

Distributed Administration of Multi-Agent Model Properties

Alena Burova^a, Sergey Burov^a, Danila Parygin^a, Alexander Gurtyakov^a and Nikolay Rashevskiy^a

^a *Volgograd State Technical University, 28 Lenina Ave., Volgograd, 400005, Russia*

Abstract

Modeling long-term or large-scale processes is associated with a significant investment of researchers' time, as well as computer time. In this regard, it can be effective to make adjustments to the model directly in the process of modeling. This article discusses the design and implementation of a web client that acts as an administration system (panel) for a platform for multi-agent modeling of movements and interactions of actors within a city map area. All modeling logic in this platform is implemented directly in modules, while the modeling platform only calls it for specific, connected modules. The modeling platform is implemented on the ASP.NET Core 5.0 framework. For the implementation of the web client, the Angular 11 framework was chosen with the Ant Design UI components.

Keywords

Simulation, city, C#, modularity, client-server, administration panel

1. Introduction

The study of socio-economic systems, aimed at analyzing the interactions of subjects in the real world, requires the development of approaches to processing and representing the state of actors in the model. A common solution is to display simulation results [1]. At the same time, the processes that led to this result remain hidden and require additional research, taking into account various restrictions. The decision maker's understanding and credibility of the simulation result can be diminished.

There are platforms that allow modeling with a graphical representation of the model process in the user interface (Table 1). The graphical user interface in such platforms often has a number of limitations and works inseparably from the platform itself. Because of this, it becomes a direct part of the platform and imposes a number of restrictions on its operation.

Table 1

Overview of Modeling Platforms with Administration System

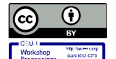
Platform Name	Base Map Editor	Dynamic change of modeling properties	Modularity	Displaying simulation results	Dynamic change of the modeling range
Ant Road Planner [2]	+	-	-	+	-
NetLogo [3]	+	-	-	+	+
AnyLogic [4]	+	-	-	+	-

The target component of this work is an application that is a client-server platform for modeling the movements and interactions of actors within a city map section [5]. The construction of the model

IMS 2021 - International Conference "Internet and Modern Society", June 24-26, 2021, St. Petersburg, Russia

EMAIL: attapi343@gmail.com (A. 1); sergey.burovic@gmail.com (A. 2); dparygin@gmail.com (A. 3); agurtyakov@gmail.com (A. 4); rashevsky.n@gmail.com (A. 5)

ORCID: 0000-0002-6733-8386 (A. 1); 0000-0001-7328-0953 (A. 2); 0000-0001-8834-5748 (A. 3); 0000-0002-8013-5778 (A. 4); 0000-0002-8076-4187 (A. 5)



© 2021 Copyright for this paper by its authors.
Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).
CEUR Workshop Proceedings (CEUR-WS.org)

is based on a multi-agent approach [6, 7]. The program expands the modeling functionality due to the modules plugged into it: it initializes them and displays the functionality on a graphic map of the city. The development of various modules is ongoing and the system is constantly being expanded.

Modules contain all the logic and modeling rules. Initially, the modules are not supplied as part of this software package, each of them is a separate class library. However, the modules are directly involved in the operation of the software package in the case of connecting one or more of them. For example, the module for managing data about objects on an online city map parses data from a file with OSM XML format and converts them to the necessary structure for storing in the list of objects of the main program and further using them by other modules.

The developed administration system is required to execute various modeling scenarios in real time with an extensible property list of the model itself and actors and plug-in components to answer the multiple "what if" question [8, 9].

In this regard, the purpose of the work is to develop a system for distributed administration of the behavior of actors and properties of the city model in real time. At the same time, as a key component of the concept of creating such a system, it was decided to focus on enabling the user to influence the course of modeling by changing the properties of models and actors in real time.

2. Problem statement

The existing application models the movement of actors on the basis of modules, however, it has a number of drawbacks and limitations, some of which must be eliminated in the created administration system.

The object of this research is the process of administration of modeling and obtaining results.

The subject of the research is the methods of distributed administration of the behavior of actors and properties of the city model in real time.

The business process diagram (“BPMN AS IS”) in the case using the application is shown in Figure 1.

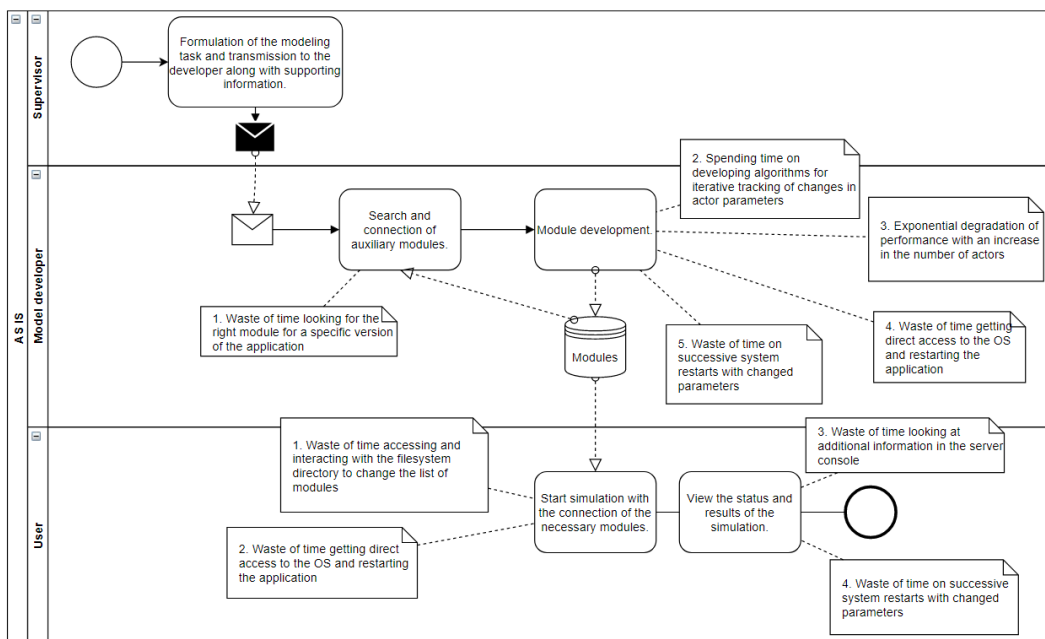


Figure 1: Application usage process (BPMN AS IS)

One of the problems of current modeling administration systems is the lack of approaches to processing and representing the state of actors in real time. The systems display the simulation result without the ability to pause the simulation or restart it during the current simulation with different modules and / or properties.

In some modeling systems, such as Ant Road Planner, it is not possible to dynamically change any properties of the model in general and actors in particular in real time after the initial setting of the modeling properties and the launch of the modeling. The simulation can be restarted on the changed properties only after the simulation has been worked out and the results are obtained.

In this regard, the task was set to develop a system of distributed administration of the behavior of actors and properties of the city model in real time, allowing the user to change the course of modeling by changing the properties of models and actors.

To implement the web client, the Angular framework [10] version 11 was chosen using the Ant Design user interface components [11].

3. Platform architecture

The platform architecture takes into account all the needs of the updated processes. The platform architecture is shown in Figure 2.

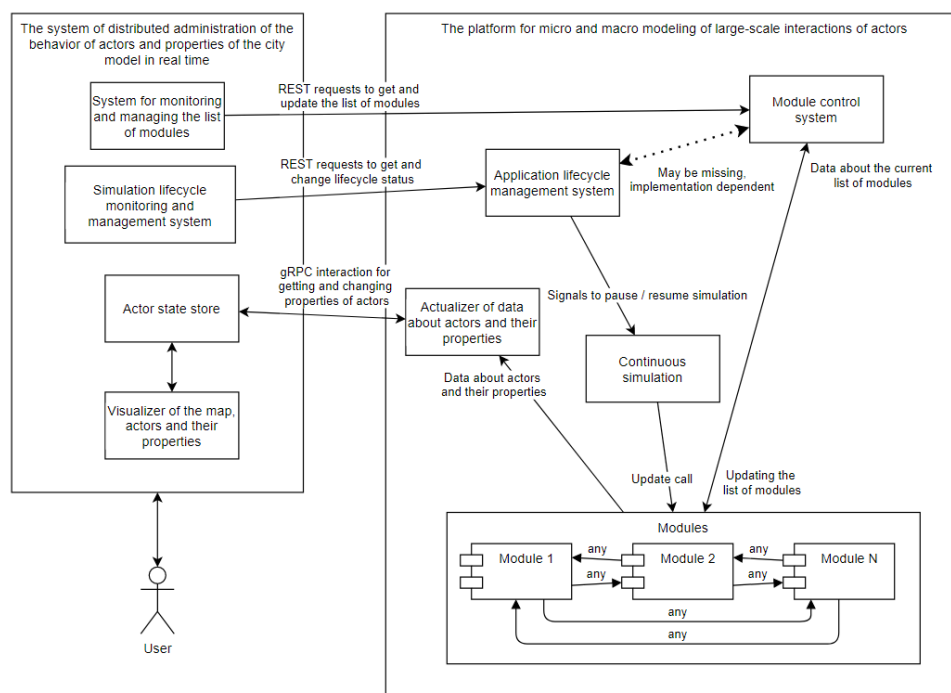


Figure 2: Platform architecture

4. Description of the methods used in the research

To solve the problem with the development of an approach to processing and representing the state of actors in real time, the proto3 [12] API of the modeling platform was analyzed, a part of which is shown in Figure 3.

Based on the API, there are several methods that will be needed to process and represent the state of the actors. One of these methods is to divide actors by their type (from the `type_full_name` field) into groups that represent layers. Splitting into layers opens up a number of possibilities, one of which is the ability to turn off the display of certain types of actors, if necessary.

Another important feature, as well as the next used method, opened by using layers, is the ability to apply specific styles to individual layers (from the `open_layers_style` field). This style is javascript code that must be executed before an object of the `Style` class is obtained.

```

1  syntax = "proto3";
2
3  option csharp_namespace = "OSMLS.Map";
4
5  package map;
6
7  import "google/protobuf/empty.proto";
8
9  service MapService {
10     rpc GetMapFeaturesMetadata (google.protobuf.Empty) returns (stream MapFeaturesMetadata);
11     rpc GetMapFeaturesMetadataUpdates (google.protobuf.Empty) returns (stream MapFeaturesMetadata);
12
13     rpc GetMapFeatures (google.protobuf.Empty) returns (stream MapFeature);
14     rpc GetMapFeaturesUpdates (google.protobuf.Empty) returns (stream MapFeature);
15     rpc GetRemoveMapFeatureEventsUpdates (google.protobuf.Empty) returns (stream RemoveMapFeatureEvent);
16
17     rpc GetMapFeaturesObservableProperties (google.protobuf.Empty) returns (stream MapFeatureObservableProperty);
18     rpc GetMapFeaturesObservablePropertiesUpdates (google.protobuf.Empty) returns (stream MapFeatureObservableProperty);
19     rpc SetMapFeatureObservableProperty (MapFeatureObservableProperty) returns (google.protobuf.Empty);
20 }

```

Figure 3: proto3 modeling platform API

The method for displaying actors in specific coordinates should be based on the GeoJSON format [13], since data in this format can be obtained from the `geo_json` field. Because methods provided by the OpenLayers library [14] are already used to solve most of the other subtasks; methods [15] from this library, aimed at working with this format, can also be used to work with the GeoJSON format.

To solve the problem with the development of an approach to representing and changing the properties of the model in general and actors in particular in real time, a number of methods from the REST API of the modeling platform can be used. For example, the State group of methods allows the user to manage the life cycle of a simulation, which can be started, temporarily paused, and stopped. The Assemblies method provides the ability to add new assemblies with modules and module dependencies to the application, and the Modules method group allows the user to get the entire list of modules obtained from the added assemblies, and also provides the ability to manage the list of modules participating in the modeling process (specific modules from the general list can be activate and deactivate for the model). In addition, there are a number of methods for working with the list of properties of specific actors.

Thus, a brief description of the methods that will be used in the research when creating a targeted method for distributed administration of the behavior of actors and properties of the city model in real time is given.

5. Implementation of the basic part of the administration system

5.1. Generation of REST API and gRPC-web infrastructure

It was customary to add the entire generated infrastructure along the `src/app/generated` path, so this path was added to the standard `.gitignore` file.

The modeling platform route `/swagger/v1/swagger.json` is called to get the `swagger.json` file used for generation. This route is created automatically using the `Swashbuckle.AspNetCore` library. Then this file is added to the root of the project.

The npm module `ng-openapi-gen` is used to generate the API infrastructure.

To generate the javascript gRPC-web infrastructure [16], `protoc` is installed with the ability to access it by the appropriate command. After that, the npm module `ts-protoc-gen` was added, which allows converting the javascript generated using `protoc` into typescript code. Modules have also been added to support gRPC-web and `google-protobuf`.

After that, a script is created in the `package.json` file to generate the infrastructure based on `map.proto`;

The generated client does not require DI and can be obtained directly from the `grpc` namespace.

5.2. Assembly management implementation

Assemblies are managed in `AssemblyCompositorComponent`, a child of `AssemblyComponent`. The `NzUploadModule` component and `AssembliesService` are used to upload files.

The view of the final component is shown in Figure 4. When the component button is pressed, a file system window opens with a proposal to select one or more assembly files. After selection, assemblies are immediately uploaded to the server.

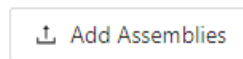


Figure 4: View of the "Add Assemblies" component

5.3. Module management implementation

Modules are managed in the `ModuleManagerComponent`, a child of the `ModuleComponent`. For this, `ModulesService` is used.

The view of the resulting component is shown in Figure 5. Each of the component buttons can be active or inactive, depending on the current state of the model.

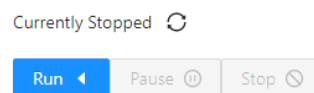


Figure 5: Button view when the simulation is stopped

5.4. Model state management implementation

Model state is managed in `ModelStateManagerComponent`, a child of `ModelComponent`. The `StateService` is used for this.

The view of the resulting component is shown in Figure 6. This component supports multiple choice of modules for the model.

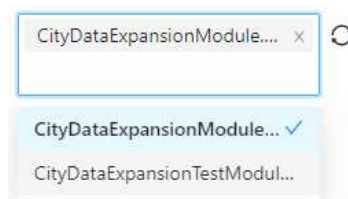


Figure 6: Component with selected module

After selecting or deselecting one of the modules, the changes are immediately sent to the server.

6. Receiving and processing of map data

6.1. Metadata handling

Getting the current metadata occurs when the map component is initialized. To get metadata, the contract method `GetMapFeaturesMetadata` is used, which returns a stream of objects of type `MapFeaturesMetadata`.

Each of the obtained objects is transferred both to the component of the map browser (to create a new layer [17], on which the actors will be placed in the future), and to the component of the map properties (to create a new table with actors of this type).

The map browser component uses the type (`type_full_name`) as the name of the layer to create and the style (`open_layers_style`) as the layer style. This component does not work with observable properties.

The map properties component uses the type (`type_full_name`) to classify actors into various tables. The list of observable properties is used by this component to create the table infrastructure. So, if the “editable” flag of the observed property is “true”, then the corresponding cell in the table can be edited. The value type (`value_type`) is used for correct conversion from gRPC types to javascript / typescript types and vice versa.

After the method for obtaining metadata has finished its work, subscribes to the method for updating the metadata `GetMapFeaturesMetadataUpdates`, which returns objects of the same type, which are processed in the same way. This subscription exists until one of the applications (modeling platform or administration system) stops working.

6.2. Actors handling

Getting the current list of actors occurs after getting the list of metadata. This is necessary in order for the resulting actors to be correctly placed on a previously created map layer. The list of actors is obtained using the `GetMapFeatures` method, after which a subscription to updates is performed using the `GetMapFeaturesUpdates` method (using an approach similar to working with metadata). Both methods return an object of type `MapFeature`.

The resulting object from any of the methods is sent to the map browser component, where it is converted from GeoJson (`geo_json`) to the OpenLayers library format, from which the actor ID is obtained. An actor obtained from the GeoJson format is placed on a layer that has the corresponding type, the same as the type of the resulting actor (`type_full_name`). If an actor with such an identifier already exists in the corresponding layer, then it is previously removed from the layer.

After the initial receipt of the actors, there is also a subscription to the `GetRemoveMapFeatureEventsUpdates` method, which provides a stream of events for removing actors, objects of type `RemoveMapFeatureEvent`.

When objects are received from this stream, objects with parameters corresponding to the type (`type_full_name`) and identifier (`id`) of the received event are removed from both the map browser component and the map properties component.

6.3. Actors observable properties handling

The retrieval of the current list of the observable properties of the actors occurs after the retrieval of the list of metadata. The retrieval of the current list of the watched properties of the actors occurs after the retrieval of the list of metadata. The list of actor observable properties is obtained using the `GetMapFeaturesObservableProperties` method, after which a subscription to updates is performed using the `GetMapFeaturesObservablePropertiesUpdates` method (using an approach similar to working with metadata and with actors). Both methods return an object of type `MapFeatureObservableProperty`.

After receiving this object, it is added to the map properties component, in which it is displayed in the table, in accordance with its metadata. The type of the observable property (`type_full_name`) affects the table in which the object will be placed. The identifier (`id`) groups several different properties by one actor, the verbal title of the property (`title`) defines the column in the table, and the value (`value`) defines the current value of the column.

In addition to displaying the properties of actors, there is also the ability to edit them. After finishing editing the cell that has the corresponding property, the updated value is sent to the modeling platform, to the `SetMapFeatureObservableProperty` method, as a `MapFeatureObservableProperty` object.

7. Testing the administration system

7.1. General information about testing

To test the results, a modeling platform with a connected test module is used.

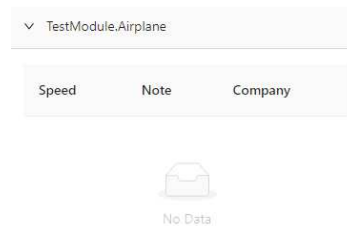
The test module adds several actors to the map:

1. Point (PointActor)
2. Line string (LineStringActor)
3. Polygon (PolygonActor)
4. Airplane 1 with an initial speed of 1000 by "Boeing" (Airplane, actor class created in the module)
5. Airplane 2 with an initial speed of 0 by "Airbus" (Airplane, an actor class created in the module)

The Airplane class inherits from the PointActor class and has several additional observable properties.

7.2. Displaying metadata

Metadata is displayed regardless of the status of the modeling process; it is always displayed in the map properties component for all types of actors that have at least one observable property. Figure 8 shows the mapping of actor property metadata for Airplane type actors. Since the modeling process is stopped and there are no actors at the moment, the table has only columns indicating the properties being viewed and has no rows.



Speed	Note	Company
No Data		

Figure 8: Displaying property metadata for Airplane type actors

The metadata responsible for displaying styles can be seen when displaying actors in the map browser component (see Figure 9), displaying occurs only when the modeling process is running or paused, since only at this moment there can be actors on the map.

There are two Airplane type actors on the map (lower left corner), displayed as small purple dots, a line string actor (in the center of the map), a point actor (at one of the ends of the line string actor), and a polygon actor (upper left corner). Of all the listed objects, only Airplane objects have a changed style.

7.3. Displaying actors

Actors are displayed both when the administration system is connected to the modeling platform with simulation already running, and when modeling is started from the administration system with the same result.

At the same time, updates to actors are correctly displayed by the administration system over time (see Figure 10).

When the modeling process is stopped, the Map Browser component is cleared of actors.

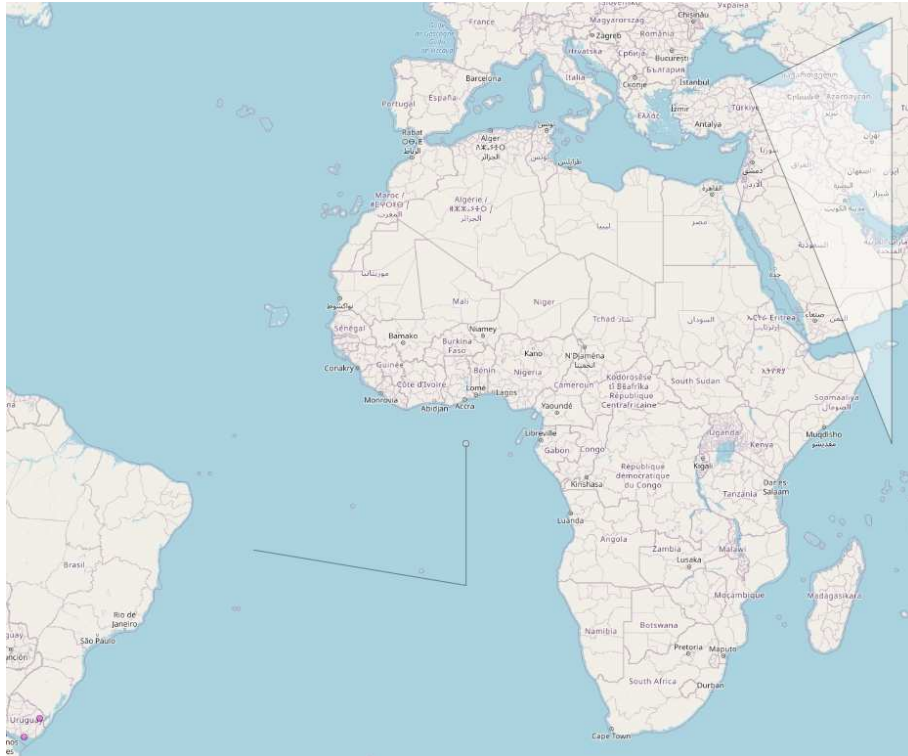


Figure 9: Displaying Actors in the Map Browser Component



Figure 10: Displaying a moving actor at timestamps 1 and 2

7.4. Displaying observable properties

When the simulation is run, the map browser component displays the browseable properties according to the table layout and metadata (see Figure 11, a).

TestModule.Airplane			TestModule.Airplane		
Speed	Note	Company	Speed	Note	Company
1000		Boeing	1000.1		Boeing
0		Airbus	0		Airbus

< 1 >

< 1 >

Figure 11: Displaying observable properties of actors with Airplane class: a. Current property state; b. With one property changed

If the observable properties on the modeling platform change (for example, the speed of one of the Airplanes increases over time), the changes will be reflected in the administration system, without the need to reload the page (see Figure 11, b).

7.5. Editing observable properties

When the cursor is hovering over a row that contains editable observable properties, the editable property cells will be highlighted (see Figure 12, a). Clicking on one of these cells will open the editing element (see Figure 12, b)

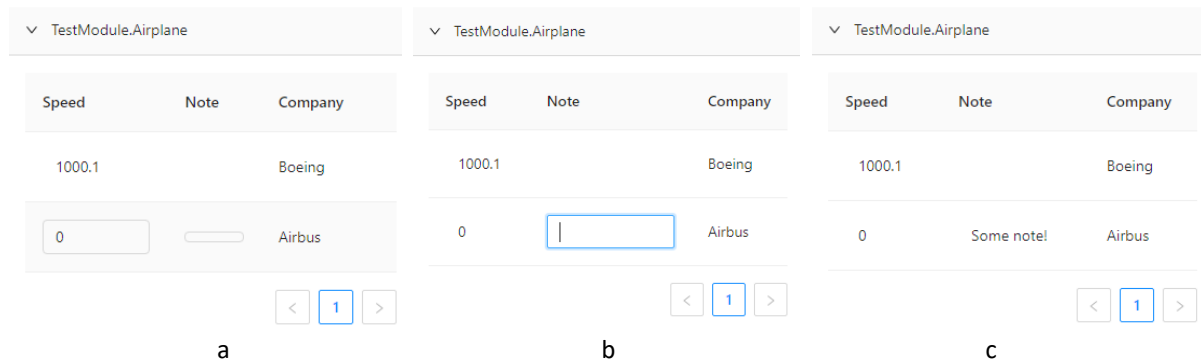


Figure 12: Interacting with editable properties: a. Howering editable properties cells; b. Opening a cell edit element; c. Displaying a property with a changed value

If the property value is changed and the edit dialog is closed, this property will be changed on the modeling platform, as well as in the current and all other connected administration systems (see Figure 12, c).

8. Conclusion

To interact with the platform for modeling the movements and interactions of actors within the city map section, a web client was developed that plays the role of an administration system and provides:

- Custom display of real-time actors with specific styles based on actor types
- Ability for the user to manage the list of model modules
- Ability to the user to manage the life cycle of the simulation
- The ability for the user to influence the course of modeling by changing the properties of the model and actors

Compared to existing solutions, this administration system is distinguished by the ability to work with a specific modeling platform, which in turn offers a number of advantages over other modeling solutions, such as:

- Cross-platform
- Open source
- Extensibility of functionality using modules

Thus, the developed administration system makes the modeling platform more accessible for interaction with the end user by providing a graphical user interface instead of a software one (REST and gRPC API).

9. Acknowledgements

This work has been supported by the Russian Science Foundation (RSF) grant (project No. 20-71-10087). The authors express gratitude to colleagues from the Urban Computing Laboratory (UCLab)

and the Department of Digital Technologies for Urban Studies, Architecture and Civil Engineering, VSTU involved in the development of Live.UrbanBasis.com project.

10. References

- [1] A. Davtian, O. Shabalina, N. Sadovnikova, D. Parygin, Cyber-Social System as a Model of Narrative Management, *Studies in Systems, Decision and Control* 333, Springer, 2021, pp. 3–14. doi: 10.1007/978-3-030-63563-3_1.
- [2] Ant Road Planner.Ru, Pedestrian simulator, 2021. URL: <https://antroadplanner.ru/>.
- [3] NetLogoWeb.Org, NetLogo is a programming language and integrated development environment (IDE) for agent-based modeling, 2021. URL: <https://www.netlogoweb.org/>.
- [4] AnyLogic.Ru, AnyLogic is a multimethod simulation modeling tool developed by The AnyLogic Company, 2021. URL: <https://www.anylogic.ru/>.
- [5] D. Parygin, A. Usov, S. Burov, N. Sadovnikova, P. Ostroukhov, A. Pyannikova, Multi-agent Approach to Modeling the Dynamics of Urban Processes (on the Example of Urban Movements), *Communications in Computer and Information Science* 1135, Springer, 2020, pp. 243–257. doi: 10.1007/978-3-030-39296-3_18.
- [6] A. Anokhin, S. Burov, D. Parygin, V. Rent, N. Sadovnikova, A. Finogeev, Development of Scenarios for Modeling the Behavior of People in an Urban Environment, *Studies in Systems, Decision and Control* 333, Springer, 2021, pp. 103–114. doi: 10.1007/978-3-030-63563-3_9.
- [7] D. Parygin, Rebalancing Cycle of Ensuring Needs for an Exoactive Management System, in: *Proceedings of the 2020 International Multi-Conference on Industrial Engineering and Modern Technologies, FarEastCon 2020, Vladivostok, Russia, IEEE, 2020*, art. no. 9271512. URL: <https://ieeexplore.ieee.org/document/9271512>. doi: 10.1109/FarEastCon50210.2020.9271512.
- [8] Yoav Shoham, Rob Powers, Trond Grenager, If multi-agent learning is the answer, what is the question?, *Artificial Intelligence* 171(7), 2007, pp. 365–377. doi: 10.1016/j.artint.2006.02.006.
- [9] Saehwa Kim, Seongsoo Hong, Naehyuck Chang, Scenario-based implementation architecture for real-time object-oriented models, in: *Proceedings of the Seventh IEEE International Workshop on Object-Oriented Real-Time Dependable Systems, WORDS 2002, 2002*, pp. 147–152. doi: 10.1109/WORDS.2002.1000047.
- [10] Angular.Io, Introduction to the Angular Docs, 2021. URL: <https://angular.io/docs>.
- [11] Ant Design of Angular, An enterprise-class Angular UI component library based on Ant Design, 2021. URL: <https://ng.ant.design/docs/introduce/en>.
- [12] Developers.Google.Com, Language Guide (proto3), 2021. URL: <https://developers.google.com/protocol-buffers/docs/proto3>.
- [13] Datatracker.Ietf.Org, The GeoJSON Format, 2021. URL: <https://datatracker.ietf.org/doc/html/rfc7946>.
- [14] OpenLayers.Org, OpenLayers is an open-source JavaScript library for displaying map data in web browsers as slippy maps, 2021. URL: <https://openlayers.org/>.
- [15] OpenLayers.Org, ol/format/GeoJSON~GeoJSON, 2021. URL: https://openlayers.org/en/latest/apidoc/module-ol_format_GeoJSON-GeoJSON.html.
- [16] Anthonygiretti.Com, Create gRPC-web app with Angular 8 on Windows, 2021. URL: <https://anthonygiretti.com/2020/03/29/grpc-asp-net-core-3-1-how-to-create-a-grpc-web-client-examples-with-angular-8-and-httpclient/>.
- [17] OpenLayers.Org, ol/layer/Layer~Layer, 2021. URL: https://openlayers.org/en/latest/apidoc/module-ol_layer_Layer-Layer.html.