System Simulation Using Python Techniques

Sonali B. Gagare[1] Prof. Monika D. Rokade[2]
[1,2]Department of Computer Engineering
[1,2]Sharadchandra Pawar College of Engineering, otur, Maharashtra, India

Abstract— This article describes the development of a simulation engine developed in the Python programming language, with a focus on continuous system simulation. The function of this engine is based on CSMP block-oriented language and software, which can carry out continuous system simulation. The purpose of this article is threefold. First, this article describes the requirements and usage of this engine. The second goal is to analyze the Python concept used to implement the simulation engine-API development, communication with intelligent systems, real-time data processing, differential equation integration, threading and parallel programming. Finally, this article will describe the engine architecture, use cases, and the usage of the concepts analyzed during the development process. Finally, we will discuss the overall results and the process of future engine improvements and upgrades.

Keywords: Continuous System Simulation, Python, Engine, Intelligent System, Internet of Things

I. INTRODUCTION

Continuous system simulation is based on a model that is represented via differential equations and recurrence relations[1][2]; each change, observed in the system, has a corresponding differential equation. It should be borne in mind that digital computers cannot perform computations of continuous variable changes, hence, they should be approximated by a set of discrete values[1][3][4].

The simulation of continuous systems has been extensively researched over time and implemented in various ways. Available software solutions for the continuous system simulation are mostly desktop client-only software (Simulink [5], CSMP/FON [6]) or web-based software with limited application (Insight Maker [7]). Realtime data processing and communication with smart systems are not supported in current solutions.

This article describes the development of an engine for network-based simulation using/using the Python programming language, focusing on the function of continuous system simulation. The simulation engine realizes model union, in which discrete event elements are used to extend the classic model for continuous system simulation. It will be developed based on the principles of distributed and concurrent programming so that it can be integrated with different systems and environments. The engine represents an integral part of the system, which includes educational components, scientific research components, and a framework for real-time management of smart devices.

The primary reason for choosing Python as the programming language was its use in the educational process, as well as being a modern and easy to use programming language.. In addition, being a modular service. Taking this into account, an existing formal model of the simulation engine [1] was extended with IoT component [2].

II. THEORETICAL BACKGROUND

A. *Continuous system simulation:*

Systems whose state variables continuously change over time are called continuous systems [1][3]. An example of this system would be a runner on a track whose position and velocity are continuously changing over time. The given example represents the first of three basic simulation models [8]:

Models described via ordinary differential equations

1) (ODE): motion, physical, chemical, biological and other processes involving a singular unknown function and unknown variable. Models described via systems of partial differential equations
2) (PDE): aerodynamic, hydrodynamic and meteorological processes.
3) System dynamics: a methodology of research, modeling and simulations of complex dynamical systems [9]. The most common problems modeled by system dynamics are feedback models such as social, population and economical systems.

Web-based simulation makes the simulation of systems much easier and accessible; due to the fact it runs on the remote server so it's not limited by a user computer system and its possibilities [10][11]. Furthermore, it allows numerous users to share simulation results or simultaneously access the resulting data, because it is stored on the web server.

There are numerous web-based software solutions for the different types of simulation, but mostly simulation of the discrete events. For instance, one of the well-known solutions Insight Maker [7] allows the continuous system simulation of system dynamics.

Authors [12] described web-based simulation using Java programming language and applets that allows modeling of the ODEs and PDEs. The main issue with this software is that is not fully web-based, only its graphical user interface.

B. *CSMP/FON a reference software model*

Continuous System Modeling Program (CSMP) is a block oriented simulation language specialized for continuous system simulation[1][13].

The development of the continuous system simulation features of the simulation engine described in this paper is based on the CSMP/FON software, developed at the

Department of e-business, Faculty of organizational sciences, University of Belgrade [4]. This software is being used in the teaching process within Computer simulation course. CSMP/FON is a desktop application (Fig 1.) that allows a user to create a block model defined by set of differential equations[1][6]. The software is an open source solution [6] developed in Pascal programming language following principles listed below [13]:

- Minimal required hardware resources, high speed of execution
- Simple and rich interface, easy to use

– Educational support
– Scientific research support

The software is a client-only desktop application [14], without a web interface or an API. The CSMP/FON software does not support integration.

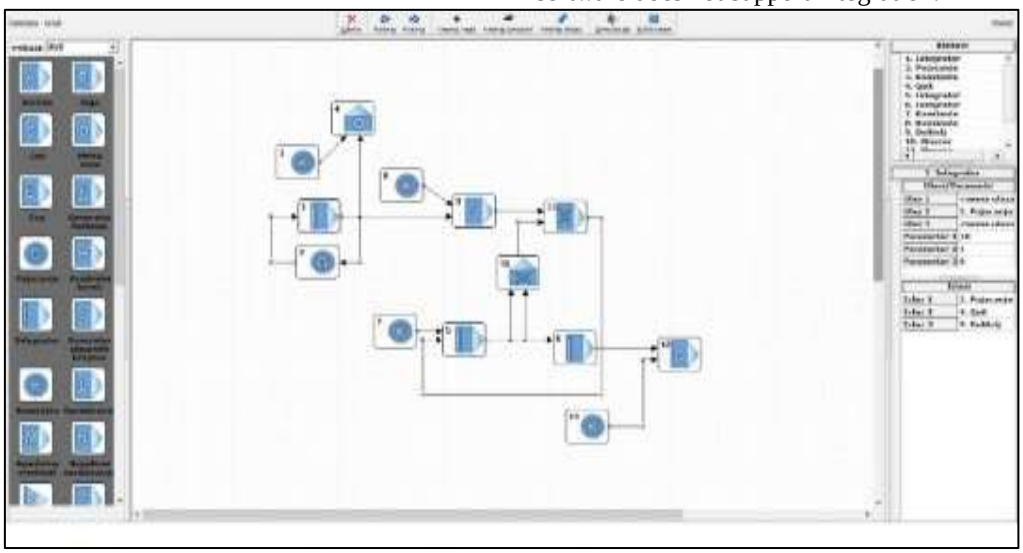
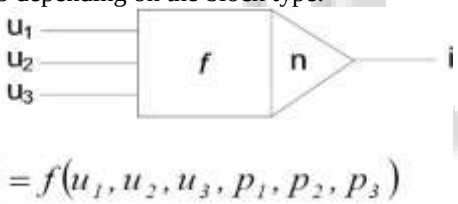


Fig. 1: CSMP/FON Simulation software

Each simulation block specified in the configuration has its own order number, assigned chronologically during model creation, an ID and graphical symbol, generally represented as in the fig.2 [13] that visually depicts a single mathematical operation or a function. The set of input values is represented by u1, u2, u3 and p1, p2, p3 represent a set of parameters. The number of inputs and parameters can vary from 0 to 3 depending on the block type.



$$i = f(u_1, u_2, u_3, p_1, p_2, p_3)$$

Fig. 2: General graphical block representation [1]

The program itself is based on the sorting algorithm of the block order numbers, giving the computation schedule for calculating the block output, where each block input is already provided from the calculations in the previous step [1]. Software uses the numerical integration for the calculation of the output of the Integrator (I) block. Numerical integration is performed by the Runge-Kutta algorithm IV order (RK4)[1][15].

1) Continues System Simulation

Formal model for the hybrid IoT system for real-time simulation formulated in the previous research [2] defined functions for obtaining values of state variables from the IoT system (SIoT):

– ρ : reading the values of variables from IoT system
– ω : writing the values of variables into the IoT system
– γ : reading the outputs of the IoT systems.

The communication of the entire engine is based on signals (messages) and shared memory space, while the communication with intelligent systems and other distributed components is achieved through APIs. The autonomous performance of the continuous system simulation engine is realized using the concepts of distributed and concurrent computing (threads and processes) (Figure 5):

1) Serve the request of simulation process control and error report,
2) Provide services for configuration change requests,
3) Read the data and send the control measures to the IoT system,
4) Serving requests for simulation results,
5) Perform the simulation process

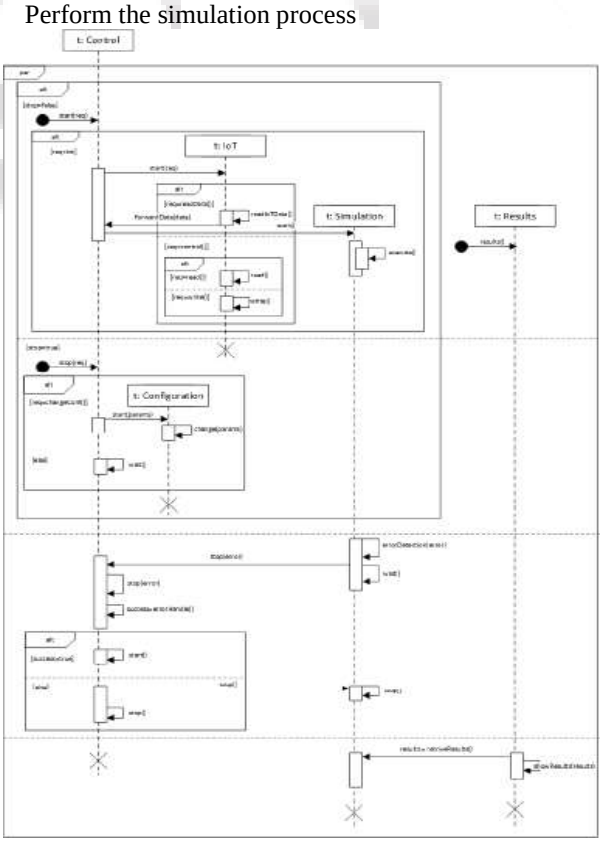


Fig. 3: Thread/process communication – UML Sequence diagram

III. DEVELOPMENT CONCEPTS

In the previous section, the continuous system simulation features and functions of the engine have been introduced.
In summary, the simulation engine should include the following functions:

1) Numerical integration of di *ff*erential equations
2) Simulation process control
3) Real-time data processing
4) Communication with intelligent systems
5) Enable result preview

These functions are implemented using di*ff*erent concepts of the Python programming language:

1) Numerical analysis of the integral of di *ff*erential equations,
2) Parallel programming, used to serve the simulation process, di*ff*erential equation integration and real-time data processing,
3) Develop an API for communicating with intelligent systems.

1) Numerical analysis

Numerical analysis uses numerical approximations for the problems of mathematical analysis, such as di*ff*erentiation and integration [16]. As previously mention, computations of continuous variable changes performed by the simulation engine needed to be approximated by the set of discrete values.

Python programming language does not have an "out of the box" solution for the numerical analysis computations.

During the development of the simulation engine a Python-based ecosystem for mathematics, science and engineering – SciPy[17] was used as areference model for the development of the integration method. SciPy o*ff*ers explicit and implicit Runge-Kutta methods for solving initial value problems for ODEs systems.

2) Concurrent programming

Concurrent programming allows a computer system to run the computation without waiting for currently running computation to complete [18]. Flynn's taxonomy [19][20] classifies computer systems into four categories based on two criteria: number of data streams and number of instruction streams:

1) Single Instruction Single Data
2) Multiple Instruction Single Data
3) Single Instruction Multiple Data
4) Multiple Instruction Multiple Data

Multiple Instruction Multiple Data is the most general class of parallel computer systems [20], and represents the engine classification as well. This is a powerful architecture, but it has its drawbacks: memory access. There two ways to organize memory access in parallel programs [21]:

– Shared memory
– Distributed memory

Shared memory allows all processors to have equal access to data and instructions in memory, wherein distributed memory each processor is allocated with its local memory that is not accessible to other processors [20].

Python programming language implements two ways for achieving parallelism: threads and processes. because of the Pythons Global Interpreter Lock (GIL). GIL disables parallel execution of multiple threads [24], mainly because C Python's (Python interpreter) memory management is not thread-safe – meaning that only one thread can execute the Python code at the same time [20].Several computations are executed, via threads, throughout overlapping time periods [18]..

Either way, an implementation of di*ff*erent lock mechanisms on the shared memory space should be considered. Out of the box, Python provides two locking mechanisms: Lock and RLock [25]. Python Package Index provides di*ff*erent modules and solutions for the Python programming language, such as Reader Writer locking mechanism[26].where "reader" threads may share the section with each other, but the "writers" must have exclusive access (fig.7)

1) Distributed computation
2) Reading data from IoT system /distributed component
3) Display of the results in real-time

```
import threading
from readerwriterlock import rwlock

marker = rwlock.RWLockFair()

def getResults(sim_id):
    global pointer
    read_marker = marker.gen_rlock()
    msg = Message("GET_RESULTS", data=sim_id)
    msg.send()
    lsn = Listener()
    resp = lsn.recv()
    while resp !=null:
        read_marker.acquire()
        print(sim.readData(sim_id))
        read_marker.release()

def data_writer(sim_id, data):
    global pointer
    write_marker = marker.gen_wlock()
    lsn = Listener()
    resp = lsn.recv()
    while resp.message == "SIM_START":
        write_marker.acquire()
        pointer = (pointer + 1) % 7
        sim.writeData(sim_id, data)
        write_marker.release()
```

Fig. 4: Python implementation of the readerwriter lock

With this in mind, the simulation engine implements threads and processes at the same time to achieve concurrency and parallelism. Services defined in this section

The "system architecture" can be distributed in three processes:

1) Control process: Serving the request of simulation process control and error report,
2) The simulation process with the following threads:
   a) Execution of the simulation process,
   b) Serving configuration change requests,
   c) Read the data and send the control measures to the IoT system,

3) Results request process: provide services for simulation results.

This implementation allows parallel execution of process control, simulation and calculation processes, and displays the results (if required). The concurrency of the second process is necessary because the execution of the simulation

process depends on the configuration and communication with the intelligent system.

4) API development

The simulation engine communicates with smart systems over an application programming interface (API), more specifically over the REST API. Author [28] defines several reasons why API should be considered as a communication protocol:

1) Manipulation of an extensive data set can be resource consuming,
2) Access to the data set should be real-time,
3) Frequent data set updates
4) Access to a portion of the data set at any time
5) Manipulation of the data set by other users

Continuous system simulation engine generates large dataset over a short period of time; just single simple simulation of the motion with the duration of 20 seconds can produce 14000 data entries., where:

1) Engine retrieves output data from the IoT system
2) IoT listens for control action from the engine.

In previous research [2][29] Django framework [30] was considered for the implementation of required engine functionalities, but after further research and analysis the API was developed using Python Flask framework. Flask is a lightweight web server gateway interface, a micro framework that keeps a core of the application simple but extensible [31] [32].

IV. Conclusion

The system architecture and Python programming concepts in the continuous system simulation engine development process are the focus of this article. In the previous research, a formal model and system architecture were established

The white paper proposes an engine design that allows concurrent and parallel execution of some important services:

1) Simulation control and error handling,
2) Simulation process and communication with intelligent systems,
3) Display the simulation results.

Acknowledgement

The authors thank the Ministry of Education, Science and Technology Development for their support. Grant number 174031.

References

[1] Radenkovi c´, M. Stanojevic´, and A. Markovic´, Racˇunarska simulacija. Belgrade: Fakultet organizacionih nauka, 1999.
[2] T. Naumovi c´, L. Baljak, L. Živojinovic´, and F. Filipovic´, "Development of software framework for real-time management of intelligent devices ," Proc. ISP RAS, vol. 5, no. 31, pp. 35–46, 2019, doi: 10.15514/ISPRAS-2019-31(3)-3.
[3] F. E. Cellier and E. Kofman, Continuous system simulation. Springer US, 2006.
[4] M. Despotovi c´-Zrakic´, D. Barac´, Z. Bogdanovic´, B. Jovanic´, and B. Radenkovic´, "Software Environment for Learning Continuous System Simulation," Acta Polytech. Hungarica, vol. 11, no. 2, pp. 187–202, 2014.
[5] "Simulink Simulation and Model-Based Design MATLAB & Simulink." [Online]. Available: https://www.mathworks.com/products/simulink.html. [Accessed: 30-Jan-2020].
[6] "CSMP Katedra za elektronsko poslovanje." [Online]. Available: https://elab.fon.bg.ac.rs/softver/csmp/. [Accessed: 20-Jan-2020].
[7] "Insight Maker Free Simulation and Mod eling in your Browser." [Online]. Available: https://insightmaker.com/. [Accessed: 30-Jan2020].
[8] M. F. Aburdene, Computer Simulation of Dynamic Systems,. Dabuque, Iowa: Wm. C. Brown Publishers, 1988.
[9] J. W. F. Germeshausen, "The Beginning of System Dynamics," Stuttgart, Germany, 1989.
[10] H.S. Han, "Web-based dynamic simulation system for multi-body systems," Adv. Eng. Softw., vol. 35, no. 2, pp. 75–84, 2004, doi: 10.1016/j.advengsoft.2003.10.003.
[11] G. D'Angelo and M. Marzolla, "New trends in parallel and distributed simulation: From many-cores to Cloud Computing," Simulation Modelling Practice and Theory, vol. 49. Elsevier B.V., pp. 320–335, 01-Dec-2014, doi: 10.1016/j.simpat.2014.06.007.
[12] J. A. Miller, A. F. Seila, and X. Xiang, "JSIM web-based simulation environment," Futur. Gener. Comput. Syst., vol. 17, no. 2, pp. 119– 133, 2000, doi: 10.1016/S0167-739X(99)00108-9.
[13] B. Radenkovi c´, "Program za simulaciju kontinualnih i diskretnih sistema CSMP/MICRO," Automatika, pp. 235–238, 1984.
[14] L. Mili c´evic´, A. Markovic´, and B. Radenkovic´, "Modeliranje i simulacija kontinualnih sis tema u CSMP-W95/NT simulacionom jeziku," SYMOPIS '98, Herceg Novi, Zb. Rad., pp. 627–630, 1998.
[15] C. Runge, "Ueber die numerische Auflösung von Differentialgleichungen," Math. Ann., vol. 46, no. 2, pp. 167–178, Jun. 1895, doi:10.1007/BF01446807.
[16] Z. Drma cˇ, V. Hari, M. Marušic´, M. Rogina, S. Singer, and S. Singer, Numericˇka analiza. Zagreb: SVEUCILIŠTE U ZAGREBU, PMF – MATEMATICˇ KI ODJEL, 2003.
[17] "Documentation — SciPy.org." [Online]. Available: https://www.scipy.org/docs.html. [Accessed: 30-Jan-2020].
[18] A. Silberschatz, "Chapter 4: Threads," in Operating System Concepts, 9th ed., 2013.
[19] M. Flynn, "Flynn's Taxonomy," in Encyclopedia of Parallel Computing, Springer US, 2011, pp. 689–697.
[20] G. Zaccone, Python parallel programming cookbook: over 70 recipes to solve challenges in multithreading and distributed system with Python 3.
[21] G. Lanaro, Python high performance programming: boost the performance of your Python programs using advanced techniques. Packt Pub, 2013.

[22] S. H. Unger, "Hazards, Critical Races, and Metastability," IEEE Trans. Comput., vol. 44, no. 6, pp. 754–768, 1995, doi: 10.1109/12.391185.
[23] E. W. Dijxstra, "Solution of a Problem in Concurrent Programming Control."
[24] "Global Interpreter Lock Python Wiki." [Online]. Available:
[25] https: //wiki.python.org/moin/GlobalInterpreterLock. [Accessed: 29Jan-2020].
[26] "threading — Thread-based parallelism — Python 3.8.1 documentation." [Online]. Available: https://docs.python.org/3/library/threading.html. [Accessed: 29-Jan2020]. "readerwriterlock PyPI." [Online]. Available: https://pypi.org/project/readerwriterlock/. [Accessed: 30-Jan-2020].
[27] P. J. Courtois, F. Heymans, and D. L. Parnas, "Concurrent Control with 'Readers' and 'Writers,'" Commun. ACM, vol. 14, no. 10, pp. 667–668, Oct. 1971, doi: 10.1145/362759.362813.
[28] P. Smyth, "Creating Web APIs with Python and Flask | Programming Historian," 2018. [Online]. Available: https://programminghistorian.org/en/lessons/creatingapis-withpython-and-flask. [Accessed: 30-Jan2020].
[29] T. Naumovi c´, B. Radenkovic´, M. Despotovic´Zrakic´, D. Barac´, and A. Labus, "A framework for real-time management of intelligent devices: an educational perspective," in International Conference on New Horizons in Education, 2018, pp. 33–34.
[30] "The Web framework for perfectionists with deadlines | Django." [Online]. Available: https://www.djangoproject.com/. [Accessed: 30-Jan-2020].
[31] "Welcome to Flask — Flask Documentation (1.1.x)." [Online]. Available: https://flask.palletsprojects.com/en/1.1.x/. [Accessed: 30Jan-2020].
[32] "Flask vs Django | Django Flask Comparison | Blogs @ Mindfire Solutions." [Online].
[33] T. Naumovic, M. Despotovic-Zrakic, B. Radenkovic, L. Zivojinovic and I. Jezdovic, "Development of a Continuous System Simulation Engine in Python Programing Language," 2020 19th International Symposium INFOTEH-JAHORINA (INFOTEH), East Sarajevo, Bosnia and Herzegovina, 2020, pp. 1-5, doi: 10.1109/INFOTEH48170.2020.9066334.

