Modeling & Simulation as a Service with the Massive Multi-Agent System MARS

Christian Hüning
Dept. of Computer Science
HAW Hamburg, Germany
christian.huening@haw-hamburg.de

Mitja Adebahr
Dept. of Computer Science
HAW Hamburg, Germany
mitja.adebahr@haw-hamburg.de

Thomas Thiel-Clemen
Dept. of Computer Science
HAW Hamburg, Germany
thomas.thiel-clemen@haw-hamburg.de

Jan Dalski
Dept. of Computer Science
HAW Hamburg, Germany
jan.dalski@haw-hamburg.de

Ulfia Lenfers
Dept. of Computer Science
HAW Hamburg, Germany
ulfia.lenfers@haw-hamburg.de

Lukas Grundmann
Dept. of Computer Science
HAW Hamburg, Germany
lukas.grundmann@haw-hamburg.de

ADDITIONAL AUTHORS

1. Janus Dybulla, Department of Computer Science, HAW Hamburg, Germany, E-Mail: janus.dybulla@haw-hamburg.de
2. Gregory A. Kiker, Associate Professor, Agricultural and Biological Engineering Dept., University of Florida, P.O. Box 110570, Gainesville, FL 32611-0570, E-Mail: gkiker@ufl.edu

ABSTRACT

There is an increasing demand for very large-scale agent-based models. High numbers of individual entities and complex interactions between them require new ways of modeling and simulation. The creation of a distributed simulation model imposes a major challenge in the fields of network communication and coordination to the developer. Integrating multi-scale GIS and time-series data into such a model is another challenge altogether.

We introduce the massive Multi-Agent Research and Simulation system (MARS). It is designed to provide a Modeling and Simulation as a Service (MSaaS) solution to end users. MARS allows domain experts to integrate their data and models through a user-friendly web interface. The complexity of distributed and scalable simulation is handled in the background by a mechanism we call Agent Shadowing. Finally a layer-based segmentation of the model is proposed. It allows domain specialists to focus on one aspect at a time while developing their simulation model.

A large-scale model from the ecological modeling domain is showcased. The model integrates various GIS data formats with collected time-series datasets and simulates a scalable amount of agents. Results from this simulation demonstrate the capabilities of MARS to support the workflow as required by the development of large-scale agent-based simulation models.

Author Keywords

MSaaS; Multi-Agent System; distributed simulation; cloud computing;

ACM Classification Keywords

I.6 Computing Methodologies: Simulation and Modeling; I.6.0 Simulation and Modeling: General; I.6.4 Simulation and Modeling: Model Development; I.6.6 Simulation and Modeling: Simulation Output Analysis; I.6.8 Simulation and Modeling: Types of Simulation—*Distributed, Parallel*

1. INTRODUCTION

Among the most demanded features in the field of multi-agent systems are the requirement to run simulations in the cloud as web accessible services [18, 16, 15] and to allow efficient simulation and analysis of massive-scale models [21, 10, 6, 11, 4, 19]. Especially the definition of MSaaS (Modeling and Simulation as a Service) [2] and first follow-up papers [9] express the interest in and necessity of cloud-based simulation. MSaaS systems are expected to deliver scalable simulation execution by means of a simple enough user interface suitable for non-technical domain experts.

The MARS simulation framework [5] developed by the MARS research group at Hamburg University of Applied Sciences, is very well aligned with the trend to move massive-scale simulations into the cloud. It offers a convenient web interface for modelers and features a chain of processes, which is designed alongside the modeling workflow.

In this paper we discuss an overview of related work and ideas (section 2), briefly present the simulation model used to showcase the MARS system (section 3), introduce

SpringSim-ADS 2016 April 3-6 Pasadena, CA, USA

our MARS workflow, role model and modeling paradigm (section 4) and outline the technological background of our cloud-based simulation execution (section 5). Finally the resulting workflow from experiments with our simulation model are shown (section 6) and a conclusion is drawn.

2. RELATED WORK

Research in the MARS Group has been focused on large scale models, scalability and MSaaS from the very beginning. Recent publications have shown that there is a growing interest on these topics by other researchers. We therefore provide a brief overview of related work in the domains of both MSaaS and high-performance computing.

2.1 MSaaS

The first system we like to address is mJADES [12]. It is a SaaS framework which as a cloud application allows the user to run multiple simulations in parallel. The name of mJADES and its technology is based on the cloud middleware mOSAIC and the simulation library JADES. JADES is implemented in the Java programming language. While MARS is a multi-agent system mJADES uses Discrete Event Simulation as simulation technique.

C^2SuMo (Cloud-based, Collaborative, and Scaled-up Modeling and Simulation Framework for STEM Education) [1] is a SaaS framework for traffic simulations. It uses SUMO, an open source road traffic simulation package, and enables scalability by employing multiple SUMO simulators in the cloud. Like its name says, C^2SuMo is developed to support education. Therefore it simplifies the SUMO interface to provide a more intuitive way for high school students creating traffic simulations.

There is another SaaS traffic simulation service called SEMSim Cloud Service[22]. It is agent-based, web-based, uses cloud computing to execute multiple simulations at the same time, enables multi-core usage and provides a real-time visualization for running simulations. Based on these attributes SEMSim CS exhibits great similarity to MARS. But a main difference consists in the supported simulation domains. SEMSim CS is made for traffic simulations while MARS makes no assumptions regarding the model domain.

2.2 High Performance Computing

Collier et al. [3] present Repast HPC as the distributable version of RepastJ or Repast Symphony. The motivation behind Repast HPC to build a large scale MAS is very similar to that of the MARS Groups'. That is to allow large-scale model simulation instead of optimizing a smaller-scale model by running many parallel simulations of the same model.

Repast HPC translates models into working simulations through a concept of agents, contexts and projections. A context is a set of agents, whereas the term set corresponds to its mathematical definition. Projections at last use contexts to model the environment. This structure allows multiple agents taking part in multiple environments and reusing projections. To distribute a simulation, Repast HPC uses a concept called Shared Projections. The environment created by a projection basically is a 2D grid due to the usage of the Logo language. This grid is sliced and then distributed across several processes. The buffer holds non-local agent stub objects from the neighboring slices and thus allows for changes / interactions to be made locally at first. The system then distributes the changes to the corresponding home objects in the other processes and takes care of synchronization matters. The communication and synchronization mechanism requires the user to provide specific pieces of code for each class that should take part in it.

Repast HPC provides a very scalable solution, which also allows for model re-usability through its projections and contexts. Test runs conducted with Repast HPC were run on high-end super computing hardware namely the IBM BlueGene cluster with up to 65.536 cores and Infiniband network.

3. SIMULATION MODEL

This section introduces a savanna ecosystem model, which is developed within the scope of the ARS AfricaE project (`www.ars-africae.org`) and is used as example model for this paper. ARS AfricaE is a joint project between German and South African research facilities that deals with the adaptive resilience of savanna ecosystems.

Savanna vegetation structure is usually characterized by large, solitary trees and shrubs scattered in a grassland matrix. It is assumed that savanna biomes are controlled by fire and herbivores, e.g. elephants [14]. The relevance of savanna ecosystems results in their ability to absorb large amounts of CO_2 and their distribution. Savannas cover approximately 20% of the earth's surface [13]. This turns them into an important stakeholder in the global carbon cycle system [20].

The savanna ecosystem model is under development within the ARS AfricaE project. It will help to achieve a better understanding of the population dynamics of woody species, particularly under the disturbance regime of global change.

To reduce the model complexity for the aim of this paper we are going to focus on a single species of trees and the interaction with elephants. The Marula tree species (Sclerocarya birrea) was selected because of its representative role for tree-elephant-interaction. Thus, our experimental design comprises agents with fixed spatial locations (trees) as well as mobile ones (elephants). Both agent types interact with each other and with their environment.

We simulate the whole area of the Kruger National Park - almost 20.000 square kilometres - with 15.000 elephant, 415 waterpoint and 5.2 million tree agents.

4. CONCEPT & WORKFLOW

The MARS system is conceptualized as a Modeling & Simulation as a Service system. This is an important

difference to other simulation frameworks. Every phase of the modeling lifecycle can be realized without installing additional software packages on the computer of the domain expert. Instead she or he accesses all functionality of MARS through a user friendly web interface. MARS is hosted and maintained by the MARS Group at the University of Applied Sciences in Hamburg.

These system features have been extracted from the MARS team's experiences in developing public transport disease spreading [7] and crowd evacuation models with the predecessor version of MARS called WALK [8]. It quickly became apparent that consolidating required datasets, model design decisions and the discussion of results were too inefficient when working in geographically distributed teams, without a system supporting that workflow.

4.1 Roles in the MARS Framework

Users of systems like MARS are mainly domain experts. They want to utilize the capabilities of multi-agent simulations to gain a better understanding of the complex systems they consider in their research. Since creating, using and analyzing a simulation model is rarely done by a single individual, we propose to accommodate each domain expert or group of experts with at least one tandem partner to deal with the more technical aspects of model implementation and simulation execution.

Therefore, within the MARS system we define and support a number of user roles.

Modeler A domain expert who creates the model to be used in the simulation.

Model Implementer A computer scientist developing the code for the model utilizing MARS APIs and libraries.

GIS & Data Scientist An expert in the field of data integration and GIS operations, who prepares datasets to be used by the simulation model and manages these datasets within MARS.

Of course one person may be assigned to more than a single role.

4.2 MARS Workflow

MARS follows a modeling and simulation workflow as shown in figure 1. This workflow is designed to be executed in a number of iterations, which include continuous refinement, simulation and validation of the model. In the final stage the results of the model are ready to be used in publications or further research.

Usually the modeler starts by creating the conceptual model according to the research question [17]. It might be useful to consult a computer scientist, when translating a conceptual model to a technical MARS model for the first time.

Once the modeler decides if her or his model is complete enough to try a first simulation run, the model should be discussed with a computer scientist to discover possible pitfalls, which might occur throughout implementation or simulation. Sometimes the model code needs additional information, which was not obvious during the more abstract modeling stage. Also this discussion should be used to clarify certain aspects, since there might be ambiguities to the model developer when simply reading the model description and not having deep knowledge about the domain. It should be mentioned that a modeler might write the model code himself, if she or he is trained in programming with the C# language.

With the first implementation done, the model can be uploaded to the MARS Websuite. GIS data and time-series data may be mapped to the simulation's layers and agent attributes. The data used in this step should have either been prepared (e.g. normalized) by a data scientist or the modeler himself. This task should be done in parallel with the modeling process and the data scientist should also partake in the discussion with the MARS developer.

The modeler can now trigger one or more simulation runs from the Websuite's interface and examine the results as a 3D visualization. A visual analytics page offers fundamental diagram types. Additionally, MARS offers the capability to export the results as a CSV file for further analytics with R or other solutions. These results may be accessed as soon as the first chunks of data are available from the simulation, thus a modeler does not have to wait for the simulation to finish. Validation of the results is the next step as designed by the MARS workflow. If fundamental errors are found in the results, the source of these errors will have to be searched either in the source code or the conceptual model itself. Usually modelers will work hand in hand with their tandem partner to fix these. In case the results are technically acceptable, it must be decided whether the model needs further refinement. If so, the next iteration starts. As soon as the modeler is satisfied, the results can be used in further work and the MARS cycle ends. The result files, model code, uploaded data and configurations will persist inside the MARS system for later usage.

4.3 MARS Modeling Paradigm: Layers & Agents

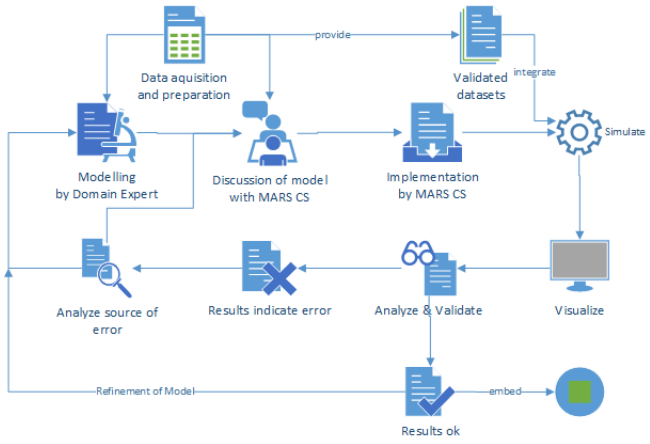

Figure 1. MARS Modeling Workflow.

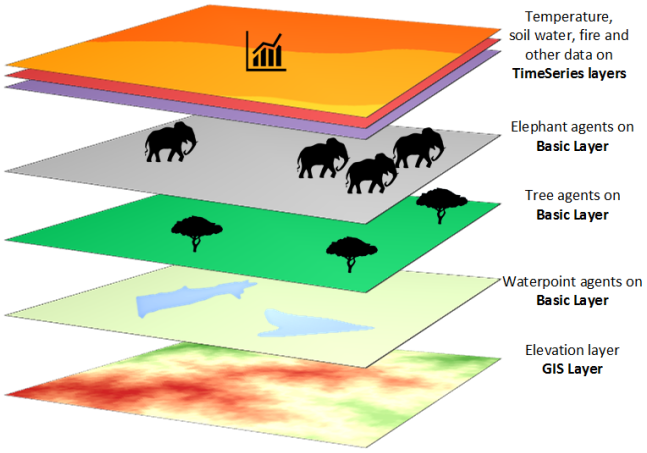

Figure 2. MARS Layer concept example from ARS AfricaE model.

The basic concept in MARS are agents and layers. That allows a unified way of developing MARS simulation models. This concept is essential and it must be used in every model. This section outlines the basic idea and showcases the application of layers and agents in our example model from section 3.

The layer concept is inspired by the way GIS data is composed. These files are structured in layers, where each layer represents a specific aspect. This aspect may be an agent type as well as a part of the environment.

We translate this idea to a general approach for modeling the implementation of our simulation system. A domain-specific model is transformed into working code by writing a layer for every aspect of the conceptual model. An aspect should be a considerable sized, self-contained but yet manageable piece of the original model. The layers represent the environment into which the agents are placed.

Figure 2 shows a layer model of the ARS AfricaE simulation model used in this paper. The bottom level keeps the digital elevation map (DEM). Internally it is represented by a GIS shapefile, in a resolution of 90 meters. Waterpoints, trees and elephants are represented as agents, which are placed on their corresponding layers. Finally we define a number of time-series layers, which are a special type of layer used to handle multi-scale time-series data from a database located in the MARS cloud.

This approach applies best-practice techniques from software engineering, e.g. separation-of-concerns. Hence layers could be seen as components with interfaces to each other. Each layer may expose well-defined operations to other layers through its interface. Agents may use the exported interfaces to access offered properties and services. MARS libraries provide capabilities to define sensors for agents, e.g. discover their surrounding environment. Thus, like in a service oriented architecture each layer is self-describing to external users and enables an agile way to compose and reuse agent and layer components.

4.4 Types of Layers

MARS currently provides three different base types of layers:

Basic Layer A blank layer, which has to be implemented by the user. Agents and environment are defined here.

GIS Layer A pre-implemented GIS layer. MARS is capable of filling this layer with a provided GIS file either in SHP or ASC format. Refer to section 6.1 for further information on how to map data to such a layer. The layer allows to query for data based on a position or a geometry, which includes polygons, multi-points and lines.

TimeSeries Layer A pre-implemented layer to access time series previously stored in the MARS ecosystem. The layer allows to query for data based on time and position.

5. MARS SYSTEM & AGENT SHADOWING

5.1 Overview

MARS is deployed in two major parts. The first and most visible part is the MARS Websuite. It hosts the website which modelers and model developers use to manage their data, configure their simulations, start simulation runs and analyze the results. The second part of MARS is the simulation system. It is instantiated and configured specifically for each simulation run which is started through the Websuite. All output and results from the simulation system are transferred back to the Websuite to be evaluated by the users.

The actual simulation component in the overall MARS architecture [5] is called LIFE. It consists of two main processes which make up the distributed simulation system. The SimulationManager is the centralized controlling application for the simulation. It manages the model, calculates the distribution and scheduling pattern, takes care of the distributed initialization and finally controls the simulation run. The LayerContainer houses layers and agents, which are the two primary logical components MARS simulations are made of. A LIFE system may be composed of any number of LayerContainers among which layers and agents are distributed. Layers are treated like plugins by LIFE and thus are loaded on demand when initializing a new simulation. This approach allows for automatic dependency injection, when one layer depends on another.

5.2 Implementation and Deployment

LIFE is completely implemented in the C# language. It can be run via the .NET runtime system on Windows and via the Mono project on Linux and OS X. By that it resembles the same platform independence as Java. However during development and deployment into production the MARS team chose to solely use Linux Docker containers (`www.docker.io`) for running the MARS Cloud infrastructure services and the actual simulation runs. Hence we rely on the Mono runtime for C# (`www.mono-project.com`).

To host MARS we make use of the well known Infrastructure and Platform as a Service (IaaS & PaaS) paradigms. To provide IaaS we rely on Linux KVM as our virtualization technology and OpenNebula as our management

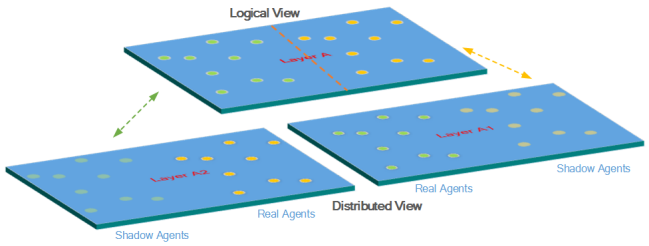

Figure 3. Agent Shadowing Concept - Logical View.

tool to operate virtual machines on top of our hardware. Linux KVM (`www.linux-kvm.org`) and OpenNebula (`www.opennebula.org`) both are open source projects and run on a wide range of hardware, which helps a great deal in achieving a cost-effective cloud environment. While hardware virtualization is provided by KVM, we use Docker to virtualize our applications. Docker allows to use the same environment during development and production, which enables a very fluent deployment process.

5.3 Agent Shadowing

Distribution and thus communication are two key aspects of scalability. In a very early version of the MARS system, layers were only distributable as a whole, so each LayerContainer needed to take care of one or more complete layers. However one layer may be too complex for a single computer or there may be some rather slow compute nodes involved. So MARS also allows to distribute each layer across several LayerContainers, which resembles true horizontal scalability. Since distributing layers has direct influence on the agents living on them, the approach for layer distribution is tightly coupled with the distribution of agents and is meant to make the overall system scalable. The approach is called Agent Shadowing.

Agent Shadowing is the depiction of an agent living on layer instance A having its shadow cast onto layer instance B, where it is not actually instantiated, but instead is represented by a stub-like object as in remote communication concepts like RPC/RMI (see figure 3).

In RPC/RMI each agent's methods and properties are callable by third parties through its stub object. Usually a stub just provides the capabilities to establish an interface-bound communication with the remote object. If the remote reference changes, in classic RPC/RMI the stub simply becomes useless, since its reference is not updated. The protocol then has to notice the broken link and re-establish a new one. MARS however prevents this problem by addressing each agent through a unique ID and a multicast address, which is calculated based on the agent type. Each message send to any agent thus is send via multicast to only those nodes, which house agents of the specific type. Each of the receiving nodes can then quickly decide whether it is responsible for the specified agent by looking up the ID in its local dictionary. If the ID is found, the message is translated into a method call and executed on the real agent object, otherwise the message simply is discarded. MARS LIFE creates shadow agent stubs (SAS) by making use of C#'s RealProxy class. This class consumes an interface and translates calls to that interface's methods into network messages. A SAS is extended by the ability to hold cached attributes like its environmental position or any other attribute. The real agent object then pushes new values to the SAS whenever its attributes are altered either internally or by an external influence. Managing state updates like this, minimizes the amount of messages needed throughout the whole system. Also these updates are delivered via multicast when in LAN to further reduce the amount of traffic.

Calling or referencing another layer, works by the same pattern of either having a local instance of that layer to address directly or a stub to communicate with a remote reference. Environmentally it does not matter which remote reference to a layer is provided, since each layer instance locally has a full view of its state, even if distributed.

All implementations regarding AgentShadowing are encapsulated within the AgentShadowingService, which itself is exposed to model developers by means of the AgentManager. This architecture allows the provision of a meaningful interface to developers, while hiding a lot of the technical implementation inside.

6. RESULTS

This section uses the ARS AfricaE model from section 3 to showcase the workflow with MARS. Screenshots are taken from actual running software. We don't see MARS as a prototypical implementation, but instead aim for a production ready solution, which can and should be used by as many modelers as possible.

At this point it is worth highlighting that Modeling as a Service is not yet achieved with MARS, since neither the conceptual model nor the implementation code can be created through the Websuite. However these features were planned during the initial design phase of MARS and are currently under development. They will be subject of future updates and research.

6.1 Simulation Model Preparation

After the conceptual model has been created and transformed into a MARS model (see figure 2), it needs to be implemented by a computer scientist. This step has to be accomplished in external development tools (e.g. Visual Studio or Xamarin Studio). The LIFE API is a direct match of the layer-based MARS model and together with additional supporting libraries (e.g. for agent creation) streamlines this process. However an in-depth discussion of how models are implemented would exceed the scope of this paper.

Once the model is implemented and uploaded, the domain experts may start working with the Websuite. We start out by creating a project and a scenario inside of that project. Projects are the largest organizational unit in MARS, while scenarios are more specific setups in a project. A scenario defines wall-clock simulation timespan, temporal and spatial

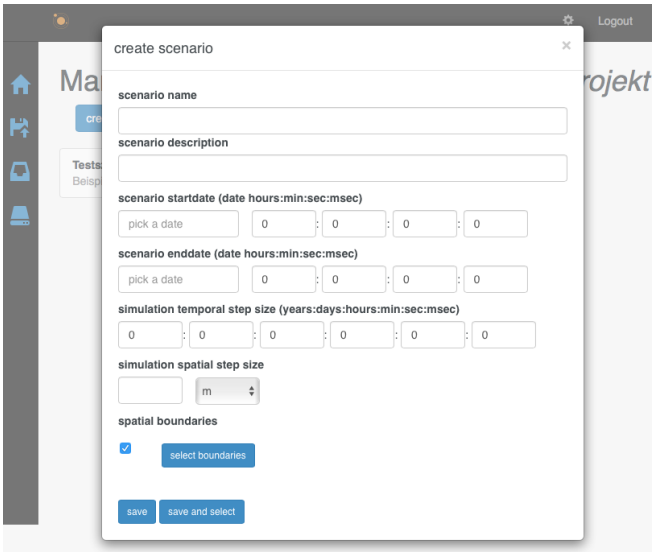

Figure 4. Scenario creation in Websuite.

step size and an optional spatial boundary. Figure 4 shows the corresponding form.

For the ARS AfricaE model several datasets are needed. We use a 90 meter resolution elevation map for the Kruger National Park, a shapefile containing tree positions, another shapefile with elephant herd positions and size as well as several time series in CSV format for temperature and precipitation data. All these datasets can be uploaded through the Import Data dialogue. Depending on the type of data different information has to be provided, i.e. where and when the dataset has been collected, who is the owner of the data etc. Datasets usually are available to all users of MARS, but can be flagged as being private to address data confidentiality concerns.

After uploading the datasets, the user has to define which compilation of data shall be used in the selected simulation scenario. With MARS DEIMOS we offer a tool to review the data and perform a validation against the scenario definition in terms of temporal and spatial scale and data availability. Once satisfied with the selection the tool will create a specially prepared compilation to be used in the next step. Note that MARS never alters the original data. Everything is either stored as meta-data or as copies of the original files, e.g. when a transformation in another format is needed

The next and penultimate step is to map the selected datasets to the simulation model implementation uploaded by the computer scientist. This is achieved with a tool we call SHUTTLE. SHUTTLE will only show parameters of agent constructors, attributed with "[PublishInShuttle]" in the model code, and further only those parameters which are mappable from the outside. This excludes other agents or layers for instance, since they will be injected automatically by MARS LIFE (see chapter 5). SHUTTLE provides a split-pane view featuring the extracted layers and agents from the model on the left hand side and the provided datasets on the right. Users can now use the data mapping buttons to dynamically create the domain specific language expressions in the middle pane, and thus map each needed agent parameter to a column from the datasets.

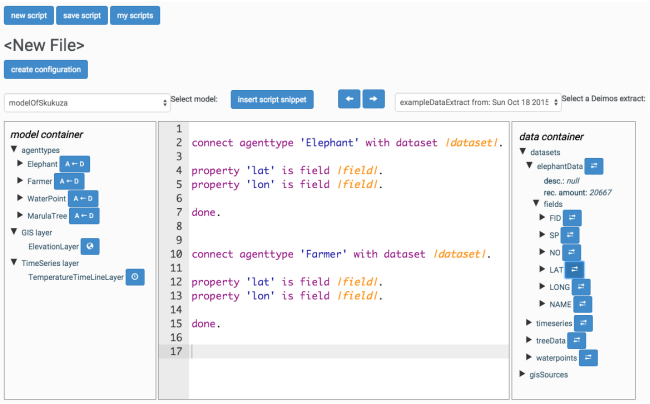

Figure 5. Data Mapping with SHUTTLE.

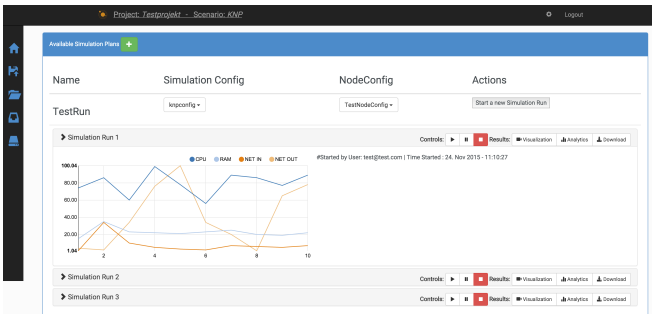

Figure 6. Starting a SimulationPlan from the Websuite.

Furthermore SHUTTLE exposes all GIS and TimeSeries layers and asks the user to map GIS and table datasets respectively to them. Figure 5 shows this process. The result of SHUTTLE's mapping process is a SimConfig file, which will be used in the simulation run to automatically initialize the simulation model with the data uploaded into the websuite.

6.2 Simulation Model Execution

The final step is about creating a SimulationPlan which will then be used to start one or more SimulationRuns. The user creates a SimulationPlan by selecting a SimConfig, a NodeConfig and providing a name. The NodeConfig controls on how many nodes and resources the SimulationPlan will be executed. The SimulationPlan may then be started via the web user interface, which results in a SimulationRun being created. Figure 6 shows the corresponding page used in the process. Basic real-time usage statistics for CPU, memory and network load for the current run are shown, when the SimulationRun tab is expanded.

When starting a model for the first time a Docker container image containing all relevant files is created. This image includes the needed C# runtime, model code, GIS files and the SimConfig description. Once the image has been created, it is stored and can be reused. This results in subsequent runs starting almost immediately. After the model container has been started, MARS LIFE will automatically

Figure 7. 3D visualization displaying data from the ARS AfricaE model.

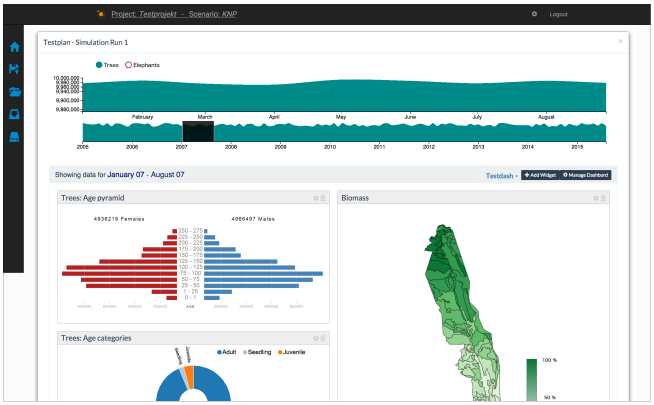

Figure 8. A visual analytics dashboard for the ARS AfricaE model featuring age distribution charts and a biomass choropleth map.

begin the model initialization by creating all layers in the order of their dependencies and by using the mapped GIS and time-series files. When the layers have been put together, LIFE instantiates all agents according to the mapping created in the SHUTTLE tool.

When run in a distributed manner with more than a single LayerContainer, LIFE automatically takes care of remotely initializing all layers and agents. Dependencies for layers are resolved by means of a LayerRegistry service.

6.3 Simulation Model Analyses

Once the model is running, first results are being sent to the Websuite and may be analyzed in any of three ways.

3D Visualization

First a 3D visualization can be displayed. This allows users to quickly check whether their simulation is performing in the way they envisioned. It is a particular good way to check for movement patterns, areal distribution of agents and if overall areal boundaries are done right. However a significant performance impact has to be expected when using this feature, since all information needs to be send to the visualization observer for every tick even though MARS optimizes this by only sending the information currently inside the viewing cone of the virtual camera. Figure 7 shows a sample visualization of our ARS AfricaE model.

Visual Analytics

A visual analytics page featuring basic graph types and maps like heat maps, may be used to create a dashboard for a SimulationPlan. The visualized data is updated in real-time as new data arrives and is very useful to check a model's indicator values as soon as they become available. This allows users to stop and readjust long-running simulations in case something is off right from the start. Also modelers can leverage the dashboard while optimizing their models, without ever leaving the Websuite or having to download large data blocks for offline analyses. Figure 8 displays the dashboard used for the ARS AfricaE model.

CSV Export

The third option is to download result datasets as CSV files. This is necessary when the data will be used in further research or when the capabilities of the visual analytics page are exceeded and more sophisticated statistical or visual procedures need to be performed (e.g. in R).

Finally, we want to highlight, that all result datasets are preserved until the corresponding SimulationPlans and SimulationRuns are deleted by a user with extended rights.

7. DISCUSSION & OUTLOOK

In this paper we introduced MARS as a MSaaS system. It is designed as a tool for global work groups that are considering multi-agent modeling and simulation to be used throughout their research. MARS provides a complete tool chain from data import to result visualization and analysis, which allows for large-scale model development in a web environment and execution in a high-performance cloud.

The current state of development includes an advanced Simulation as a Service approach, that enables users to start multiple simulation models in various configurations from their web browsers. Modeling as a Service however is in a work in progress state, but first prototypes of a visual modeling tool combined with a simplified modeling language are currently under development and look very promising.

Concerning 3D visualization and analysis we are planning to move to a more flexible approach of being able to review a complete simulation run and to allow fast-forward and fast-backward features. Based on the same data backend, another work is in preparation to provide a contextual search feature to efficiently browse through the results. For future updates and more information visit **`www.mars-group.org`**.

REFERENCES

1. Caglar, F., Shekhar, S., Gokhale, A., Basu, S., Rafi, T., Kinnebrew, J., and Biswas, G. Simulation Modelling Practice and Theory Cloud-hosted simulation-as-a-service for high school STEM education. *Simulation Modelling Practice and Theory 58* (2015), 255–273.

2. Cayirci, E. Modeling and simulation as a cloud service: A survey. *Proceedings of the 2013 Winter Simulation Conference - Simulation: Making Decisions in a Complex World, WSC 2013* (2013), 389–400.

3. Collier, N., and North, M. Parallel agent-based simulation with Repast for High Performance Computing. *Simulation 89*, 10 (nov 2012), 1215–1235.

4. Craenen, B., Murgatroyd, P., Theodoropoulos, G., Gaffney, V., and Suryanarayanan, V. MWGrid: A System for Distributed Agent-Based Simulation in the Digital Humanities. *2012 IEEE/ACM 16th International Symposium on Distributed Simulation and Real Time Applications* (oct 2012), 124–131.

5. Hüning, C., Wilmans, J., Feyerabend, N., and Thiel-Clemen, T. MARS - A next-gen multi-agent simulation framework. *Simulation in Umwelt- und Geowissenschaften, Workshop Osnabrück 2014*, 2008 (2014), 1–14.

6. Kiran, M., Richmond, P., and Holcombe, M. FLAME: simulating large populations of agents on parallel hardware architectures. *Proceedings of the 9th . . .* (2010), 1633–1636.

7. Münchow, S., Enukidze, I., Sarstedt, S., and Thiel-Clemen, T. WALK: A Modular Testbed for Crowd Evacuation Simulation. In *Proceedings of the 6th International Conference on Pedestrian Evacuation Dynamics*, U. Weidman, Ed., Springer (2014).

8. Noetzel, C., Reintjes, R., and Thiel-Clemen, T. Die Rolle öffentlicher Verkehrsmittel bei der Übertragung und Verbreitung von Krankheitserregern. In *58. Jahrestagung der Deutschen Gesellschaft für Medizinische Informatik, Biometrie und Epidemiologie e.V. (GMDS)*, H. Handels and J. Ingenerf, Eds. (sep 2013).

9. Padilla, J. Cloud-Based Simulators: Making Simulations Accessible To Non-Experts and Experts Alike. *Proceedings - Winter Simulation Conference 2014*, 2012 (2014), 3630–3639.

10. Parker, J. A flexible, large-scale, distributed agent based epidemic model. *Proceedings - Winter Simulation Conference* (2007), 1543–1547.

11. Parker, J., and Epstein, J. M. A Distributed Platform for Global-Scale Agent-Based Models of Disease Transmission. *ACM Transactions on Modeling and Computer Simulation 22*, 1 (2011), 1–25.

12. Rak, M., Cuomo, A., and Villano, U. mJADES: Concurrent Simulation in the Cloud. *2012 Sixth International Conference on Complex, Intelligent, and Software Intensive Systems* (2012), 853–860.

13. Scholes, R. J., and Walker, B. H. *An African Savanna - Synthesis of the Nylsvley study*. Cambridge University Press, 1993.

14. Shannon, G., Thaker, M., Vanak, A. T., Page, B. R., Grant, R., and Slotow, R. Relative Impacts of Elephant and Fire on Large Trees in a Savanna Ecosystem. *Ecosystems*, 14 (2011), 1372–1381.

15. Taylor, S. J. E., Balci, O., Cai, W., Loper, M. L., Nicol, D. M., and Riley, G. Grand challenges in modeling and simulation: expanding our horizons. *Proceedings of the 2013 ACM SIGSIM conference on Principles of advanced discrete simulation.* (2013), 409–414.

16. Taylor, S. J. E., Fujimoto, R., Page, E. H., Fishwick, P. a., Uhrmacher, A. M., and Wainer, G. Panel on grand challenges for modeling and simulation. *Proceedings - Winter Simulation Conference* (2012).

17. Thiel-Clemen, T. Designing Good Individual-based Models in Ecology. In *Simulation in Umwelt- und Geowissenschaften, Workshop Leipzig*, J. Wittmann and M. Müller, Eds., GI, Shaker (2013), 97–106.

18. Tolk, A., and Mittal, S. A necessary paradigm change to enable composable cloud-based M&S services. *Proceedings of the 2014 Winter Simulation Conference*, 2011 (2014), 356–366.

19. Vigueras, G., Orduña, J. M., Lozano, M., and Jégou, Y. A scalable multiagent system architecture for interactive applications. *Science of Computer Programming 78*, 6 (jun 2013), 715–724.

20. Williams, C. A., Hanan, N. P., Neff, J. C., Scholes, R. J., Berry, J. A., Denning, A. S., and Baker, D. F. Africa and the global carbon cycle. *Carbon balance and management 2*, 3 (2007), 1–13.

21. Yamamoto, G., Tai, H., and Mizuta, H. A platform for massive agent-based simulation and its evaluation. *Massively Multi-Agent Technology* (2008).

22. Zehe, D., Knoll, A., Cai, W., and Aydt, H. {SEMSim} Cloud Service: Large-scale urban systems simulation in the cloud. *Simulation Modelling Practice and Theory 58* (2015).