The architecture of the software tool for the agent-based simulation models specification development*

A.I. Pavlov[1 [0000−0002−7753−7514]], A.B. Stolbov[1 [0000−0001−6513−7030]], and A.S. Dorofeev[2 [0000−0002−8498−3301]]

[1] Matrosov Institute for System Dynamics and Control Theory of Siberian Branch of the Russian Academy of Sciences, 134 Lermotov st., Irkutsk, Russia
{asd,stolboff}@icc.ru
http://http://idstu.irk.ru
[2] Irkutsk National Research Technical University, 83 Lermotov st., Irkutsk, Russia
dorbaik@istu.edu
https://www.istu.edu

Abstract. The architecture and a set of components of the software tool providing the development of agent-based simulation models (ABSMs) specifications are considered in the paper. The underlying of the proposed tool is the author's methodology which specializes the well-known MDD (Model-Driven Development) approach to the ABSM development process. The components are implemented as interacting web-oriented applications utilizing modern standards of communications (HTTPS, SOAP, WSS). The third-party software (Madkit, Jade, Drools, Jboss) was also adapted for supporting component functionality.

Keywords: agent-based simulation models · Model-Driven Development · web-oriented software.

1 Introduction

The overall contemporary level of software and hardware development has significantly enhanced in recent years and the practical relevance of many areas of artificial intelligence and system analysis has thereby increased. The accumulation of large amounts of data, together with the high-performance processing capabilities, has led to the emergence and active development of research areas such as data mining, machine learning, and simulation. In the current situation, a multi-agent approach has great prospects in carrying out simulations. Recent authors studies contribute to developing methods and tools that provide non-programming users with the ability to create and use agent-based simulation models (ABSMs) in various subject areas. The relevance of this direction is due to the need to expand the scope of application of ABSM and the intensity of their use in solving applied problems. The limiting factor affecting the popularity

* Supported by RFBR 18-07-01164, 18-08-00560

of the multi-agent modeling paradigm is the high entry threshold for mastering ABSM technologies: lack of clear, uniform and well-established methodological approaches and guidelines; a variety of languages for representing ABSM; a large number of software libraries and tool platforms for implementing ABSM.

One of the ways to solve the mentioned problem related to the spread rate of the simulation modeling approaches in general, and ABSM in particular, in the practice of solving applied problems is to create problem-oriented systems based on general-purpose tools (for example, anyLogistix is a tool for designing, optimizing and analyzing supply chains from developers of a multi-paradigm general platform AnyLogic). Another approach is the development of the tools, the architecture of which would originally include methods and tools for designing and implementing ABSM, paying attention to the low level of user skills in programming and modeling, i.e. it would provide an opportunity for domain experts to create applied ABSMs independently (in an ideal case) or with minimal participation of specialists in simulation modeling.

2 The application of MDD approach to the process of ABSMs development

The rich scientific and applied experience accumulated to date in the creation and application of ABSM allows one to proceed to solve the problems of designing and implementing ABSM at a higher level, abstracting from specific means by unifying existing methods and software development systems. It is proposed to use a model-driven approach, MDD (Model-Driven Development) [1], as a methodological basis, according to which all the information needed to develop ABSM is formalized in the form of a hierarchical set of models, and the specifications of a specific applied ABSM are obtained as a result of transformation of elements of this set. A more detailed description of the approach, a system of models, and transformation rules are presented in a series of publications by the authors [2–5].

Currently, the number of models at all levels of the hierarchy reaches 15. At the same time, research is carrying out to create new and detail existing models. Further, as part of a brief description of the capabilities of the MDD approach, we describe the most significant models. The top-level metametamodel M^{Ont} defines the most abstract elements for the specification of lower-level metamodels. The M^{Ont} is defined as a well-known ontological form: concept - attribute - relation:

$$M^{Ont} =< Concept, Attribute, Relation, Instance >,$$
$$< Concept >=< Name >< Description > < Attribute >,$$
$$< Attribute >=< Name >< Attributetype >< Defaultvalue >,$$
$$< Attributetype >= Literal|Concept,$$
$$< Relation >=< Concept >< Name >< Relationtype >< Concept >,$$
$$< Relationtype >= Literal.$$

Metamodel M^A is a domain-independent, agent-related metamodel for describing the architectural elements of a certain ABSM development methodology, including both structural and behavioral aspects. In this case, the corresponding metamodels of the implementation tools are used, which describe, for example, the interfaces of the simulation tool, the inference engine, the imperative engine for invoking external procedures, etc. Models M^{Ont-D} and M^{KB-D} are respectively the ontology of the domain area under consideration and the knowledge base that describes the meaningful domain-specific behavior. Model M^{A-D} – an overall specification of a specific ABSM for the domain area under consideration – includes both information from M_A, and M^{Ont-D} , M^{KB-D}. The M^{A-D-I} model is about the specificity of the initial conditions of a particular ABSM. The *codI* structure contains results of the M^{A-D} simulation execution in some ABSM specifications interpreter with particular initial conditions taken from M^{A-D-I}.

Thus, in terms of the models, the process of ABSM developing from the non-programming user point of view can be defined by the following sequence:

1. Creation of a conceptual model for the domain area under consideration (M^{Ont-D}).
2. Development of the M^{KB-D} knowledge base, including rules and fact templates, created on the basis of concepts, attributes and relationships from the M^{Ont-D} model.
3. Selection of the appropriate ABSM architecture described in the M^A metamodel. This metamodel is developed by experts (system analysts, model designers) and related implementation details, if necessary, can be hidden from a non-programming user.
4. Setting the correspondence between the elements of the M^{Ont-D} and M^A, i.e. integrating ABSM into the domain.
5. Clarification of M^{KB-D} with information related to ABSM (possible, but not mandatory).
6. Setting the parameters of the computational experiment (creation/clarification of the M^{A-D-I} model).
7. Performing the simulation.
8. Representation of the simulation results.

In the current article, the authors present the results associated with stages 1-6, i.e. the formation of specific ABSM specification. The issues related to stages 7 and 8 are not considered in this paper. An approach to related solutions can be found in publications [2–8].

3 The software architecture description

The software tool that implements the proposed approach for the formation of agent-based simulation models is a natural evolution of the author's previous projects: a prototype support system for designing agents [7, 8] and the software platform for creating knowledge-based systems [4]. Thus, experience in the

development of a typical agent allowed us to proceed to the creation of the basic ABSM-related component models. The former imperative block of a typical agent is currently described in more general terms using models of the simulation engine and the run-time controller. The former declarative part of a typical agent transforms into the inference engine model. When implementing a software tool, these models are the basis for creating appropriate software interfaces.

At the current stage of the software tool development the creation of a new version of architecture (see Fig. 1) pursued the following goals:

1. Implementation of the technical feasibility of forming models and metamodels and their transformations during the development of ABSM in accordance with the original methodology based on the MDD approach.
2. Reuse of existing software developed by the authors.
3. Providing the possibility of utilizing the currently existing third-party libraries and tool platforms at the stage of interpreting ABSM specifications.
4. Orientation on modern software development approaches, including the active use of new technologies for network data exchange (Websocket), the latest versions of the DBMS and user interface implementation libraries.

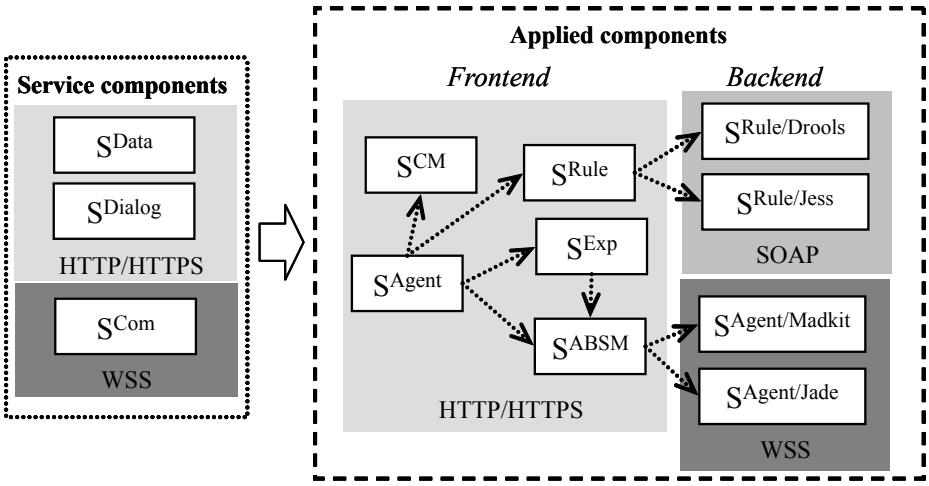

Fig. 1. The software tool architecture

At the moment, the list of components of the software tool is as follows: data management component – S^{Data}, component for conceptual modeling – S^{CM}, component for the reasoning based on the rules – S^{Rule}, component for the organization of two-way data exchange – S^{Com}, component for the dialogue interaction with the user – S^{Dialog}, an entry point of the tool for the user – S^{Agent} component used for direct management of the ABSM specification creation, component for the computational experiment management - S^{Exp}, component

for the simulation implementation (specification interpretation) - S^{ABSM}. At the same time, some of these components are service ones and are used to implement the functionality of other (applied) components.

The basic functionality of the S^{Data}, S^{CM}, S^{Rule}, S^{Com}, S^{Dialog} components is based on the capabilities of the authors platform for creating knowledge-based systems [4]. Let us consider in more detail the changes that are proposed to be made to existing components.

4 The software components description

The S^{CM} component ensures the creation of a conceptual model (CM) for the selected domain area with a given level of details [4]. For the current version of the tool, it is proposed to expand the functionality of the S^{CM} with the possibility of establishing a connection between two CMs, in which the first CM will be interpreted as an abstract model, and the second as its implementation. Supporting the connections between the concepts of different conceptual models will provide the possibility of implementing the double inheritance mechanism when a certain concept inherits the structure not only from the basic concept within its CM, but also from the concept of the associated CM. The main advantage of this innovation is the possibility of obtaining several variants of the agent-related, subject-dependent description of the ABSM structure for one CM domain.

The S^{Rule} component provides the creation of rule-based knowledge bases with the CM of the subject area as a data source, as well as the visual construction of the rules, the formation of initial conditions, code generation and execution of reasoning [4]. In addition to the existing S^{Rule} functionality that uses JESS [9] to process rules, the support for the new environment [10] was provided, with the Drools code generation implemented on the basis of the mvel dialect.

The component of two-way data exchange (S^{Com}) is introduced to provide the possibility of fast, asynchronous data exchange between users and/or components of the tool, as well as to enable the option of starting data exchange by the server's initiative. For example, to organize a request for additional information from the user in the process of inference. The S^{Com} component is proposed to be developed on the basis of one of the existing libraries for working with the WebSocket protocol, for example, Node.js WS.

The S^{Dialog} component contains a standard set of graphical controls, firstly, providing support for the most common simple data types: single/multi-line input fields, date and time representations, a checkbox for working with a logical type. Secondly, more complex elements for displaying data in tabular form are implemented for S^{Dialog}, including nested tables, selecting one/several options from the list, and displaying data in the form of a network.

The following components are developed as new problem-oriented based on the functionality of the components described above. So, the S^{Agent} component is a key in terms of ensuring the organization of the ABSM specification devel-

opment process and provides a unified user terminal through which interaction with other components is carried out. The main tasks of the S^{Agent} include the following: control the sequence of stages for designing the ABSM specification; storage of all necessary information about the state of the design process, support for model transformations according to the used methodology [2–5]; adapting the functionality of other components to the current context and calling the appropriate methods. The current implementation of the S^{Agent} user terminal is made in fairly general terms related to the field of multi-agent systems and simulation. However, the component architecture allows, if necessary, the development of domain- and problem-oriented modifications of the S^{Agent}, replacing specific concepts of the ABSM paradigm with terms more appropriate for the end-user.

The component for computational experiment management (S^{Exp}) allows you to create the initial conditions of ABSM and save the simulation results, as well as directly control the simulation process: start, stop and/or pause the ABSM, dynamically change the ABSM parameters (number and duration of simulation steps, manual generating new ABSM elements, editing or deleting them).

The responsibilities of the S^{ABSM} component include the interpretation of ABSM specifications. To this end, S^{ABSM} uses the capabilities of the existing Madkit multi-agent modeling support library [11] and the platform for creating multi-agent Java Agent DEvelopment framework (Jade) systems [12], for which software shells using the facade design pattern have been developed. In the process of interpretation, the S^{ABSM} core provides a call to the methods of these third-party means required in accordance with the specific ABSM specification.

5 Software implementation features

The software package is designed as a web application using the thin client technology, within which the components of the tool are hosted on a server, and a standard web browser acts as a user terminal. As a part of a typical component of a software tool, we can distinguish the client part, which contains the implementation of the user interface, and the server part, where the implementation of data processing methods that constitute the components functionality is located. In the case when a component uses existing software, libraries or services to implement its functionality, there are two sets of modules in its server-side: one available to the users and another with limited access. For example, in S^{Rule} modules for designing knowledge bases, code generation, and logical inference are available to the users, whereas access to inference engines of Jess and Drools have only S^{Rule} modules. Thus, on the server side of the software tool, it is possible to distinguish both the external (frontend) and internal (backend) parts (see Fig. 1).

In the process of organizing the interaction between the external and internal parts of the server for calling methods and receiving (sending) data, it is proposed to use the following set of protocols: HTTP, SOAP, and WebSocket. This kit

provides the ability to utilize a significant portion of existing software systems and libraries.

To implement the user interface of components, a well-established approach to creating interactive web pages using the languages of HTML, CSS, JavaScript and popular libraries (jQuery, jQueryUI, jQueryGrid, jsPlumb) was used. The data exchange process is organized both on the basis of client-initiated requests via the HTTPS protocol, and using bidirectional channels of the WebSocket protocol. Currently, the server part includes two HTTP servers: based on Apache and Node.js, which together with the Node.js WSS server form the external part of the server (see Fig. 1). HTTP-Apache server provides access to the functionality of the following components: S^{Data}, S^{CM}, and partially to the functionality of S^{Rule} (designing knowledge bases and code generation). Node.js HTTP server provides a solution to the tasks of authorizing S^{Com} clients, and the Node.js WSS server provides S^{Com} with the possibility of organizing bidirectional data exchange, in which each client is able to interact with others in an arbitrary order.

6 Future works

As one of the directions of the future works let us describe the possible implementation of the proposed software tool and related ABSM development approach into educational process for project-based learning (PBL). The main features of PBL pedagogy model are the following: long-term study, multidisciplinary, student-oriented, focusing on problems with practical relevance (from understanding the phenomenon to possible application in real life). The students design projects to learn as much as possible about the topic under consideration, rather than looking for the right answers to questions asked by a teacher. So, theoretically, the final result can be obtained from different sources via a long-lasting iterative process throughout the studying period.

From a methodological point of view, the multi-agent modeling paradigm provides the appropriate facilities to meet the requirements of PBL. MAS and ABSM are especially valuable in situations where the output of the work (and it is the project) can be composed of heterogeneous elements with consistent refinement. So, on the one hand, the project itself can be represented in terms of MAS/ABSM. On the other hand, the project can be a part of another higher-order MAS project. Thus on the macro and micro levels due to its MAS nature, the evolution, inclusion, and integration of the project can be done regarding MAS principles.

The suggested author's approach allows the explicit separation domain-specific and agent-related parts of ABSM, so there is no predefined development methodology and build in software means. This feature gives an additional freedom degree increasing the variety of methods and tools to study and apply in a project. The application of the considered in the paper software tool for supporting the process of project-based learning gives us the following preliminary structure of

the typical educational project (TEP):

$$< TEP >=< Project - description >, < Educational - resources >,$$

$$< MAS - method >, < MAS - software >, < Domain - model >,$$

where the last three elements have supported by existing functionality of the proposed system, whereas the first two are under construction.

From a practical perspective, the project topics can be related to the digital economy, IoT, AI that nowadays in need of qualified specialists. And here the objects under consideration and research tool also would have a common nature. In that case, Irkutsk National Research Technical University campus can be chosen as a sandbox for testing the vitality and adequacy of the projects.

7 Conclusions

The architectural solutions proposed in the article make it possible to implement the original methodology developed with the participation of the authors to the process of creating agent-based simulation models based on the specialization of the MDD approach. The application of this methodology allows us to solve the problem of expanding the areas in which agent-based simulation models can be utilized by supporting non-programming users at all phases of the model development process, from the conceptualization stage the automated creation of an executable model.

The component organization of the software tool architecture provides the capabilities of reuse, modification, and refinement of previously obtained results regarding knowledge-based systems and agent modeling support systems. The article provides a brief description of the main components of the tool and the features of its software implementation. The structure of a typical component in the context of the client-server interaction model is considered. The future plans concerning introducing the proposed tool into the educational process are discussed.

The current research is partially supported by RFBR (18-07-01164, 18-08-00560).

References

1. France, R., Rumpe, B.: Model-Driven Development of Complex Software: A Research Roadmap. In: Briand, L.C., Wolf, A.L. (eds.) Proceedings of the International Conference "Future of Software Engineering", pp. 37–54, IEEE, Minneapolis MN (2007).
2. Nikolaychuk, O.A., Pavlov, A.I., Stolbov, A.B.: Web-Oriented Software System for Agent-Based Modeling Driven by Declarative Specification of Implementation Process. In: Proceedings of the 3rd Russian-Pacific Conference on Computer Technology and Applications (RPC), pp. 1–5. IEEE, Vladivostok Russia (2018). https://doi.org/10.1109/RPC.2018.8482149

3. Nikolaychuk, O.A., Pavlov, A.I., Stolbov, A.B.: Models and software for agent-based model development based on model-driven approach. In: Bychkov, I.V., Karastoyanov D., Arshinsky, L.V. (eds.) Proceedings for First Scientific-practical Workshop Information Technologies: Algorithms, Models, Systems (ITAMS), CEUR Workshop Proceedings, vol. 2221, pp. 13–19. CEUR-WS.org, Irkutsk Russia (2018).
4. Nikolaychuk O.A., Pavlov A.I., Stolbov A.B.: The software platform architecture for the component-oriented development of knowledge-based systems. In: Proceedings of the 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), pp. 1234–1239. IEEE, Opatija Croatia (2018). https://doi.org/10.23919/MIPRO.2018.8400194
5. Nikolaychuk, O.A., Pavlov, A.I., Stolbov, A.B.: The Agent-Based Modeling Method For The Study Of Unique Mechanical Systems. In: Shakhmametova, G., Mironov, K., Galimova, L. (eds.) Proceedings of the 7th Scientific Conference on Information Technologies for Intelligent Decision Making Support (ITIDS 2019), Advances in Intelligent Systems Research, vol. 166, pp. 201–206. Atlantis Press, Ufa Russia (2019).
6. Korshunov, S.A., Pavlov, A.I., Nikolaychuk, O.A.: Concept of simulation visualization software based on ontology approach. Scientific Visualization **8**(2), 332-343 (2016).
7. Pavlov, A. I., Stolbov, A.B.: Architecture of agents design support system for complex systems simulation models. Software & Systems **109**(1), 12-16 (2015).
8. Pavlov, A.I., Stolbov, A.B.: A prototype of an agents design support system for complex system simulation models. Software & Systems **29**(3), 79-84 (2016).
9. Friedman-Hill, E.: Jess in Action: Java Rule-Based Systems. Manning Publications Co., Greenwich (2003).
10. Salatino, ., Maio, M., Alivertie, E.: Mastering JBoss Drools 6. Packt Publishing Ltd., Birmingham (2016).
11. Gutknecht, O., Ferber, J. The madkit agent platform architecture. In: Wagner, T., Rana, O.F. (eds.) Infrastructure for Agents, Multi-Agent Systems, and Scalable Multi-Agent Systems, LNCS, vol. 1887, pp. 48–55. Springer, Berlin Heidelberg (2000).
12. Bellifemine, F.L., Caire, G., Greenwood, D.: Developing multi-agent systems with JADE. Wiley Series, NJ Wiley (2007).