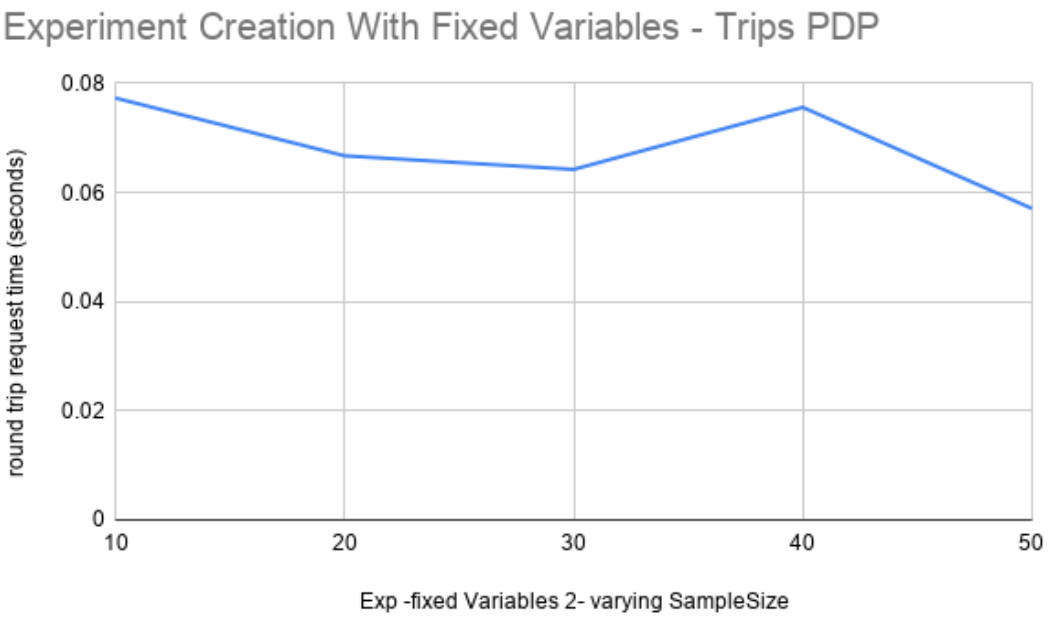


Figure 5.20: Experiment Creation Fixed Number of Variables —Line Chart

All experiments were carried out on a Core-i5 machine clocked at 2.30 GHz, with 8GB of RAM, running Ubuntu 18.04.3 LTS. It is also important to note that the sample sizes were not extremely high because our aim is to measure the communication between JupyViM and the ViM server, not to measure the performance of the ViM server. Also the testing hardware is a fairly average machine and CrowdNav and the ViM server are extremely resource-intensive so to serve our purpose of analysis, the figures provided prove to be sufficient.

- **Result Retrieval**: The current prototype of JupyViM only supports retrieving results for experiments that have successfully finished running. For this timing experiment, the results for the experiments created in the "Constant Variable Size and Varied Sample Size" section are fetched. Figure 5.21 shows that all the results were returned by the ViM server within, the range of, roughly a hundredth of a second. This means that no matter how large a successful experiment is, the results are served to JupyViM almost instantaneously after querying for them from the ViM server.

Table 5.4: Result Retrieval Fixed Number of Variables

Experiment No.	Sample size	Round trip request time (seconds)
1	10	0.02254
2	20	0.02648
3	30	0.02628
4	40	0.024727
5	50	0.032485

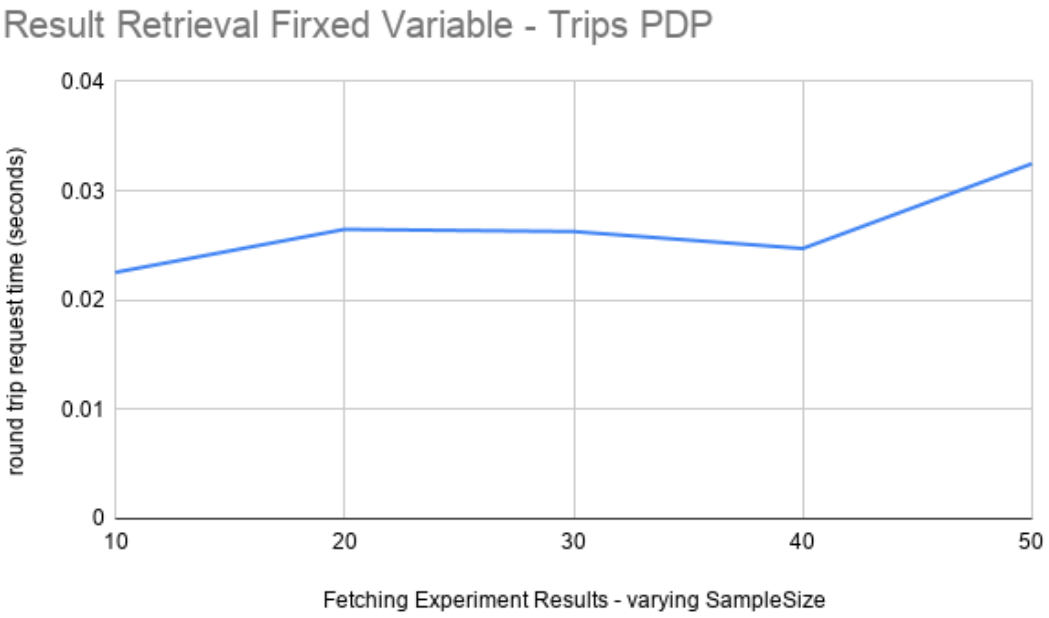


Figure 5.21: Result Retrieval Fixed Number of Variables —Line Chart

The analysis, of the results for the tests carried out to measure the idle time between JupyViM's request and the ViM server's response, shows that their communication is fairly instantaneous. Neither the number of configuration parameters nor the size of the result payload cause any significant transmission delays between the two communicating entities. This makes JupyViM a suitable alternative to the ViM client.

6 Conclusion

Throughout this thesis, we have investigated the idea of an augmentation to the ViM experimentation platform's client application. To that end, we have designed and implemented JupyViM. We have seen ViM's client being limiting in terms of providing users with a familiar environment and we have seen JupyViM trying to address that specific issue by leveraging technologies like Python and Jupyter Notebooks to provide a more accessible environment to data scientists. Through this effort, JupyViM aims to make the workings of the ViM client more open and approachable by its users in order to make the process of experimentation and analysis smoother and more productive.

Contributions by JupyViM

- JupyViM is designed to carry out the core functions that the ViM client offers. However, owing to demands of flexibility, JupyViM does so in an environment that is familiar to data scientists; in a language that is also familiar to them, creating avenues that were up until this point, locked.
- JupyViM tries to bridge the gap between fast prototyping and a decent use-able tool. While accomplishing this task, it makes multi-tasking just as open as the ViM client by keeping, the definition part of target systems, the creation of experiments and the retrieval of results all de-coupled, to gain the benefit of the asynchronous server that ViM provides.
- JupyViM also keeps the option of customization and future support open by defining a structure of form creation and class support in its architecture that makes all elements of the notebook project independent. This is possible because JupyViM is independent of the ViM client and does not use any of its services when communicating with the ViM server.

Being modular, fast and intuitive, JupyViM focuses on the functionalities of ViM, intending to improve the process that would eventually increase the productivity of the analysts using the tool.

As is the case with every tool, JupyViM is also subject to some limitations.

1. In it is current state, JupyViM only has implemented a single default user for logging into the ViM platform to bypass the security feature of ViM during its user-token sessions. Although this feature is added by design to not spend time in logging into the ViM platform, if future iterations of ViM were to limit and restrict user access (as is not the case for now), then JupyViM would only be able to display experiment results for experiments that this default user creates, instead of all of them. This of course is easily fixable by adding user login support but for now it remains a limiting factor in the trade-off between quick access and security.

2. Jupyter Notebooks display the result of code executed in a cell, underneath that cell. The forms and experiment results displayed to the user, are basically the outputs of code that is executed to invoke them. The ViM platform has a feature that allows it to show experimental results for experiments that are still running. The results are updated periodically. In its current iteration, JupyViM does not support the displaying of results for experiments that are still running. This is because continuous updating of incoming data points requires the current output to be cleared and then re-drawn upon. This essentially also wipes out the form structure, which means that to display live experiment results, the user would only be able to view one experiment's results. In a design-functionality compromise, this feature was omitted for the current prototype.

3. Another important point to note is that JupyViM is only an alternative to the ViM client, not the ViM server. The functionality that JupyViM provides is dependent on the features that the ViM server supports. Adding support for newer simulation environments on JupyViM will not enable them unless the ViM server also supports them.

6.1 Future Possibilities

To improve the current JupyViM prototype, one should first look at improving its limitations. The default user limitation is probably not a very urgent change because the ViM server still does not partition user spaces. The matter of showing live experiment results, however, could be prioritized. One possible solution to this problem could be to separate the displaying of results into two separate modules, one for presenting all successful experiments and the other for showing live experiments. These would require separate cell executions in the notebook. The live experimentation module could be tied to the feature that starts a new experiment within JupyViM and then streams the results of the experiment or continuously updates the results by drawing

over the live experimentation cell.

Since JupyViM is written entirely in Python, future works could include support for adding newer simulation platforms. Creating query forms and class structures, modeled after existing ones, for target systems and experiments, would achieve this. However, for JupyViM to support the addition of new simulation systems, the ViM server would also need to be updated.

In its process of retrieving and handling results, JupyViM un-packs the JSON payloads that contain result data and creates robust data frames out of them. These data frames are used to create visuals for plotting graphs for the user. Depending on future support of newer simulation platforms, another possible direction for carrying JupyViM forward could be to utilize these data frames for creating newer, more meaningful visuals.

7 Appendix

7.1 Jupyter Notebook

Jupyter Notebook [11] is a web-based interactive application based on IPython (command shell for programming languages - primarily python) [24] and supports parallel computing by spinning a new task every time a new running notebook is created. Jupyter follows the concept of compartmentalizing a source code file into smaller manageable parts (called cells) that can be run independently and are stored in memory. The output for each independent cell is tied to the code block, so running the code displays the output right underneath it. It also provides the ability to share runnable notebook files from one machine to another (irrespective of the operating system), provided that the destination machine also has an interpreter (kernel) for the language that the notebook was written in e.g a python notebook would require a python interpreter installed. In addition, one can use Markdown to note down important static information (like running instructions, questions, future ideas, etc.) to make the notebook a nice all-in-one project prototyping and documentation tool

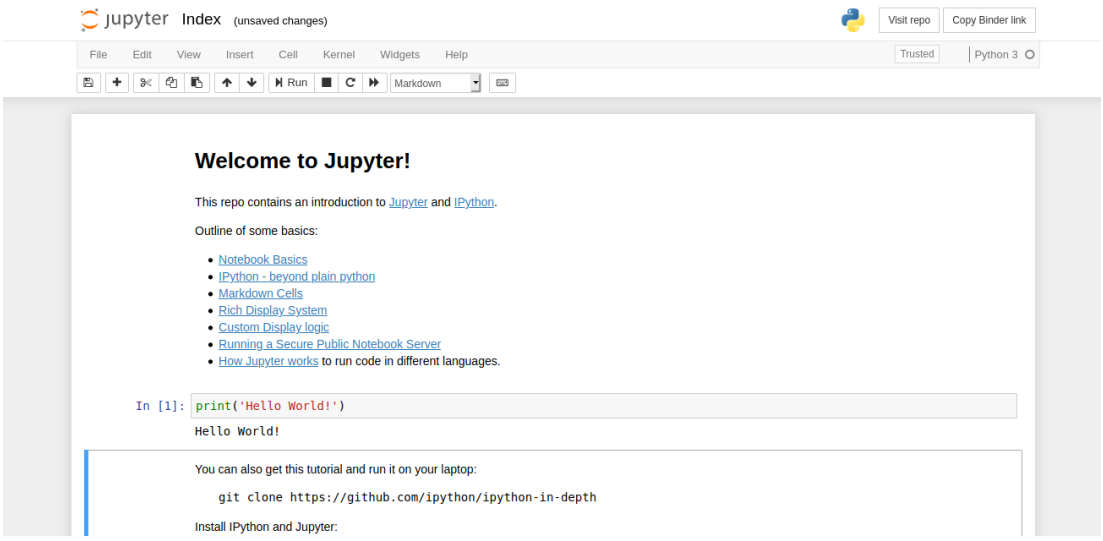


Figure 7.1: Sample Jupyter Notebook (Figure from [11])

7.2 Three-Hump Camel Function

A common method of testing optimization algorithms in applied mathematics, is through the use of test-functions. One such function is the bi-variate, non-convex Three-Hump Camel Function (3HCF [20]):

$$f(x_1, x_2) = 2x_1^2 - 1.05x_1^4 + \frac{x_1^6}{6} + x_1x_2 + x_2^2$$

Being continuous, the function can take any domain as input. However, the recommended domain is: $x_1, x_2 \in [-5, 5]$ The global minimum for this function $f(x_1, x_2) = 0$ exists at $(x_1, x_2) = (0, 0)$.

In ViM, the Three-Hump Camel Function acts as a simulation environment that is used to test the installation of the ViM project. This function is incorporated in a separate server, but it is bundled within the ViM project for easy testing.

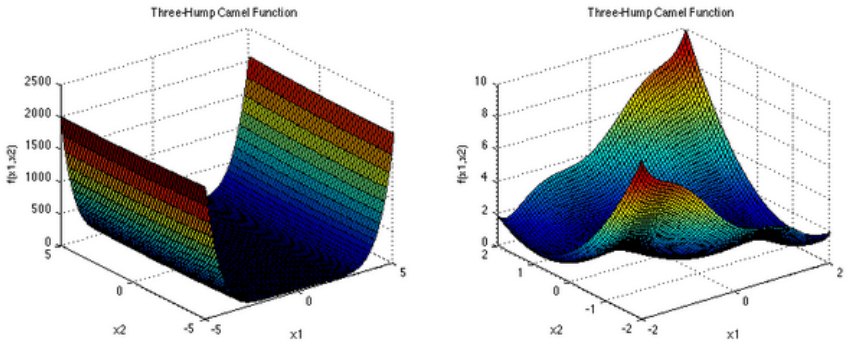


Figure 7.2: Three-Hump Camel Function (3HCF) (Figure from [20])

List of Figures

List of Tables

Bibliography

[1] Apache. https://kafka.apache.org/intro. Accessed on 2020-03-20.

[2] Z. D. BAYERN. *Virtual Mobility World (ViM)*. https://vim-project.org/project/. Accessed on 2020-03-20.

[3] C. B. 4. Ch.ko123 - Own work. https://commons.wikimedia.org/w/index.php?curid=59871096. Accessed on 2020-03-20.

[4] N. Diamantopoulos, J. Wong, D. I. Mattos, I. Gerostathopoulos, M. Wardrop, T. Mao, and C. McFarland. "Engineering for a Science-Centric Experimentation Platform." In: *arXiv preprint arXiv:1910.03878* (2019).

[5] T. Economist. *The world's most valuable resource is no longer oil, but data*. https://www.economist.com/leaders/2017/05/06/the-worlds-most-valuable-resource-is-no-longer-oil-but-data. Accessed on 2020-03-20. May 2017.

[6] Elastic. https://www.elastic.co/what-is/elasticsearch. Accessed on 2020-03-20.

[7] A. Fabijan, P. Dmitriev, H. H. Olsson, and J. Bosch. "The evolution of continuous experimentation in software product development: from data to a data-driven organization at scale." In: *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*. IEEE. 2017, pp. 770–780.

[8] D. Fisher, B. Chandramouli, R. DeLine, J. Goldstein, A. Aron, M. Barnett, J. C. Platt, J. F. Terwilliger, and J. Wernsing. "Tempe: an interactive data science environment for exploration of temporal and streaming data." In: *Tech. Rep. MSR-TR-2014–148* (2014).

[9] I. Gerostathopoulos. https://github.com/iliasger/OEDA. Accessed on 2020-03-20.

[10] I. Gerostathopoulos, A. N. Uysal, C. Prehofer, and T. Bures. "A tool for online experiment-driven adaptation." In: *2018 IEEE 3rd International Workshops on Foundations and Applications of Self* Systems (FAS* W)*. IEEE. 2018, pp. 100–105.

[11] Jupyter. https://jupyter.org/. Accessed on 2020-03-20.

[12] D. Krajzewicz, G. Hertkorn, C. Rössel, and P. Wagner. "SUMO (Simulation of Urban MObility)-an open-source traffic simulation." In: *Proceedings of the 4th middle East Symposium on Simulation and Modelling (MESM20002)*. 2002, pp. 183–187.

[13] J. Kreps, N. Narkhede, J. Rao, et al. "Kafka: A distributed messaging system for log processing." In: *Proceedings of the NetDB*. Vol. 11. 2011, pp. 1–7.

[14] G. A. C. .-.-. D. Z. für Luft- und Raumfahrt e.V. `https://www.dlr.de/ts/en/desktopdefault.aspx/tabid-9883/16931_read-41000/`. Accessed on 2020-03-20.

[15] G. A. C. .-.-. D. Z. für Luft- und Raumfahrt e.V. `https://sumo.dlr.de/docs/Simulation/Routing.html`. Accessed on 2020-03-20.

[16] G. A. C. .-.-. D. Z. für Luft- und Raumfahrt e.V. `https://sumo.dlr.de/docs/Screenshots.html`. Accessed on 2020-03-20.

[17] G. A. C. .-.-. D. Z. für Luft- und Raumfahrt e.V. `https://sumo.dlr.de/docs/TraCI.html`. Accessed on 2020-03-20.

[18] MICROSOFT. `https://www.typescriptlang.org/`. Accessed on 2020-03-20.

[19] N. TechBlog. `https://netflixtechblog.com/notebook-innovation-591ee3221233`. Accessed on 2020-03-20.

[20] S. F. University. `https://www.sfu.ca/~ssurjano/camel3.html`. Accessed on 2020-03-20.

[21] A. Uysal. `https://github.com/alinaciuysal/OEDA`. Accessed on 2020-03-20.

[22] Wikipedia. `https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm`. Accessed on 2020-03-20.

[23] Wikipedia. `https://en.wikipedia.org/wiki/A*_search_algorithm`. Accessed on 2020-03-20.

[24] Wikipedia. `https://en.wikipedia.org/wiki/IPython`. Accessed on 2020-03-20.

[25] G. Wilson. "Software carpentry: getting scientists to write better code by making them more productive." In: *Computing in Science & Engineering* 8.6 (2006), pp. 66–69.