2023 IEEE 21st World Symposium on Applied Machine Intelligence and Informatics (SAMI) | 979-8-3503-1986-6/23/$31.00 ©2023 IEEE | DOI: 10.1109/SAMI58000.2023.10044485

An application programming interface for the widely used academic version of the UVA/Padova Type 1 Diabetes Mellitus Metabolic Simulator

Máté Siket[*, †, §], Rebeka Tóth[*], László Szász[*], Kamilla Novák[*, §], György Eigner[*, ‡], and Levente Kovács[*, ‡]
[*]Physiological Controls Research Center, Óbuda University
[‡]Biomatics and Applied Artificial Intelligence Institute, John von Neumann Faculty of Informatics, Óbuda University
[§]Applied Informatics and Applied Mathematics Doctoral School, Óbuda University,
H-1034, Budapest, Bécsi street 96/B.
Email: {siket.mate, toth.rebeka, szasz.laszlo, novak.kamilla, eigner.gyorgy, kovacs }@uni-obuda.hu
[†]Institute for Computer Science and Control,
Kende utca 13-17, H-1111, Budapest, Hungary
Email: siket.mate@sztaki.hu

***Abstract*—The UVA/Padova Type 1 Diabetes Simulator is a widely applied tool to test control algorithms among diabetes researchers. The academic version is implemented in Matlab; while Matlab is very popular in the academic field, Python is the most popular programming language. We developed an application programming interface (API) for the simulator in Python, and a representational state transfer (REST) API for additional extension route with a Python reference implementation for easier usage. The interface is designed with the specific purpose of testing control algorithms, thus it maps the functionalities that are needed to implement closed-loop control and virtual patient simulation. The developed API is tested for different simulation scenarios, showcasing identical results between the API and the programming interface of the original simulator. Additional information and the source code can be found in https://github.com/NeuroDiab/UVAPadovaAPI.**

I. Introduction

UVA/Padova Type 1 Diabetes Metabolic simulator [1], [2] is a widely used tool among diabetes researchers during the development, testing and validation phase of their work [3]–[8]. The simulator serves as an in-silico trial, which allows the replacement of animal trials, as it was approved by FDA (U.S. Food and Drug Administration) in 2008 [1], [9]. Multiple studies exist not only using but also expanding and improving the simulator. For instance, Man and colleagues expanded the original system, including a glucose and an insulin subsystem, with a glucagon subsystem [9]. This extended version of the simulator was published in 2013 and it has also been approved by the FDA. Visentin and colleagues also had a role in the validation process [10] and continued working on this version of the simulator; in multiple steps [11] they managed to upgrade the original „single-meal" validation option with a more realistic „single-day" one [12]. Due to its complexity, reliability and the FDA approval, researchers often use the UVA/Padova simulator to validate their software. However, this procedure can be complicated, if the language of their algorithm differs from MATLAB, in which the simulator was written. This happens often, as other languages, such as Python, are more popular to use in this scientific field. In this case, researchers need to correspond their software with the simulator, for example by implementing their algorithms in MATLAB. Schmitzer and colleagues implemented the algorithms of AndroidAPS in MATLAB/Simulink to speed up their simulation [13]. Another solution is to implement the simulator into the language of the software requiring validation. Xie did this in 2018, using the original 2008 version of the simulator [14]. However, the complete implementation requires further validation. Even in case of proper validation results, it is difficult to guarantee the perfect match in all parameters, values and structure of the model. To avoid these uncertainties, a possibly better option is to develop an interface between the system and the simulator, leaving both in their original form and environment.

We propose a Python interface for the academic version of the UVA/Padova Type 1 Diabetes Simulator. The interface is developed with the aim of simulating various carbohydrate and insulin intakes and testing control algorithms. Thus, on the one hand the solution makes those features easier to use, on the other hand it does not provide full functionality of the Matlab simulator. To connect the two platforms we used MATLAB Engine API for Python [15]. This appeared to be a better solution than the complete implementation of the simulator into Python. One advantage is that the whole system is more consistent this way since the simulator runs in its original form without making any changes of potential errors. Moreover, a REST API is wrapped around the Python interface for two reasons. First, it makes separating the virtual patient

The work was supported in part by the Eötvös Loránd Research Network Secretariat (Development of cyber-medical systems based on AI and hybrid cloud methods) under Agreement ELKH KÖ-37/2021. Project no. 2019-1.3.1-KK-2019-00007. has been implemented with the support provided from the National Research, Development and Innovation Fund of Hungary, financed under the 2019-1.3.1-KK funding scheme. This project has been supported by the National Research, Development, and Innovation Fund of Hungary, financed under the TKP2021-NKTA-36 funding scheme.

from the rest of the components possible, providing easier embedding in software-in-the-loop and hardware-in-the-loop environments; the REST API [16] allows the Python wrapper to be re-implemented in any preferred language.

II. Materials and methods

A. Interfacing the simulator

In the original simulator the scenarios can be defined via a graphical user interface or by a text file. Then the input data is transformed and fed to the $run\_simulation$ function. In our work, we bypass the scenario definition and data transformation steps and directly feed the input data from Python to the $run\_simulation$, which is embedded in the $connect\_function$ shown in Figure 1. Its results are gathered using the Matlab Engine API and stored in a Python class.

The project implements a client and a host side of the interface. The client wraps the REST API call functions for the creation of the virtual patient, input definitions and simulation. The server side implements the actual interfacing between the Python and Matlab, and simulation by using the Matlab Engine API. Important to note, the Matlab simulator can be interfaced without the REST API layer as well. To do that, the $VirtualPatientT1DMS$ class has to be used; examples are given in [17].

B. Client side

The $UvaPadovaAPI$ class is a reference implementation of the API in Python, it contains the top-level features and implements the virtual patient creation, device settings and the one-step forward simulation via API requests. While the wrapper makes it easy to test control algorithms, it also poses constraints, for example, the $doSimulation$ function propagates the model with a fixed sampling time of 5 minutes. The API requests are wrapped in the following functions:

- $UvaPadovaAPI$: Constructor for the API wrapper, the host address can be set, defaults to local host.
- $initializePatient$: Initializes the virtual patient by defining the ID, additionally the type of the insulin pump and the glucose sensor can be defined.
- $setPump$: Sets the type of the insulin pump.
- $setSensor$: Sets the type of the glucose sensor.
- $doSimulation$: Implements the one-step forward propagation of the virtual patient.

An important note for the correct working of the $doSimulation$ method is that the first call must always be preceded by a patient initialization using the $initializePatient$ method. Otherwise, the server will return a 409 HTTP error code with ”Patient wasn’t initialized.” error message.

Another note that, based on the operation of the UVA/Padova simulator, carbohydrate intake cannot occur at any time. If the $doSimulation$ method is called with carbohydrate value at a time when carbohydrate intake would not be possible, the server ignores the carbohydrate value, continues the simulation and will return a ”Carbohydrate intake was ignored.” alert message in addition to the result. Such scenarios are when consecutive meal intakes are not at least an hour apart or meal intake occurs in the first hour of the simulation.

C. Documentation

Besides storing the source code, the repository provides information on the goals of the project, requirements and setup. A schematic layout of the project can be seen in Fig. 1. Circles in different colors represent alternative ways for interfacing the UVA/Padova simulator. Black circle represents the native Python implementation without the REST API layer, blue circle represents the reference implementation of the client side in Python, while the white circle indicates the extension route for other platforms and languages.

The evaluated test scenarios can be found in the repository and serve as examples. Examples which do not use the REST API layer utilize further libraries which were designed with the purpose of easier scenario definition and data processing. The documentation [17] of these libraries with the documentation of the API can be found also in the repository in HyperText Markup Language format. If the REST API layer is used, Numpy [18] arrays can be defined to describe the schemes of insulin and meal intakes.

III. Results

We tested the interface on two levels: on the level of the Python link to the Matlab simulator and on the API level as well. The differences between the glucose values across all timepoints were 0 for all the predefined 6 scenarios. Table I summarizes the test scenarios, where we aimed for a wide range of settings. Two simulated blood glucose traces can be seen in Fig. 2 and 3, subplot on the left is the result obtained through the API, while the right subplot is generated from the UVA/Padova simulator.

We also measured the running times in the two versions in all test cases. For comparability, we started the measurements just before the $run\_simulation$ function was called, and finished when the function gave back the final results. In the Python version it means the points before and after the call of the Matlab Engine API. Table II shows the measured times.

Table II: Runtimes in the original (Matlab) version and the API version (Python) in all cases.

Test ID	Runtime Matlab [s]	Runtime Python [s]	
		First run	Average of next 10 runs
1	1.239176	31.120796	0.989194
2	0.984779	19.295156	0.864834
3	3.431623	25.137429	3.043499
4	2.165344	21.783566	2.026083
5	0.939902	19.448893	0.914267
6	3.255037	21.710503	2.297150

IV. Discussion

During the test of the developed API, 18 simulations were evaluated; 6 scenarios for the original Matlab version, for the Python implementation and for the REST API reference implementation. For the evaluated test scenarios the differences

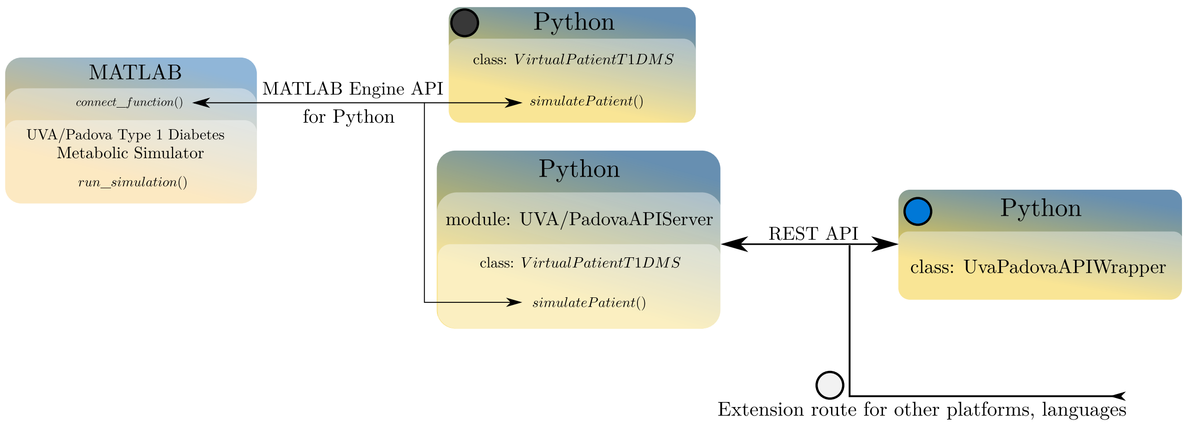
MATLAB
$connect\_function()$
UVA/Padova Type 1 Diabetes Metabolic Simulator
$run\_simulation()$
MATLAB Engine API for Python
Python
class: $VirtualPatientT1DMS$
$simulatePatient()$
Python
module: UVA/PadovaAPIServer
class: $VirtualPatientT1DMS$
$simulatePatient()$
REST API
Python
class: UvaPadovaAPIWrapper
Extension route for other platforms, languages

Figure 1: Project layout, which represents the three major goals: 1) Black circle represents the native Python virtual patient 2) The Blue circle represents the reference implementation of the REST API, and 3) The white circle represents the possibility of additional interfaces for the API.

Table I: Summary of test cases. The default $Generic\_1$ insulin pump was selected in all cases.

Test ID	**Simulation time [min]**	**Basal insulin [U/hour]**	**Insulin bolus [Unit]**	**Bolus time [min]**	**Meal amount [g]**	**Meal time [min]**	**Patient**	**Sensor type**
1	60	0.0	0.1	in every 5 minutes	-	-	adult #5	Guardian
2	60	1.2	-	-	-	-	adult #5	Guardian
3	1440	1.0	1/2	every two hours	30/45	every two hours	adolescent #3	Dexcom
4	1440	0.0	-	-	-	-	adult #2	Dexcom25
5	60	0.0	increasing by 0.1 in every 5 minutes, starting from 0.1	in every 5 minutes	-	-	adult #1	Guardian
6	1440	0.0	[4, 7, 2, 8, 2]	[420, 720, 960, 1080, 1380]	[45, 70, 20, 80, 20]	[420, 720, 960, 1080, 1380]	adolescent #1	Guardian

were 0, which indicate that the API can be used to simulate various patients with different basal, bolus insulin profiles and meal intakes. Although in Table I the results are simulated using the $Generic\_1$ insulin pumps, others were also tested.

Runtimes significantly increase during the first simulation using the API, where the Matlab Engine API initializes. The consecutive simulations average close to the runtime of the Matlab simulations as Table II suggests. Tests were done by Matlab versions 2020a and 2021a, while 2022a supposed to implement significant improvements regarding the Matlab Engine API, so the differences are expected to be lower.

V. Conclusion

In this paper we introduced an API for the academic version of the UVA/Padova Type 1 Diabetes Metabolic Simulator. The interface achieves three goals: First, it can be used directly in Python; through REST API using a reference implementation; and also can be extended to other platforms using the API. Results show that the original Matlab version, and the interfaced versions do not differ in terms of the glucose values for the test scenarios, but due to the Matlab Engine API the runtimes of the interfaced versions are significantly increased.

References

[1] B. P. Kovatchev, M. Breton, C. Dalla Man, and C. Cobelli, "*In Silico* Preclinical Trials: A Proof of Concept in Closed-Loop Control of Type 1 Diabetes," *Journal of Diabetes Science and Technology*, vol. 3, no. 1, pp. 44–55, Jan. 2009. [Online]. Available: http://journals.sagepub.com/doi/10.1177/193229680900300106

[2] "Diabetes Simulation." [Online]. Available: https://tegvirginia.com/services/diabetes-simulation/

[3] B. Kovatchev, C. Cobelli, E. Renard, S. Anderson, M. Breton, S. Patek, W. Clarke, D. Bruttomesso, A. Maran, S. Costa, A. Avogaro, C. D. Man, A. Facchinetti, L. Magni, G. De Nicolao, J. Place, and A. Farret, "Multinational Study of Subcutaneous Model-Predictive Closed-Loop Control in Type 1 Diabetes Mellitus: Summary of the Results," *Journal of Diabetes Science and Technology*,

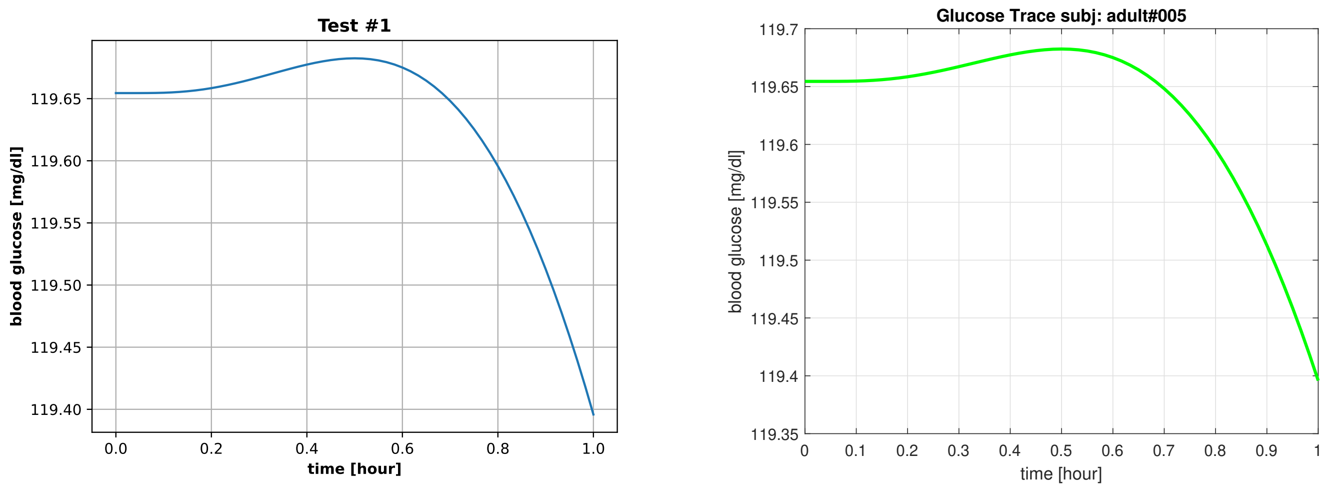

Figure 2: Result of test #1 regarding the Engine API version (left) and the original version of the simulator (right).

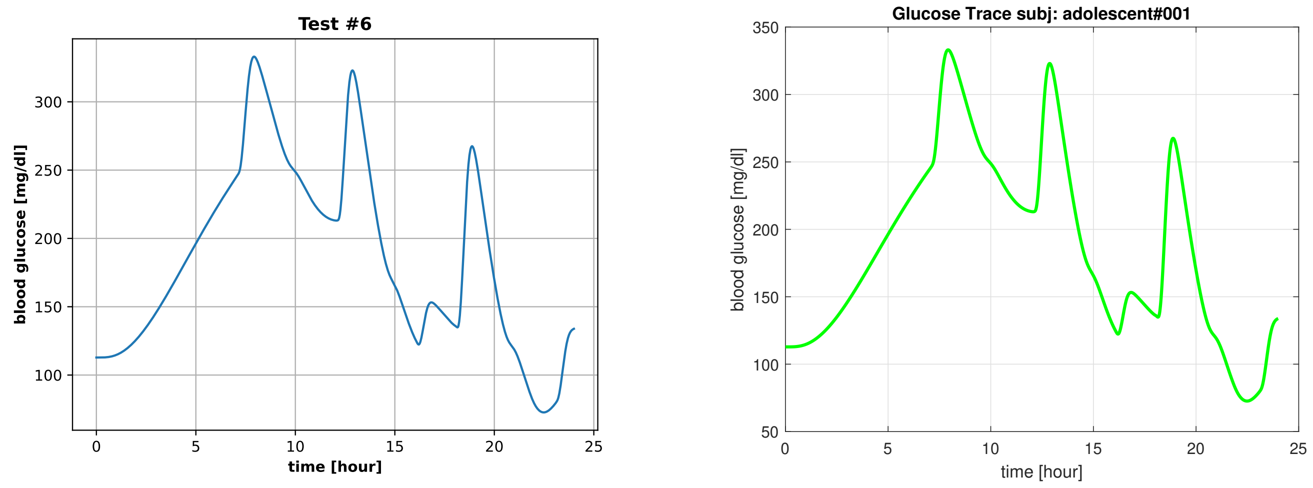

Figure 3: Result of test #6 regarding the Engine API version (left) and the original version of the simulator (right).

vol. 4, no. 6, pp. 1374–1381, Nov. 2010. [Online]. Available: http://journals.sagepub.com/doi/10.1177/193229681000400611

[4] VisentinRoberto, GiegerichClemens, JägerRobert, DahmenRaphael, BossAnders, GrantMarshall, D. ManChiara, CobelliClaudio, and KlabundeThomas, "Improving Efficacy of Inhaled Technosphere Insulin (Afrezza) by Postmeal Dosing: In-silico Clinical Trial with the University of Virginia/Padova Type 1 Diabetes Simulator," *Diabetes Technology & Therapeutics*, Sep. 2016, publisher: Mary Ann Liebert, Inc. 140 Huguenot Street, 3rd Floor New Rochelle, NY 10801 USA. [Online]. Available: https://www.liebertpub.com/doi/10.1089/dia.2016.0128

[5] A. Molano-Jiménez and F. León-Vargas, "UVa/Padova T1DMS dynamic model revision: For embedded model control," in *2017 IEEE 3rd Colombian Conference on Automatic Control (CCAC)*, Oct. 2017, pp. 1–6.

[6] M. D. Breton, R. Hinzmann, E. Campos-Nañez, S. Riddle, M. Schoemaker, and G. Schmelzeisen-Redeker, "Analysis of the Accuracy and Performance of a Continuous Glucose Monitoring Sensor Prototype: An In-Silico Study Using the UVA/PADOVA Type 1 Diabetes Simulator," *Journal of Diabetes Science and Technology*, vol. 11, no. 3, pp. 545–552, May 2017. [Online]. Available: http://journals.sagepub.com/doi/10.1177/1932296816680633

[7] E. Campos-Náñez, J. E. Layne, and H. C. Zisser, "In Silico Modeling of Minimal Effective Insulin Doses Using the UVA/PADOVA Type 1 Diabetes Simulator," *Journal of Diabetes Science and Technology*, vol. 12, no. 2, pp. 376–380, Mar. 2018. [Online]. Available: http://journals.sagepub.com/doi/10.1177/1932296817735341

[8] "Long-acting Insulin in Diabetes Therapy: In Silico Clinical Trials with the UVA/Padova Type 1 Diabetes Simulator," Jul. 2018, pp. 4905–4908, iSSN: 1558-4615.

[9] C. D. Man, F. Micheletto, D. Lv, M. Breton, B. Kovatchev, and C. Cobelli, "The UVA/PADOVA Type 1 Diabetes Simulator: New Features," *Journal of Diabetes Science and Technology*, p. 9.

[10] R. Visentin, C. Dalla Man, B. Kovatchev, and C. Cobelli, "The University of Virginia/Padova Type 1 Diabetes Simulator Matches the Glucose Traces of a Clinical Trial," *Diabetes Technology & Therapeutics*, vol. 16, no. 7, pp. 428–434, Jul. 2014. [Online]. Available: http://www.liebertpub.com/doi/10.1089/dia.2013.0377

[11] R. Visentin, C. D. Man, and C. Cobelli, "One-Day Bayesian Cloning of Type 1 Diabetes Subjects: Toward a Single-Day UVA/Padova Type 1 Diabetes Simulator," *IEEE Transactions on Biomedical Engineering*, vol. 63, no. 11, pp. 2416–2424, Nov. 2016, conference Name: IEEE Transactions on Biomedical Engineering.

[12] R. Visentin, E. Campos-Náñez, M. Schiavon, D. Lv, M. Vettoretti, M. Breton, B. P. Kovatchev, C. Dalla Man, and C. Cobelli, "The UVA/Padova Type 1 Diabetes Simulator Goes From Single

Meal to Single Day," *Journal of Diabetes Science and Technology*, vol. 12, no. 2, pp. 273–281, Mar. 2018. [Online]. Available: http://journals.sagepub.com/doi/10.1177/1932296818757747

[13] J. Schmitzer, C. Strobel, R. Blechschmidt, A. Tappe, and H. Peuscher, "Efficient Closed Loop Simulation of Do-It-Yourself Artificial Pancreas Systems," *Journal of Diabetes Science and Technology*, vol. 16, no. 1, pp. 61–69, Jan. 2022. [Online]. Available: http://journals.sagepub.com/doi/10.1177/19322968211032249

[14] J. Xie, "simglucose," Aug. 2022, original-date: 2017-12-31T19:15:11Z. [Online]. Available: https://github.com/jxx123/simglucose

[15] "Get Started with MATLAB Engine API for Python - MATLAB & Simulink." [Online]. Available: https://www.mathworks.com/help/matlab/matlab_external/get-started-with-matlab-engine-for-python.html

[16] A. Ospanova, A. Zharkimbekova, L. Kussepova, A. Tokkuliyeva, and M. Kokkoz, "Cloud service for protecting computer networks of enterprises using intelligent hardware and software devices, based on raspberry pi microcomputers," *Acta Polytechnica Hungarica*, vol. 19, no. 4, 2022.

[17] "Welcome to uva/padova api's documentation." [Online]. Available: https://neurodiab.github.io/UVAPadovaAPI/

[18] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020. [Online]. Available: https://doi.org/10.1038/s41586-020-2649-2