2021 International Seminar on Electron Devices Design and Production (SED)

Client-Server Implementation in Solving the Problem of Modeling the Small Spacecraft Flight at Low-Earth Orbit

V.Baleskin, A. Kosinov, L. Romanov, I. Motajlenko,
National Research University Higher School of Economics, Moscow
vbaleskin@hse.ru, akosinov@hse.ru, lromanov@hse.ru, imotajlenko@hse.ru

***Abstract*—The possibility of implementing a modeling of the flight of a small spacecraft in low earth orbit based on a client-server architecture was researched. Issues such as transmitting and displaying the system state, aggregating states at different points in time, and executing user code were discussed. The conducted studies demonstrate the benefits of using a client-server implementation in solving the problem of simulating the flight of a small spacecraft in near-earth orbit in a world of increasing interest in satellite developments.**

Keywords—client-server systems, low earth orbit satellites, design automation, computer simulation

I. INTRODUCTION

In modern realities, people's interest in space exploration, in particular in the development of satellite technologies, is actively growing. As a result, the demand for professionals in this area is also increasing. This causes an increase in demand for educational resources on the topic. The creation of new software that simulates the flight of a small spacecraft in near-earth orbit makes it possible to increase the availability of the necessary knowledge for people wishing to continue working in this area in the future. However, nowadays, only few have expensive hardware to use demanding engineering software.

There are several options of implementation for such software. One of the most affordable models is the client-server model, which allows you to work with applications via an external, more powerful computer. Already existing analogues of such applications, for example, Autodesk CAD [1], have shown their efficiency and usefulness, but at the moment all such services have their use in other areas.

All procedures are sent to execute by the client remotely and processed directly on the server. This can significantly reduce the necessary system requirements for the user, as well as expand the potentially interested audience of the software. In addition, when one need to make some calculations on the road, this model allows to send a request to the server and perform all the necessary calculations remotely.

A. Client-server architecture

Client-server model is an architecture that consists of two main parts: client systems and server systems. Data is exchanged between them. A client-server application, therefore, is a category of distributed systems consisting of client and server software. Client application constantly maintains a connection with the server and sends requests, and the server application listens for requests from the client. Such applications provide an opportunity to distribute the load, and to transfer part of the load from the client system to the server one.

To implement communication and load distribution, it is necessary to define the interface between the client or server. In other words, to create an Application Programming Interface (API).

B. Remote Procedure Call

Remote Procedure Call (RPC) is an interprocess communication method that allows a computer program to call the execution of a procedure in a different address space, coded as a normal procedure call. In addition, RPC allows low-level transport protocol control.

C. Small spacecrafts flight simulation in low-earth orbit

In order to simulate the flight of a small spacecraft in a near-earth orbit, it is necessary to determine the state of the spacecraft and the state of the environment at each moment of time. Earth-centric inertial coordinate system may be used as the basic coordinate system for calculating the parameters of the motion of dynamic objects. The very same model of the Earth is set in the geocentric inertial coordinate system. The angle of rotation is determined by time. The motion of a spacecraft as a rigid body can be modeled by determining at each moment of its position and orientation in the geocentric inertial coordinate system.

D. Spacecrafts position modelling; Simplified General Perturbation; Two-Line Element

The main reason to model the position of a spacecraft is to predict the position of that spacecraft at a specific point in time, usually based on the orbital parameters. SGP4 - is an orbit prediction model that was developed by the headquarters of the North American Air Defense Command. Together with the SSN observational materials, the largest database of space objects in the world [2] was generated and presented in the form of TLE. TLE is a two-line data format representing a set of orbit parameters for an Earth satellite. Due to the fact that this database was generated using SGP4, it is impossible to achieve better prediction accuracy for TLE elements without using SGP4 itself. This leads us to the conclusion that SGP4 is widely used in all kinds of tasks. There are also HSGP / HTLE models for more accurate modeling, however, they are not required for simulation purposes [3].

E. Earth's geomagnetic field modeling; International Geomagnetic Reference Field

The reason to model the Earth's magnetic field is to predict the induction vector or potential of the magnetic field at a given point, acting on the spacecraft from the side of the

978-1-6654-3941-1 @2021 IEEE

Earth's magnetic field. IGRF is a series of mathematical models describing the main field of the Earth, as well as possible changes. The formula used in the model is as follows:

$$V(r,\theta,\phi,t) = a\sum_{n=1}^{N}\sum_{m=0}^{n}[g_n^m(t)\cos(m\phi) + h_n^m(t)\sin(m\phi)]P_n^m(\cos\theta) \quad (1)$$

In (1), a = 6371.2 km, truncation degree N = 13. Many institutes and private organizations involved in the operation of magnetic imaging satellites constantly maintain and update the IGRF [4].

II. Methodology

A. Client-server data exchange

The project consists of three categories of entities. First on is client (including web-clients), that requests computation. The second is computation server, that responds to the requests by performing computations. Last one is the system that is being modeled. Client and server engage in a data exchange. Such data consists of sequence of remote procedure calls (RPC) that adhere to an API. In current project, gRPC [5] chosen as method for interface description due to its relative simplicity of use and comparatively low overhead (partly due to binary serialization).

B. System state; discretization of system state over time

To transmit and display the state of a system via RPC, the system's state is requested from server in the set point in time. This method implies an explicit discretization of the system's state over time. Hence, each interval $[t_0 + i_{dt}; t_0 + dt(i+1)]$ of length dt is being mapped to discrete value t_i. Each t_i is consecutively mapped to a system's state. This forms a sequence S_n of system states, where each element of a sequence is called a frame of the system's state. By defining a state transition function F: $S_i \rightarrow S_{i+1}$, as well as an initial state S_0, it becomes possible to model the later frames.

Each frame consists of the state of the modeled spacecraft (and its subsystems) and the environment's state. This suggests, the new spacecraft state at each iteration is unstable until the end of the said iteration. For that reason, the usage of the new spacecraft state by the subsystem models limited to write-only access. On the other hand, the environment's state is interchangeable throughout the simulation and hence can be read by the subsystems. The aforementioned properties lead to state transition function assuming the following form: T: $T(S_i, e_{i+1}) \rightarrow S_{i+1}$ where S_i represents the prior state of the spacecraft, e_i is the current state of the environment and S_{i+1} the new state of the spacecraft.

C. Distribution of computation

Spacecraft modelling can be optimized by running computations in parallel to each other. As an instance, modeling the spacecraft's position as well as the environmental state can be performed independently of the spacecraft state computation. Therefore, computation of certain parameters separated into services. Services are the components that are independent from the main system. To be specific, computations of parameters such as the spacecraft position, geomagnetic field, solar position and visibility of the Sun can be separated into different services. These computations are required for practically every frame, yet they do not require information on the spacecraft state. Such properties also make it possible to perform computations in batches.

D. Modeling the system and the subsystems

In most cases of modelling a system, the internal structure is considered to be known only to a certain extent. At the highest level of abstraction, the system can be modeled as a "black box" that has a "state" as well as a state transition mechanism. At a lower level it is described by a set of interacting "black boxes" - subsystems. Likewise, each subsystem can be considered as its own set of interacting "black boxes". With increasing "depth", the number of elements present in the system grows, as does the complexity of modeling. In case of a client-server implementation with many concurrent users, a compromise has to be reached. The system has to work fast enough to serve the required number of users while fulfilling the quality-of-service requirements; At the same time, only a limited amount of resources can be allocated for each user. Finally, the users should, to a certain degree, be able to design and model different spacecraft configurations. For this work it was decided to only consider the system at a "set of subsystems" level. The focus lies with the small spacecraft at low earth orbit - specifically, cubesats from one to three units (1U - 3U). Since the cubesat design specification [6] defines the size of such spacecraft, some module size can be considered a unit. That unit - a "size of a slot" – allows to represent a spacecraft of each configuration as a number of available slots. Size of each module can also be represented as a number of slots. Finally, the slots of a single spacecraft can be indexed and mapped to the modules they contain as well as the information about them.

E. State transition mechanism

As mentioned earlier, modeling the system requires defining the notion of state as well as a state transition function. However, the modular nature of the system allows to describe every subsystem the same way. Several measures can be taken. First is to define the state of a system to include the states of its modules. Second – to make all the state transition functions to process the state of a system. This reduces the state transition for the system to generating the initial value of a new state, updating common parameters, and traversing the state transition functions of the modules. Considering that, as stated earlier, the new state is generally unstable until the end of the iteration, read access to the new state is limited. This makes it difficult to model the concurrent functioning of several similar modules.

F. Modeling the algorithm provided by the user

In modeling the spaceflight, it is important to provide a way to model the controlling algorithm as well. For this purpose, mechanisms of checking and executing the user-provided code are required, as well as the data exchange system and the API.

For the checking mechanism it is enough to restrict the usage of the language's standard library as well as the third-party libraries. For the C programming language, the standard library restriction can be reduced to prohibiting the usage of most of the standard library headers. A limit of one source code file per simulation can be introduced to restrict the usage of third-party libraries. Obviously, while working in a trusted environment, such as an internal network, some if not most of the limitations may be redundant.

Code execution functionality is mostly equivalent to a combination of code compilation and execution mechanism with the data exchange system. The compilation, in case of a successful check, represents an execution of a third-party compiler. The execution is, effectively, instantiation of a child process using the compiled executable as a base. Finally, the data exchange system can be implemented via methods such as named pipes, sockets, or memory-mapped files. Due to somewhat larger delays, sockets can be excluded from the list. In comparison, in Unix-based environments the difference between named pipes and memory-mapped files is smaller, so the choice is not that important. From the standpoint of the simplicity of development, it makes sense to use two channels of different directions, as compared to using one bidirectional channel. As for the API implementation, two entities are introduced on the side of the main system – the command queue and the command processor. Every time an API call is made in the user code, the user subsystem generates a message in the data exchange system. Each message contains the command name and the arguments. The main system accepts the message, reconstructs the command, and places it at the end of the queue. Each command has a cost assigned to it and the processor has a budget for execution. At every state transition the command processor takes one or multiple commands from the front of the queue and executes them. Commands can be classified as blocking and non-blocking. A non-blocking command does not require a response to be sent to the user subsystem and does not require the flow of user code execution to be halted. After a blocking command is sent, the flow of execution is interrupted until the command is executed at the main system and a response is received. The general algorithm is to execute commands from the queue while the budget is above zero. The processor executes the command in the front of the queue and subtracts its cost from the budget. If executing it will cause the budget to drop below zero, the execution is postponed until the next iteration. At every iteration the processor budget remainder is increased by the initial value of the budget, but the total budget is limited at twice the initial value.

G. Timing and aggregation mechanism

In the exchange of data within the client-server system, timing is extremely important. Assuming framerate f and number of users n, the share of time available for one user is $1/n$; the share of time per frame is hence $1/f_n$. The time since the beginning of the simulation measured at the server may differ from the same time measured at the client. That may happen due to frame loss or errors with displaying. Generating frames at frequency that is higher than the framerate allows to lessen the scale of this problem. To be specific, let the generation rate be the display rate times k, for some natural number k. Then, on average, for each frame displayed, k frames will be lost. However, that will also increase both the accuracy of the simulation and the quality of service for the client. In addition, with generation rate exceeding framerate, it becomes possible to view the modeling process with more detail.

The amount of frames the client requests at once is also worth considering. Requesting frames in batches reduces delay as long as the average frame size is below the packet's maximum size. That causes a problem, since the frame that the user sees when the command is sent will generally not be the same as the last computed frame. Still, in real conditions there also exists a significant (relative to the period of time corresponding to a single frame) delay between when the command is sent and when it is executed. Furthermore, limiting the interaction between the user and the model throughout the simulation reduces the significance of this problem. It can completely disappear if such interaction is prohibited.

The number of frames to be requested at once can be derived at client by keeping track of several parameters. Specifically, time t_r since the last request, amount of frames N that were used since then, and the expected time t_e till the next request. Rate of frame usage $\frac{N}{t_r}$, multiplied by t_e gives an estimate $n = N\left(\frac{t_e}{t_r}\right)$ for the number of frames required. The minimum buffer size can also be found - at any moment, the buffer must contain enough frames to keep functioning properly long enough for the new batch of frames to be requested and received. This critical amount N_{cr} is equal to the amount of time the request takes divided by the rate of frame usage. It is also possible to use two buffers instead of one to make the user experience smoother. When one of the buffers is emptied, the buffers are exchanged, and new frames are requested into the now-empty buffer.

This way, the size of a buffer at any given moment of time must be greater than N_{cr}, and greater than any possible sum of N and N_{cr}, or n and N_{cr}, which can be achieved by making the buffer dynamic. It is possible to introduce an algorithm for requesting new frames before reaching the critical amount. For instance, by requesting frames when kN_{cr} frames are left in the current buffer, where k is the "instability" quotient. As an initial value, 2 is enough, then it can be corrected after each request. If by the time the new frames are received less than N_{cr} elements remain in the buffer, k has to be increased, and decreased otherwise. The absolute value of a change can be named dk. For each successive case, where less than N_{cr} frames remained—for each successive failure—dk can be increased by a factor of two. For each consecutive case of the opposite - decreased by a factor of two. As an initial value, dk can be taken to be 0.5, and the maximum value of k can be taken to be 17.5, which corresponds to five consecutive failures from the initial value.

H. Modeling and accounting for the specifics of the real-world spacecraft

The number of spacecraft in orbit of our planet only increases each year. According to statistics, in 2020, the number of launched satellites exceeded 900 [7]. The cosmonautics history so far has only one major unplanned collision of satellites, which lead to release of a huge amount of space debris [8] and many cases of targeted spacecraft destruction. The moment of spacecraft collision with debris becoming not an accident, but an economic risk is not far off. Therefore, when modeling the orbit of a spacecraft, it is important to consider not only its mission, but also the features of neighboring vehicles, whether they are working satellites or what is left of them. To account for the majority of satellites, it is sufficient to regularly update the database of orbit parameters (TLE), but satellites with engines for orbital movement can significantly change their position in a short time. That means, using regular TLE databases is not enough. To consider all risks, it is required to form a database containing not only the information about the

location of the vehicle and its orbit, but also the details of the mission, payload and orientation systems.

III. Results

Throughout modeling the spacecraft position via SGP4 model, computation delays were measured for a case of 1 frame per request and varying levels of aggregation. Results provided in Table I correspond to a configuration where the server system is located on the same device as the client system. The results display an obvious decrease in delay per frame with the rising batch size. This data implies that the computation itself takes less time than sending and receiving data. However, the batch size is limited as the overall delay increases with the number of frames per request, and only a limited share of computation time can be allocated to a single user. In case of hundred concurrent users, allocated to user 10 microseconds each every second of server work, which is more than enough to generate the number of frames needed.

TABLE I. The dependence of delays on the number of frames per request for the SGP4 model

Frames per request (batch size)	Delay (microseconds)	Delay per frame (microseconds)
1	57.8857	57.8857
10	71.5283	7.15283
25	95.9283	3.83713
50	128.01	2.5602
75	163.585	2.18113
100	193.436	1.93436
150	258.443	1.72295
200	321.611	1.60806
300	465.759	1.5492

Model construction time, which is the time taken to compute the model parameters, is not considered in table 1, as it happens only once per simulation. In addition, the table only contains the data for the case of the server and client simultaneously residing on the same device. However, the delay between the client and server can be considered constant, meaning the results comparable to the configuration with remote clients.

As the results of environmental computation are similar to results in table 1 they are not mentioned separately. The data also shows the same trends for the simulation, since the computation on the server proceed independently from user's actions.

IV. Discussion

The simulations performed show the delays of information transmission between the client and the server. Throughout the normal workflow, the user can encounter a varying delay depending on the location and connection quality of the user. Aggregation of frames allows to smoothen the effects of this delay. Although, this causes the inconsistency between the last moment displayed and the last moment computed. This problem is not significant if we take into account the fact that the software does not imply direct user interaction with simulation during the computing process, which in turn smoothens the majority of the negative aspects of the user experience associated with using remote devices.. Furthermore, the inconsistency problem can also be mitigated via the introduction of mid-simulation interruptions with the purpose of correcting the model.

V. Conclusion

The growing interest in the field of satellite developments, as well as the general growth of planetary launches, associated with the development of microelectronics and the popularity of nanosatellites, entails an increase in demand for professionals in the field. The results of this work demonstrates the benefits of using client-server implementation in solving the problem of simulating the flight of a small spacecraft in near-earth orbit in order to increase the overall availability of software tools for both education and professional work. The consequences of the pandemic in the IT sector have revealed a clear lack of devices capable of even a simple video call, not to mention a CPU-dependent computing, among users. This problem has led to the inability of suppliers to meet the demand of buyers, such as office workers switching to "work from home" lifestyle. The problem of software availability for an ordinary user does not stop at the license agreement, but directly depends on the user's equipment with the appropriate hardware. Projects like this can serve as an example of improving the availability of educational, professional and semi-professional software. Positive results of the system's efficiency consider the possibility of developing professional software for modeling flight of a small spacecraft in near-earth orbit using client-server architecture. Such a modelling task will only get more complicated over time, as the growth of outer space clutter has already begun to influence the design of the latest vehicles capable of self-destruction by burning in the atmosphere [9]. The question of spacecraft in orbit safety ceases to lean only on the choice of the orbit, and soon for this it will be necessary not only to implement orbital navigation, but to take into account similar functionality in other devices. Such task is impossible without taking into account a huge number of parameters and external influences, which can be taken into account only with full computer modeling.

VI. Acknowledgment

Authors note that this article was prepared within the framework of the Basic Research Program at the National Research University Higher School of Economics.

References

[1] "Cloud based online cad software." Accessed on: Jan. 20, 2020. [Online]. Available: https://www.autodesk.ru/solutions/cloud-based-online-cad-software.

[2] V. ~M. Agapov, "Space Objects Data Catalogue," in *Space Debris*, 2001, vol. 2, pp. 759–763.

[3] W. Dong and Z. Chang-yin, "An Accuracy Analysis of the SGP4/SDP4 Model," *Chinese Astron. Astrophys.*, vol. 34, no. 1, pp. 69–76, 2010, doi: https://doi.org/10.1016/j.chinastron.2009.12.009.

[4] "IAGA V-MOD Geomagnetic field modeling: International Geomagnetic Reference Field." Accessed on: Dec. 1, 2020. [Online]. Available: https://www.ngdc.noaa.gov/IAGA/vmod/igrf.htm.

[5] "gRPC." Accessed on: Sep. 19, 2020. [Online]. Available: https://grpc.io.

[6] "CubeSat Design Specification Rev. 13, The CubeSat Program, Cal Poly SLO." Accessed on: Oct. 3, 2020. [Online]. Available: https://static1.squarespace.com/static/5418c831e4b0fa4ecac1bacd/t/56e9b62337013b6c063a655a/1458157095454/cds_rev13_final2.pdf.

[7] "Orbital Launches of 2020 - Gunter's Space Page." Accessed on: Dec. 3, 2020. [Online]. Available: https://space.skyrocket.de/doc_chr/lau2020.htm.

[8] “2009 Iridium-Cosmos Collision Fact Sheet.” Accessed on: Sep. 21, 2020. [Online]. Available: https://swfound.org/media/6575/swf_iridium_cosmos_collision_fact_sheet_updated_2012.pdf.
[9] “Liou J. C. The 2019 US government orbital debris mitigation standard practices. – 2020.” Accessed on: Dec. 3, 2020. [Online]. Available: https://orbitaldebris.jsc.nasa.gov/library/usg_od_standard_practices.pdf.