Abstract
The growing interconnectedness of ecological, social, and political systems poses a challenge in modeling them. Individual Based Modeling (IBM) has proven to be a useful tool in these areas. As the field becomes more developed and the number and size of available models increases, new problems arise. Application scenarios become huge in size, number of agents and integration of models.

As a result, the requirement for the integration of models arises, along with the need to compute a realistic number of agents for use with IBM in, for example, prognostic simulation models. Other problems revolve around the collection and, most importantly, integration of data.

Although some of the problems have been solved others remain. For example, scalable, general-purpose, multi-agent systems are still problematic. Still, scalable, generic and performant frameworks for multi-agent simulation are sparse, if not missing at all.

We define a set of requirements for a “next-gen” multi-agent simulation framework, compare existing approaches with these demands and articulate the need for a framework that combines strengths of existing ones with improvements of previous findings in scalability.

The MARS (Multi Agent Research and Simulation) Group of the Hamburg University of Applied Sciences is developing a distributed and highly scalable framework for use in research and education. MARS is not a single program, but consists of a multitude of processes and tools chained together to provide an approach to today’s simulation requirements. We describe an overview of the system and the state of research.

1 Introduction
The usage of Individual Based Modelling (IBM) in ecological science is well accepted for 25 years now. During this time a lot of experience on when and how to use ABM has been collected (Filatova et al., (2013), Grimm, (1999), Huston et al., (1988)), as well as new approaches, methods and technologies in computer science have emerged (Bellifemine et al., (2007, 2008), Grimm et al., (2006, 2010), Le et al., (2008), Ralha et al., (2013), Vigueras et al., (2013)).

Together with these improvements, new obstacles and problems have arisen in the various domains of ecological science. The coupling of ecological, social, economic and political systems creates a huge complexity to the overall model and simulation. Some of the major challenges are the integration of different models and almost arbitrary data into combined simulation models (Thiel-Clemen, 2013). Additionally, executing performance of simulations should be adequate even for large-scale
scenarios. A high level and target-group specific visualization is also taken for granted.

Creation and usage of multi-agent based simulation systems has proven to be a great tool to explore and investigate such models (Le et al., (2008), Ralha et al., (2013)).

The MARS Group (Multi Agent Research and Simulation) of the Hamburg University of Applied Sciences is developing a distributed and highly scalable framework for use in research and education. MARS is not a single program, but consists of a multitude of processes and tools chained together to provide an approach to today's simulation requirements.

In this paper we outline our requirements for a modern MAS-framework (Section 2), discuss an overview of relating literature and solutions (Section 3), present the current state of development of the MARS SYSTEM (Section 4) and provide an outlook to our future research and development topics (Section 5).

2 Requirements for modern simulation systems

This section outlines our understanding of the key requirements for a modern simulation system. We feature the findings from various corresponding work, as well, as our own experience, while designing a solution for the tasks at hand (e.g. Pereki, 2013).

2.1 Modularity and Reusability

As shown (Liu et al., 2007), almost every ecosystem today is tightly coupled with its neighboring economic or social systems and thus these need to be taken into account when watching the evolution of that ecosystem. Filatova et al., (2013) go even further by demanding that the corresponding aspects of ecological systems like economy, social systems and bio-physical dynamics need to be integrated into the representation of a heterogeneous landscape representation.

One of the most important requirements resulting from this circumstances is the integration of existing models with each other. This can only be done if models, or even better, their parts, are designed in a modular and reusable manner. The idea is to connect and integrate domain specific models from domain specific experts to create a new super model of a certain domain (or, alternatively, focus on different compositions). If for example one would want to create a large scale model of a given ecosystem in south Africa, it would be very helpful, if one could use already existing models of certain components, such as animal behaviors, weather, land erosion and so on.

Another aspect that could profit from modular and reusable models is comparison. If it was easy to integrate most of the models available, models\(^1\) can be run directly

\(^1\) For example with competing hypotheses or simply older versions of the same model.
next to each other, consuming the same data, allowing for example to perform real-
time digression analyses.

Actually integrating models turns out to be extremely difficult, since each group of
scientists working on a model tends to use another, individual, paradigm, architecture, programming language or data format. A good solution should address
this problem.

### 2.2 Information Integration

Of huge importance in simulation is data. It is needed for nearly all tasks from
generation of hypotheses, over simulation initialization and calibration to validation.
Unfortunately the data that is being collected, has a tremendous heterogeneity in
terms of temporal and spatial resolution, reference formats, completeness and error
margins. To be viable in a simulation, this data has to be integrated. It must be
carefully corrected, the resolutions have to be aligned, the error must be treated.

Furthermore the relevant data of all the available must be singled out and connected.
Since we focus on spacially explicit simulations, a special point is also to link data
without any further reference together to establish a common context. For example
we might be designing a model for an animal species in a wildlife reserve somewhere
in Africa. For one concrete simulation it could be necessary to include weather data
for the whole region, topology data of the general landshape, as well as a rough
overview of vegetation types and population metrics for certain species in that area.

A simulation framework should assist domain experts with all the steps involved: GIS
imports, data collection and analysis and possibly transformation.

These tasks target the difficulty when technically connecting different models. A more
functional view to the importance of information integration has been made by (Liu et
al., 2007) who take a look at the complexity of coupled human and natural systems.
Their integration efforts aim at taking interdisciplinary research on a broader scale
into account, as well as exceeding local and temporal boundaries when modelling
certain ecological system.

### 2.3 Ease of use

To be useful for and accepted by experts of other domains than computer science, a
simulation system should also be as accessible as possible. There are two aspects
that we want to emphasize in this context. One is the ever important question of
usability of the general toolset. The other is the nature of the means provided by the
simulation system to model the actual questions. Specifically a good solution should
address and overcome the gap between the domain specific model and its
corresponding technical representation in the simulation system.
2.4 Scalability

Although it should always be the goal of a modeler, to design everything as simple as possible, some things are inherently computationally intensive. There are several scenarios that, often in combination, prohibit simulation execution on a single computer within reasonable time frames.

First of all the agents themselves are becoming more complex, in order to replicate natural behavior. This is especially true for animate objects, such as for example animals or humans. To come close to the real world, the modeler might need to use computationally expensive techniques, such as learning or planning algorithms, path-finding, collision avoidance and others, often even in use simultaneously. And the more models are integrated, the more of those techniques are likely to occur.

As the field of multi-agent systems research matures, the applications get also bigger, resulting in a larger number of agents. Imagine for example a continuous field with an average agent density of one agent per square meter accordingly; the system has to handle about 100 agents. Now, if the length of the sides is only doubled, the computational effort increases fourfold, in the three-dimensional case even eightfold\(^2\).

The real world areas of interest are steadily growing larger, further intensifying this problem. This is especially true, when a model is used to forecast future developments of its real world counterpart. Initially mostly used for the understanding of dynamic systems, IBM is likely to be used increasingly for large scale prognosis as well. The area of interest may for example be something like the Abdoulaye forest (Pereki, 2013) as in our current use case.

Of course it may sometimes be possible to avoid the problem by extrapolating from a sample set of agents to the bigger scenarios. But that would in return diminish the factor that sets apart IBM from other simulation techniques: the ability to track individual agent’s actions and states. Also, depending on the system, some desirable emergent properties of the real system are only achievable with a realistic density of agents. For example Yamamoto et al. (2008) found that massively increasing the amount of agents in an auction simulation, significantly changed the outcome of the simulation.

The most promising solution to really solve this problem, is to make the simulation system scalable across multiple computers. Research budgets are not limitless, so we think it is important to target commodity hardware or rentable compute clouds. Scalability by our definition means the computation speed of a single simulation run increases by a constant factor per added compute node.

2.5 Visualization

Since we are opting for large scale simulations with millions of agents, we need a visualization solution that copes with these numbers. To the best of our knowledge

\(^2\) In most cases the height dimension, although present, is going to remain constant. It can of course still be influential, when the air is of interest (e.g. when simulating flying animals).
there is no affordable current graphics engine or hardware that allows to render these numbers in real-time at once to the screen.

Therefore our solution should be able to visualize only a specified section of the whole simulation space. We further require that section to be dynamically movable and resizable.

As there might be a lot of data bound to the entities and patches of environment we are to visualize, a good solution should allow to enable and disable certain levels of information (e.g. weather, vegetation etc.).

Lastly, the rendering should also be efficiently possible in 3D, if a scenario demands or would greatly benefit from it, like for example evacuation scenarios..

2.6 Summary


3 Related work

This chapter provides an overview of recent and popular frameworks and approaches. Since we strive to create a framework, we evaluate these solutions against our requirements.

Villa, (2001) proposes his Integrating Modeling Architecture (IMA) to mitigate the requirements from chapter 2. He singles out three characterizing dimensions for connecting different models:

- **Representation** A unified semantic relating to the depiction of space, time and behavior in every respective model is needed.
- **Domain** A clear distinction between the domain spaces of each sub-model must be made. In particular this relates to the input and output parameters, which are valid for each sub-model.
- **Scale** Data, which is exchanged between models, must be compatible or translated in space and time dimensions

3.1 Simulation Frameworks

3.1.1 JADE

One of the most popular frameworks is JADE (Bellifemine et al., 2007), which allows executing a simulation distributed across several JADE container processes or just locally in a single container. JADE was developed in Java to create a reference
implementation of the FIPA agent specification (http://www.fipa.org). (Mengistu et al., 2008) extensively investigated the performance of JADE. Their findings show that JADE has significant performance issues in the fields of communication and agent migration due to the usage of the LDAP protocol and slow message transport services. JADE’s Lookup-Directory-Service also is measured to be slow, which is caused by not using local caching on the respective nodes. Mengistu et al., (2008) propose improvements to both mechanisms and present promising results from experiments they conducted. However a more recent investigation of JADE’s performance seems appropriate, given that the paper is almost 6 years old.

3.1.2 GAMA

GAMA (Amouroux et al., 2007) is a modeling and simulation framework, which is based on RepastJ. It features a model description language, called GAML, which allows nonprogrammers to create complex models. GAMA is written in JAVA and thus executable on all enabled systems. A very strong feature of GAMA is its visualization feature, especially when it comes to using GIS data. An easy import function allows to quickly create a scenario’s environment and visualization from a GIS file and thus allows for a quick integration of that kind of data. These features make GAMA very accessible.

The downside of GAMA is, that it’s not possible to distribute the system and that it does not scale well across multiple CPU cores. In fact when testing GAMA, it actually used only just up to 4 cores while running on a 24-core machine. While testing we found GAMA to have a performance threshold around 80,000 agents that were doing a basic predator-prey model. One simulation step in that model took more than 800ms on the aforementioned machine.

3.1.3 WALK

WALK is a MAS with a strong focus on evacuation scenarios and has been developed at Hamburg University of Applied Sciences (Thiel, (2013) and Thiel-Clemen et al., (2011)). It features a dynamic (re)partitioning and distribution of agents across several compute nodes and is thus capable of running simulations with hundreds of thousands agents on commodity hardware. In fact Thiel, (2013) showed in his final tests that WALK can run a 300,000 agent random walk simulation in near real time. Also remarkable about WALK is, that it has been extensively tested and validated against real world scenarios.

As a recent addition Stefan Münchow added leadership models and social behavior to the agents implemented in WALK. These additions show very promising results and create a very high interest in re-using the agent implementation from WALK in the new system whenever human agents are explored. Also our ideas for the distribution are trying to generalize the approach taken in Thiel (2013) from that work.

3.1.4 Vigueras

Another interesting architecture (Vigueras et al., 2013) proposes an almost completely asynchronous, distributed simulation execution to implement interactive
simulations, which may be visualized in near real-time. The only time (Vigueras et al., 2013) synchronize the execution of their agents is, when they happen to act or move beyond the boundaries of their respective environment patch.

When it comes to visualization of the simulation (Vigueras et al., 2013) utilize visualization nodes (VS) that also act asynchronously on the distributed nodes. Each VS has a camera-style definition of its field of view and may thus only ask those nodes for information containing parts of the environment, which is in that field of view. This is very contrary to other visualization approaches (e.g. GAMA, NetLogo), since it does not attempt to visualize the whole simulation at once.

Considering the amount of agents and the sheer size of simulated space in our upcoming scenarios, this approach might become very valuable.

3.2 Focus on scalability

As a consequence of the former considerations we come to the conclusion, that there are some simulation systems that really fit the general-purpose character and support the modeler with for example GIS integration. The same tools also offer advanced modeling solutions, such as a domain specific language (GAMA). Other tools on the other hand excel at the distribution but are crafted for the special model they implement (WALK, Vigueras). Also, most systems examined, which claim to be scalable, in fact are limited in this area (e.g. GAMA).

As discussed above, there are many reasons why a quasi-standard simulation system, which combines a general-purpose approach with full-fledged horizontal scalability, would be desirable. Other reasonable features should be the facilitation of model creation and integration as well as information integration (Thiel-Clemen, 2013). Letting computer science professionals translate the model can bridge the process of model creation from domain experts to simulation. However, individual based models with viable size for real world scenarios are not possible without a system, which computes them in reasonable time frames.

For those reasons we decided to design a completely new framework instead of building upon or augmenting an existing one. After giving a brief overview of how the simulation component fits within the whole planned tool chain, the focus of this paper will be on our ideas regarding the scalability aspects of the central simulation.

4 MARS Overall Architecture

4.1 Overview

The system’s components on the top-level are shown in Picture 1. MARS is divided into three main parts, whereas the top-level components are assigned as follows:

- **Data Integration**: GROUND, ROCK and SHUTTLE
- **Simulation**: LIFE
- **Analysis & Visualization**: VIEW and SURVEYOR
Following is a description of what each component’s purpose is and how they interact.

### 4.1.1 GROUND

MARS GROUND is the component responsible for all interaction with geo-data. It encapsulates the connection and interaction with Geoserver\(^3\), which it utilizes to import, edit and export geo-data. GROUND further solves the problem of different projections, formats and scales on heterogeneous geo-data (Hagenlocher et al., 2014).

### 4.1.2 ROCK

MARS ROCK holds all data that is not optimally representable in GIS formats, like everything being not spatial-explicit or simply not coming in a compatible GIS format. This is especially true for data that changes depending on time, e.g. the population of a country.

ROCK uses the Bing Tile Map System from Microsoft\(^4\) to reference data to geographical coordinates and to allow efficient queries on it. With that approach applied, data from ROCK and GROUND may be cross-referenced, which solves one of the big problems of information integration in MAS (Thiel-Clemen, 2013).

---

\(^3\) [http://geoserver.org/](http://geoserver.org/)

4.1.3 SHUTTLE

This part of the system integrates the data from GROUND and ROCK behind a unified interface, so that the user is not able to tell from which technical source the data originates.

MARS provides a tool by the name of DEIMOS, which makes use of the SHUTTLE API and allows the modeler to extract data from ROCK and GROUND for a specific geographical area to be used in the simulation model at hand. The modeler may then intersect this data with each other and/or build an SGI\(^5\) (Baldowski et. al., 2014). Finally an XML description file will be saved, which contains the actions needed to extract and transform the appropriate data from GROUND and ROCK to re-create the needed subset of data when the live simulation is about to start.

In order to allow the actual simulation system (LIFE) to use the data extracted by DEIMOS, a specific MARS DSL is currently under active development, which will be used to bind attributes of agents and other simulation entities to the extracted fields in the ROCK/GROUND databases. Therefore the formerly created XML file will be used to present the modeler with the fields available for mapping.

Once the binding is done, SHUTTLE creates an in-memory database containing only the subset of data extracted and transformed from GROUND and ROCK. This temporary database may then be used by LIFE to instantiate its agents and environment and to perform other tasks based on the provided data.

4.2 LIFE Architecture

LIFE is the actual distributed simulation component and is itself organized in three main components as can be seen in Picture 2:

- **Simulation Controller**: The GUI to access and control the overall system.
- **Simulation Manager**: A central management unit, which takes the orders from the Simulation Controller and executes it.
- **Layer Container**: A container to house layers and agents (see next chapter). Each LayerContainer may contain many layers or pieces of layers if they are distributed in itself (see Fehler! Verweisquelle konnte nicht gefunden werden.).

\(^5\) Spatial Gemischter Indikator, in english: Spatial Mixed Indicator
4.3 Layers & Agents

A central concept of LIFE is the layer approach. It is inspired by the way GIS files are composed. These files are structured in layers, where each layer represents a specific aspect of the depicted real world (see Picture 3). 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 may now be transformed into working code by writing a layer for each aspect from the model. An aspect should be a considerable sized, self-contained but yet manageable piece from the original model.

A useful distinction could possibly exist between different kinds of agents with the exception being two classes of agents, which are heavily interacting with each other. A separation of these classes on different layers might introduce unnecessary overhead and complexity to the code.

With that approach in place we can apply well understood techniques from software engineering and thus understand layers as components with interfaces in our distributed simulation system. Each layer can now export well-defined actions to other layers through its interface.

Layers are treated like plugins by each LayerContainer and thus may be loaded on demand when initializing a new simulation. This approach also allows for automatic dependency injection, when one layer depends on another.

To further leverage the usability of the layer concept, each layer features a three dimensional coordinate system which works with relative sizes, which may be specifically set for each simulation model. This allows for seamless geographical cross-referencing when entities of two different layers are interacting with the corresponding layer interfaces and fits nicely with the provided data from MARS’ data integration sub-systems.

So, just as in a service oriented architecture each layer is self-describing to external users and thus supports the demanded feature of re-usability while maintaining well known programming patterns at the same time.
4.4 LIFE API

This chapter is a short outline of the basic API used when implementing custom layers and / or agents.

4.4.1 ILayer, ISteppedLayer & IEVENTDrivENLayer

A new custom layer has to implement one of the interfaces offered by LIFE. One may choose between the ISteppedLayer, IEvenDrivenLayer or ILayer. Whereas ILayer mainly acts as a marker interface and therefore should not be implemented directly, the other two resemble specific behaviour. The IEvenDrivenLayer currently is not active and might become a subject of future research when other execution paradigms to MAS than the stepped approach are in focus.

ISteppedLayer features two methods

- InitLayer() : Takes three parameters. The first for providing the data needed for initialization, the other two are callback handles to register and un-register agents to the LayerContainer (see 4.4.2).
- GetCurrentTick() : To retrieve the current tick count from the stepped layer instance

As can be seen, the interface is very slim, leaving most of the implementation details to the developer. Future additions will provide several service libraries to make common tasks like pathfinding, radial searches in a 3D space or usage of height maps or other GIS data easier when developing a new layer.

4.4.2 ITickClient & IAgent

IAgent is an extension of ITickClient. ITickClient features a single method called Tick(). Every agent needs to implement IAgent and thus Tick().

During the layer initialization method InitLayer(), each layer needs to register its agents or TickClients with the LayerContainer for execution, which is done by calling the callback handle for each of the agents in the layer. Once the initialization is done, the LayerContainer has references to every tickable entity inside each layer and may thus efficiently execute each simulation tick by assigning CPU time to each tickable entity.

Currently IAgent does not add any other method, but just marks the implementing class as an Agent. This would allow to add other tickable or tickworthy entities than agents in the future, if a layer developer finds it suitable to create something different than a strict agent to implement a certain piece of the model at hand.

4.5 Analysis & Visualization

Inspired by the findings of (Vigueras et al., 2013) and the notion that other non-well-scaling MAS like GAMA or NetLogo attempt to visualize the whole simulation space at once, we decided to go with a more sophisticated approach.
Just like in the architecture of Vigueras, MARS will define camera views onto the simulation space to single out specific areas for visualization. These cameras will be managed by a specific centralized process, which is responsible for triggering the liable LayerContainer instances to send their data to the visualization client.

5 Outlook

As of the writing of this concept paper, a preliminary version of MARS is under active development to produce first test results for all described concepts. We use a simplified version of the Abdoulaye scenario (Pereki, 2013) to produce quasi-realistic loads and to evaluate our overall architecture.

Especially our distribution and communication approach, will be the main subject of our research, since it is a very cruciable part of the overall infrastructure. Though early results look promising, intensive tests need to be performed, to examine the behaviour under various circumstances.

The management and distribution of the environment is another big question, which we will have to investigate in the near future, as well as the partitioning of the agents across the distributed layer instances.

While we do that, other members of the MARS research group are using this early version to create other domain specific models and produce further experience, test results and feedback, which will lead to insights whether or not we meet our requirements.
References


<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
</table>