From testing agent systems to a scalable simulation platform

Tobias Ahlbrecht, Jürgen Dix*, and Federico Schlesinger

Department of Informatics
Clausthal University of Technology
Julius-Albert-Str. 4
D-38678 Clausthal-Zellerfeld, Germany
{tobias.ahlbrecht,dix,federico.schlesinger}@tu-clausthal.de

Abstract. Since 10 years our group in Clausthal is organizing the Multi-Agent Programming Contest, an international contest providing a flexible testbed for evaluating prototypical implementations of agent systems. We describe in this paper how the scenarios developed over time, which lessons we learned, and how this endeavour finally led to the idea of a scalable multiagent simulation platform. The important conclusion we draw is the need to move from academic prototypes to more seriously engineered software systems in order to support the uptake of academic research in industry.

1 Introduction

The year 1980 marks an important date in knowledge representation and reasoning. *Artificial Intelligence* published a special issue containing three of the most important papers starting a completely new field: *nonmonotonic reasoning*. Our dear colleague, Gerhard Brewka, is working in this area since the mid 80's and helped forming the field.

The first two decades have seen an enourmous amount of research which shed light on the relations and formal properties of many variants of nonmonotonic logics. The second author worked for many years on the relation between logic programming semantics and nonmonotonic reasoning. Two of the most prevailing goals have always been the following: *(1) Define a computable and efficient formalism to handle commonsense reasoning. (2) Develop an engineering methodology to apply this formalism to real-world problems.*

The gap between theory and practice has been huge in the beginning (it still is large). While the first goal initiated an impressive amount of work over the years, the second goal was not taken up with the same devotion.

Interestingly, one of the main researchers in nonmonotonic reasoning, Yoav Shoham, was also one of the influential people to start in the 90's another line of

*I would like to express my gratitude for working with Gerd Brewka in the late 80's and 90's, when I started my own research. It was a terrific time!

research, which led to the ever flourishing area of *agent systems* (his seminal paper on agent-oriented programming was also published in *Artificial Intelligence* in 1993).

The notion of an *intelligent agent* is perhaps the most important idea in artificial intelligence in the last four decades and turned out to be extremely influential in many areas (as evidenced by the recent textbook [10], the AAMAS conference series and many associated workshops, eg. ProMAS, EMAS, CLIMA, DALT, AOSE). The question *How does an agent take its decisions?* is closely related to classical knowledge representation and reasoning mechanisms: It is indeed a nonmonotonic procedure. Agents need to reconsider their intentions, revise their belief in the light of new information, and thus act in a nonmonotonic fashion.

An important feature is that agents always act in an environment with many other agents: they are not alone. This led to the introduction of *agent programming languages*. Most of these languages were still in their infancy at the beginning of this millenium. They were often developed within a PhD or in similar smaller projects, based on some sort of *computational logic*. Such implementations were proofs-of-concept, rather than seriously designed software systems.

In 2004, during one of the CLIMA conferences, the following idea (suggested by Paolo Torroni and Francesca Toni) came up: to organize an annual international event as an attempt to stimulate research in the field of programming multiagent systems by 1) identifying key problems, 2) collecting suitable benchmarks, and 3) gathering test cases which require and enforce coordinated action that can serve as milestones for testing multi-agent programming languages, platforms and tools. In 2014 the competition was organized and held for the tenth time.

Similar contests, competitions and challenges have taken place in the past few years. Among them are *Google's AI challenge*[1], the *AI-MAS Winter Olympics*[2], the *Starcraft AI Competition*[3], the *Mario AI Championship*[4], the *ORTS competition*[5], the *Planning Competition*[6], and the *General Game Playing Competition*[7].

The plan for this paper is as follows. In Section 2 we describe our Contest in more detail. In Section 3 we discuss the lessons we learned. Section 4 (which is based on joint work published in [3,2]) develops the idea of a scalable multi-agent simulation platform *MASeRaTi* that evolved out of (1) our work on the Contest, and (2) work on traffic simulation of the group led by our colleague in Clausthal, Jörg Müller. Finally we draw some conclusions and look into the future.

[1] `http://aichallenge.org/`
[2] `http://www.aiolympics.ro/`
[3] `http://eis.ucsc.edu/StarCraftAICompetition`
[4] `http://www.marioai.org/`
[5] `http://skatgame.net/mburo/orts/`
[6] `http://ipc.icaps-conference.org/`
[7] `http://games.stanford.edu/`

2 The Multi-Agent Programming Contest

The Contest is an international annual event that first took place in 2005. Over the years, the contest went through well defined episodes, characterized by the scenarios in which the agents perform and compete. Originally it was designed for problem solving approaches that are based on formal approaches and computational logics. But this was never a requirement. Indeed in the last few years we have seen participating teams which programmed entirely in Java or Python.

The trend in the design of the scenarios has been to increase the complexity; instead of a clever algorithm that solves the scenario (perfect algorithm), we wanted scenarios in which intelligent agents can and should make use of capabilities such as autonomy, coordination, flexibility, proactiveness and reactiveness, etc., features that the multi-agent programming paradigm aims to facilitate. We wanted to evaluate the underlying languages/systems by checking whether they support such capabilities.

Even though the contest has inspired some interest from other research communities and individuals, the core of the participants belongs to the multi-agent programming research community, and most of them are designers of multi-agent languages and platforms, who find in the Contest an excellent test-bed and benchmark for their own developments.

For this purpose the scenarios are adapted to the state of the art in multi-agent programming, and do not increase in complexity arbitrarily. Some desired features for the environments, such as a greater degree of uncertainty (e.g. by means of actions failing with higher probability, so that learning or nonmonotonic formalisms are useful), have been put off. While we could certainly put more emphasis on evaluating such knowledge representation issues in the future, the available languages currently do not provide enough constructs to deal with these problems.

2.1 The underlying platform

The first edition of the Contest presented a relatively simple scenario that had to be implemented in its totality by each participant and delivered as an executable for evaluation by the organizers. For the second edition, the *MASSim* infrastructure was introduced. *MASSim* is an extensible simulation server that provides the environment facilities. Agent programs can connect through the network to a *MASSim* server and control simulation level agents; this allowed the Multi-Agent Programming Contest to be run in a different way: the competitors should only focus on the agents' design and implementation; agents are run in the competitors' own computer infrastructure and connect to a *MASSim* instance running in the contest organizer's infrastructure, in which the scenario is implemented.

Besides freeing the competitors from dealing with the implementation of the environment, another key factor provided by *MASSim* is that agent programs from different locations can connect to the same simulation, thus enabling competitive scenarios. Since the introduction of *MASSim* in the 2006 edition, the

format of the Contest has been that of two teams competing against each other for performance in each simulation, and the overall winner of the contest defined by summing up the points after all participants have competed in simulations against each other, in a regular sports' tournament fashion.

All simulations are run in a step-by-step manner. In each step all agents execute their actions simultaneously from the point of view of the server, and there is a time limit within which agents must choose an action (otherwise they are regarded as a `no-op`). In the beginning of each step's cycle, the server sends each agent its current percepts of the environment, and waits for the response that specifies the action to execute. When the responses from all agents are received or when the timeout limit is reached, all received actions are executed in *MASSim*, and the agents' percepts for the next step are calculated. This cycle is repeated for a fixed number of steps, and then a winner is decided according to scenario-specific criteria.

MASSim is fully implemented in Java, and the information exchange with the agent programs is made through XML messages. It follows a plugin architecture for the simulations, which makes it easy to design new scenarios on top of it, as has been the case during the evolution of the Contest. Figure 1 describes this architecture. The addition of new scenarios does not imply the replacement of the previous one. Many different scenarios can convive within a single instance of *MASSim*, and they can be activated by choosing or modifying configuration files accordingly.

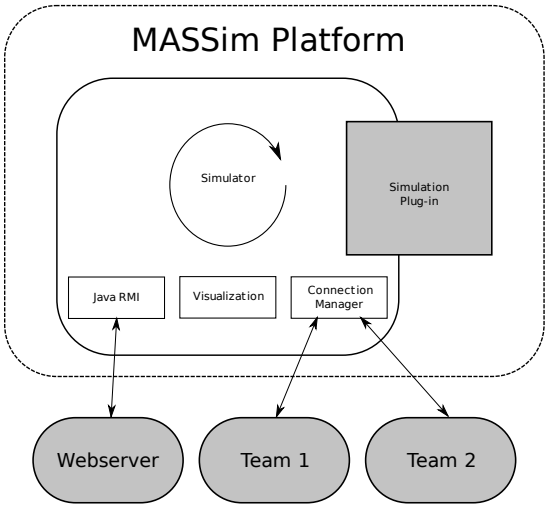

Fig. 1. The massim infraestructure.

The *MASSim* package is fully open-source and openly available (`https://multiagentcontest.org/downloads`). It is not only used for the Contest, but

has also proved useful both for researchers testing their advancements in the field, and in several classrooms, aiding the teaching of the multi-agent programming paradigm (`https://multiagentcontest.org/massim-in-teaching`).

To further ease the development of agents, the *MASSim* package includes EIS (`http://sf.net/projects/apleis/`), which is a proposed standard for agent-environment interaction. It maps the communication between *MASSim* and the agents (sending and receiving XML-messages to Java-method-calls and call-backs). On top of that it automatically establishes and maintains connections to a specified *MASSim*.

2.2 Previous scenarios

The scenario used for the first edition of the Contest (2005) consisted in a simple grid in which agents could move to empty adjacent spaces. Food units would appear randomly through the simulation, and the objective was to collect these units and carry them to a storage location. This rather simplistic scenario had to be implemented in its totality by the participants.

The idea was refined for the second edition: *Gold Miners*. Now the agents were to collect gold in a competitive environment against other team, and some obstacles were introduced in the grid to add some navigation complexity. This scenario, which was also used in the third edition of the contest, was still very simplistic, and agents acted independently of their teammates, in the solutions proposed.

For the 2008 edition, a new scenario was designed to enforce coordination of agents: *Cows and Cowboys* (Figure 2). Still using a grid as the underlying map, the goal for this scenario was to lead a group to a particular area of the map, representing the team's own "corral", while preventing the opponent team from doing the same. The cows were animated entities that reacted to the agents' positions by trying to avoid them, so solving the map required agents coordinating their positions in order to lead big groups of cows into the corrals, whereas a single agent would in most cases disperse the group of cows and fail to lead them in the desired direction.

The "Cows and Cowboys" scenario was used also in the following two editions (2009 and 2010), with further refinemets such as the addition of gates that required explicit coordination: one agent had to stand in a particular position to keep the gate open while a teammate passed through.

2.3 The Agents on Mars scenario

The Agents on Mars scenario introduced in 2011 and still in use for the 2014 edition was an important step in the contest's evolution, as it introduced many innovative features and increased the game's complexity. The map now takes the form a weighted graph representing the surface of Mars. The agents represent *All Terrain Vehicles* of different kinds, and their goal in the game is to discover the best water wells by exploring the map and then to keep control of as many wells as possible, by placing themselves in specific formations that ensure a covering

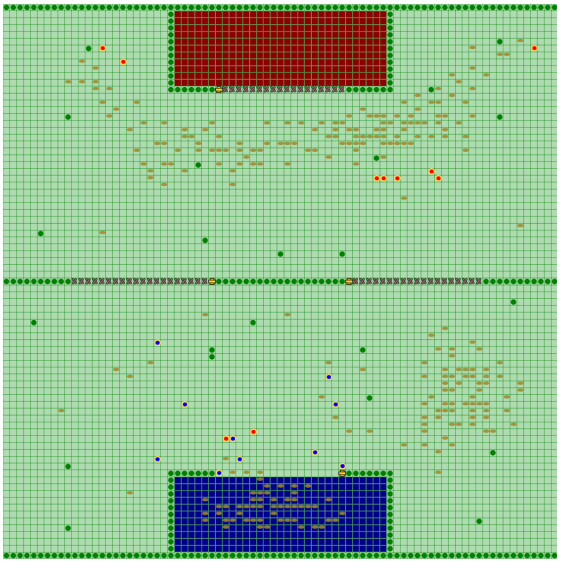

Fig. 2. The Cows and Cowboys scenario.

of an area containing the wells while keeping rival agents aside. Figure 3 shows a screenshot of this scenario.

The agents in this scenario assume different complementing roles, promoting both autonomy and coordination. For example, *Explorers* are in charge of discovering the water wells, while *Saboteurs* can attack agents of the opposing team to temporarily disable some of their capabilities. *Repairers* are responsible for restoring its teammates capabilities when they have been attacked, in a coordinated manner. All roles must collaborate to produce the best map coverings.

These agents are much more complex entities than in the previous scenarios. They have now a rich set of actions to choose from, in contrast with only moving around the map. Furthermore, they count with a set of internal parameters that can vary through the simulation—*Energy*, *Visibility Range*, *Health* and *Strength*—that can affect the choice of available actions: almost every action that an agent can perform has an associated energy cost; once an agent's energy level reaches 0, the only action it can successfully execute is the *recharge* action. If an agent is attacked by a rival saboteur, it becomes *disabled* and cannot execute its role-specific actions until it is repaired.

Another important feature that was introduced with the Agents on Mars scenario is the concept of *Achievements*. By reaching certain predefined milestones (e.g. controlling an area is worth a certain amount of points), teams earn *Achievement points*. These can be used in two different ways: either they are kept to directly contribute to the team's score, or they can be used as currency to exchange for improvements to the agent's internals.

The evolution in the complexity of the scenario has remained on a par with the evolution of multi-agent programming technologies used by the participating

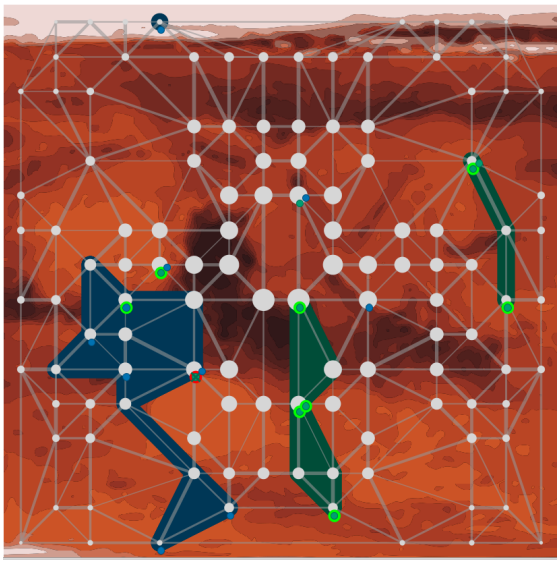

Fig. 3. The "agents on Mars" scenario.

teams. A good quality of the teams has been reached, that ensured interesting games. Unlike previous scenarios, a strategy that works against every rival has proven harder to find, and thus the winners are not unbeatable.

2.4 The next scenario

While the 2014 edition on the Multi Agent Programming Contest has just taken place once again using the Agents on Mars scenario, we are already considering ideas for a completely new scenario for the next edition in 2015. A very promising possibility, for which some research has been made in other projects, is a traffic-simulation kind of scenario, we intend to use map information from real cities. The actual game to be played in this map is still to be refined.

3 Lessons learned

In this section we will take a deeper look at some observations that we realised during ten times of hosting the Multi-Agent Programming Contest [5,4]. Most of them led to improvements of the contest platform or the employed scenario. Many lessons we learned are related to *engineering* issues (as opposed to *scientific* ones). For example, collecting statistical data or visualizations turned out to be as important as the choice of the scenarios.

3.1 From gold miners to herding cows

The first lesson we had to learn reaches back all the way to the first Contest in 2005. The agent implementations had to be submitted as an executable system

and were run locally on the Contest platform. It became clear that a standard technical infrastructure had to be provided in order to ensure a fair and objective evaluation of the agent systems and to relieve the participating teams from having to deal with low-level implementation details. Instead they should focus on the internal logic of their agents. This finally led to a separation of the scenario and the agent implementation platform, further trying not to impose unnecessary constraints on the participants systems.

As already mentioned, we wanted to see agents make use of their distinct capabilities. It became clear that those features had to be explixitly elicited. Of those, the two most important were the following:

Cooperation: Clearly, a multiagent platform or framework should be unrivalled in providing cooperating entities. While it was first believed that agents would cooperate automatically in order to achieve better results, the first editions of the Contest proved the contrary. The food gathering and gold mining scenarios were quite simple and easy to handle by individual agents. Thus, the subsequent scenarios were designed to *enforce* rather than just encourage cooperation, by making it impossible for the agents to win without seriously coordinating their actions.

Autonomy: Another feature that was not especially required in the first editions was the autonomy of the agents. It was very much possible to have a central agent deciding and coordinating the actions of all the other agents by itself. This, of course, contradicts an agent's basic characteristics. This shortcoming has been alleviated to some degree by increasing the number of agents and the size of the respective scenario, which made it less feasible (or almost imossible) for one single agent to handle all the information by itself.

From a more technical viewpoint, the Contest has clearly shown that tools for debugging and testing agent platforms are very important during development. Indeed, the participants of the first editions of the Contest were more concerned with debugging rather than with devising good strategies. This became clear to the participating teams and the agent programming community in general, making it possible and inviting to put more effort into simplifying those tasks.

Lastly, we realised that the visualization and playability of the respective scenario is a key to reaching a broader audience, especially students, e.g. when *MASSim* is used in teaching in various courses all over the world.

3.2 Mars scenario

Employing the Mars setting, we were again able to obtain a multitude of results and observations, mostly regarding (1) the usage of multiagent platforms, (2) the scenario, and (3) several technical issues.

Additionally, we now learned a lot from *interviewing the participants*, and *gathering statistical data*.

Usage of multiagent platforms. Employing the Mars domain, we noted an increasing application of multiagent platforms, i.e. starting with 33% in 2011 and up to 80% in 2013. Also, this scenario has always been won by a dedicated agent platform and those dedicated platforms seemingly outperform "ad-hoc" solutions. The presented agent solutions get better from year to year, although the complexity of the scenario is ever increasing. On the one hand, this can be attributed to some teams taking part repeatedly, but it also points to an increasing maturitiy and ease of use concerning multiagent platforms.

Scenario. We saw more coordination within the respective agent teams and of course more interaction with the opponent teams, which tells us again that the scenario has to be clearly designed to enforce cooperation and interaction.

Technical issues. As identified earlier, debugging is a key problem in multi-agent programming. Thus, in the second and third edition of this scenario, we had to work on our side of the Contest as well and improve the visualization and feedback that was sent to the agents. This made it easier to at least grasp what was going on in a simulation, maybe hinting where to start the debugging of the agents.

Asking the participants. By requiring the participants to answer a predefined questionnaire [1], we tried to learn not only about the final agent platforms and the results they produced, but about the whole development process. For example, we learned why teams participated in the first place. For many, the motivation was to learn about multiagent systems or to refine their programming skills concerning them. A lot of teams furthermore shared our goal of evaluating multiagent frameworks and platforms. Regarding their structure, teams were composed of students as well as researchers with their background mostly in MAS or at least artificial intelligence in general.

We also asked the teams how difficult it was and how much effort had to be put into getting to a point where their system behaved as it finally did. We got very diverse results, reaching from 150 to 840 person hours and 1000 to 11000 lines of code that had to be written, tested and debugged. This clearly hints at varying levels of usability concerning different agent platforms.

Furthermore, teams noted that they not only debugged their agents but found and fixed bugs in the agent framework or platform they used as well, which shows that the Contest plays an important role concerning the development and evaluation of different platforms. Nevertheless, the teams are still not satisfied with the state-of-the-art debugging tools, since it still requires a lot of effort to debug even 20 agents, each with its own individual mindset.

Not always apparent from the simulation results, we further learned

- which strategies the participants intended to use,
- how they employed different frameworks, and
- how they implemented different features of agent-based systems.

Also, the teams were able to tell us which development tools they used to which extent. The most time-consuming task in development still was debugging, or, if that worked out rather well, coming up with a good strategy.

Gathering statistical data. Lastly, we implemented a new module for the Mars scenario that allowed us to collect a multitude of statistical data for better and faster analysing a simulation once it was completed. Using these data, we can easily retrace a whole simulation's progress by looking at the automatically generated charts instead of watching the whole replay, which can be quite tedious at times. The charts mainly focused on scenario-specific data, like the development of the score or stability of dominated zones. Furthermore, we were finally able to directly and easily compare different simulation runs without having to keep a lot of details in mind. This showed that better tools for analysing on our side of the Contest were as important as better debugging tools for the participants.

For example, in Figure 4, the zone scores of the teams UFSC-SMADAS and LTI-USP from their third simulation in the 2013 edition of the Contest are given. One can see that both teams overall managed to increase the size of their zones. Starting at around step 200, it seems that the USFC team (depicted in green, having a spike there) gained control of a zone that was formerly dominated by LTI-USP which suffered a setback with the same size of the spike of USFC. However, the exchange of control seemingly did not last long, since both scores quickly return to their original values. One can now easily confirm such an assumption by directly jumping to the right point of the replay.

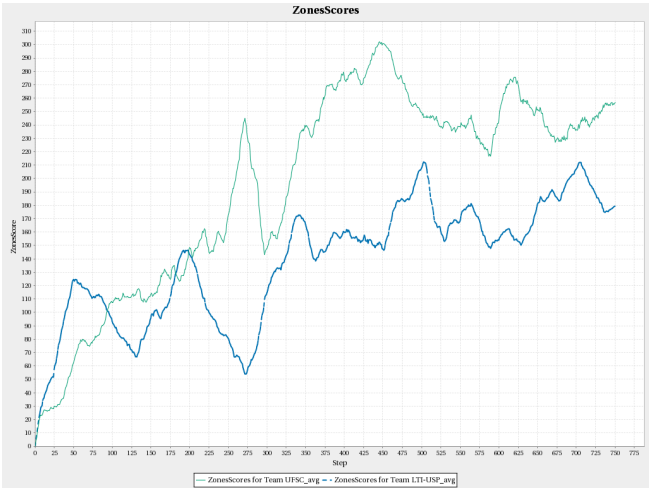

Fig. 4. UFSC-SMADAS vs. LTI-USP Simulation 3: Zone Scores.

As we have seen, the Mars scenario follows the tradition and improved further on well-tried concepts while confirming observations made in earlier contests.

3.3 The *MASSim* platform

Our platform served well over the course of many years in evaluating different agent platforms and solutions. Especially, developing the EIS standard and accompanying EISMASSim implementation [6,9] helped in easily introducing agent platforms to the *MASSim* platform.

However, the platform showed certain shortcomings as well. For one point, it is completely implemented in Java, which is known not to have the performance of e.g. C++. Additionally, it is difficult, if not impossible, to efficiently parallelize it and run it on a high performance cluster.

Another bottleneck can be found in the network traffic with its relatively high communication overhead. This was even a problem for some participants with less favorable internet connections.

From the point of view of the organizers, decoupling scenario and agent implementation made possible to exchange scenarios much more easily. However, the scenario is still hard-coded in Java and hard-wired to the simulation platform, which, again, makes implementing a new scenario a rather time-consuming task.

All in all, our long-term goal is to evolve *MASSim* into a platform that overcomes all these drawbacks. Creating a high performance platform would allow us to finally analyse and compare agent platforms with respect to their scalability. One step to reach the goal could be to establish a standard, e.g. in the spririt of EISMASSim, that enables all agents, and thus the whole Contest, to be run once again locally on our server. This would certainly free the participants from having to restart crashed agents during a simulation. It would also introduce robustness as a new and important requirement for multiagent systems in order to participate.

4 Large-scale simulation: Maserati

We have seen in the last few years that *MASSim* is a stable platform able to coordinate up to a hundred agents quite efficiently (and remotely over the net). Unfortunately, for reasons that we discuss below, it is not possible to extend *MASSim* to handle tens of thousands of agents (or even more). However, in massive simulations, eg. in traffic, energy or logistics, such numbers of agents are easily reached. *Scaling a microsimulation approach (like MASSim) to large scenarios is still a challenge.*

Nowadays massive simulations, eg. in traffic simulation, are undertaken with classical analytical models. These models only allow to deal with global properties, like the throughput or flow-rate. An example is the commercial traffic simulation platform *AIMSuN*.

Why can't we simply integrate *AIMSuN* with an agent programming platform? This has been undertaken in [7] where it was tried to integrate *AIMSuN*

with the well-known agent platform *JADE*. However, experience showed that this is extremely difficult and does not work without having access to the source code of the commercial software product (which is not available in most cases).

So we are left with two extreme approaches:

Micro-view: a massive multiagent based simulation, where each entity is an agent, or

Macro-view: a commercial product based on analytical models and describing global properties of the system.

The idea of *MASeRaTi* is to develop a simulation platform which is inbetween: it supports both a *micro-view* as well as a *macro-view*. Ideally, the designer should be able to *zoom in* and turn particular parts of the system into agents (if such a detailed view is needed), or to rely on global parameters for other parts (there are some similarities to the notion of a view in databases). Often there are questions where the additional overhead to deal with a micro-view is not needed. Thus it would be appropriate not to be forced to deal with it.

MASeRaTi, currently being developed in the DeSIM project[8] at TU Clausthal, is a *distributed MABS platform* that aims at high scalability for *networked* simulations of *systems-of-systems*, e.g. in traffic and transport. It will be capable of running simulations containing a vast number of software agents.

This section should give the reader a bird's eye view of *MASeRaTi*. For details we refer to [2] and [3] (where parts of this section were taken from).

The *MASeRaTi* platform combines several promising features from different areas in one integrated system.

Architecture: Its architecture, in particular communication and the simulation cycles, are inspired by the architectures of massively multiplayer online role-playing games (MMORPG): All simulation objects are split into two disjoint sets, synchronized and non-synchronized objects. Synchronized objects are for instance the simulation world (also called area) or objects situated within, because these objects must be consistent over all nodes of the HPC. Other objects, e.g. agents, are defined as non-sychronized objects, allowing to be transferred to other nodes.

Scalability: We use high-performance computing algorithms with the message-passing-interface (MPI), so that we can use scalable structure with a high performance datalink between the cluster nodes (in the future, a P2P overlay can be used). Scalability is achieved by splitting the simulation objects into disjoint sets, so that we can design a distributed system with an optimization process for calculation.

Lua: We define an abstract agent model for an agent programming interface in Lua[9], which can be extended or fully redefined by the programmer.

The overall *MASeRaTi* architecture consists of three layers, see Fig. 5:

[8] `http://simzentrum.de/en/projects/desim`

[9] `http://www.lua.org/`

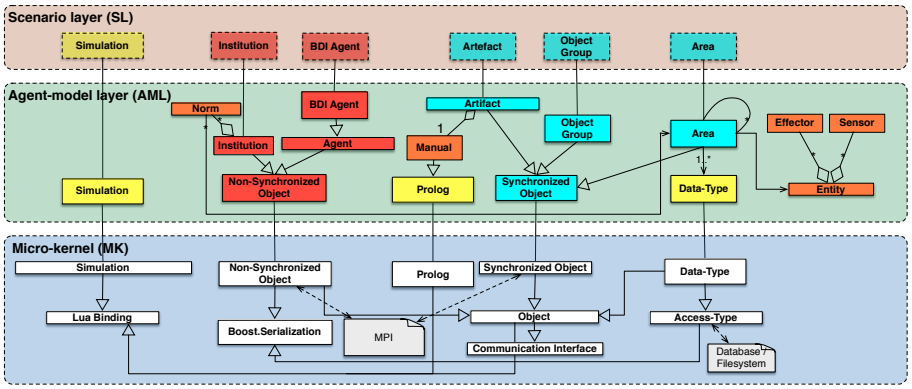

Fig. 5. Overview of the *MASeRaTi* architecture

Micro-kernel (MK): Written in C++, this layer facilitates parallelization over a HPC using structure, scheduler, scaling and optimization features of the message passing interface (MPI) library. A plug-in interface allows to replace MPI by alternative communication technologies like BitTorrent.

Agent model layer (AML): The agent model layer defines an object-oriented model of a multiagent based simulation. Micro-kernel classes and objects are mapped into this layer, extending the existing structures. Lua is used as a modeling language because of its flexibility (imperative, object-oriented and functional programming) and its property of being interpreted at runtime.

Scenario layer (SL): This layer defines instances of an AML, adding domain-specific entities and behavioural models, e.g., for traffic simulation.

The reason for using Lua as the modeling language in the AML is twofold: Firstly, it has a very small interpreter (around 100kByte) written in native C. Secondly, C/C++ data structures can be pushed into the Lua interpreter at runtime with a native pointer structure, so we can easily extend Lua. The linkage between MK and AML is defined by Lua binding frameworks, e.g. Lua Bridge[10].

Finally, the simulation layer implements a simulation as an instance of the AML. A native Prolog interpreter is provided for reasoning tasks (e.g., for the belief base). One can also store Lua functions in it. Area structures like graph or grid systems can be added with the `Data-Type` interface. Such a data type models a certain structure (like a grid, a graph etc.) and implements the corresponding search algorithms such as Dijkstra's, $A^\star$ and $D^\star$.

The process of engineering (i.e., modeling and running) a simulation is geared to exploit the structure of the *MASeRaTi* platform. The platform itself runs on a HPC system enabling large sets of experiments. The steps of the process are illustrated in Fig. 6.

After each iteration, the developer should be able to test her prototype by creating a request for computation (Step 4). The HPC system instantiates this

[10] `https://github.com/vinniefalco/LuaBridge`

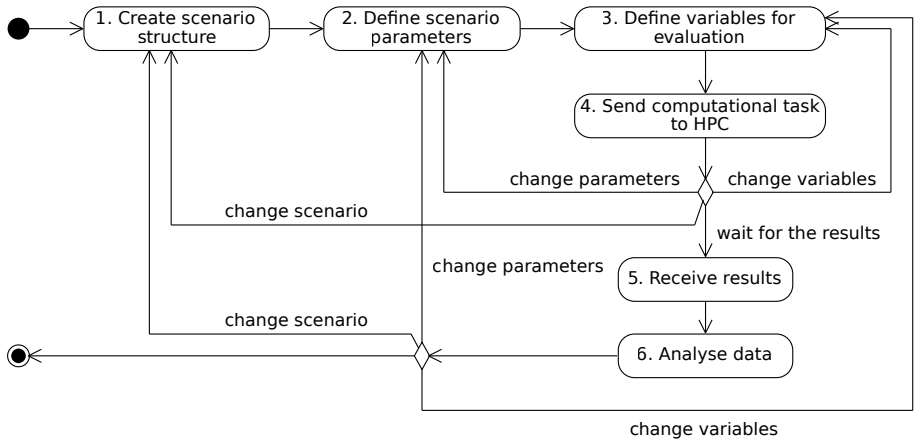

Fig. 6. Simulation engineering process

task, creates child processes and calculates the outcome. While the simulation is running, one can create another instance or a completely new scenario and add the task to the queue. Finally, in Step 5 and 6 the evaluation data of a simulation is processed by the client and additional analyses are made.

Due to the revision control system of the database, a scenario can run with different parameters and input data and the resulting datasets can be compared. The architecture is split into client and server parts, which communicate via the database. The database stores all scenario data into a repository, so the full developing process is being logged. A task can be seen as a current state of a repository with fixed parameters and input data. This mechanism enables the possibility to supervise and summarize the results of different tests.

Figure 7 sketches a user interface to support the simulation engineering process. The interface will be realised by techniques used in today's web browsers, so that each user can add modeling or analysing features to the system. The client, which can run small simulations, uses Qt QML [11] to create a browser interface. We plan to add components for visualization like `Data-Driven Documents`[12] or `Chart.js`[13].

5 Conclusions

An important outcome, or rather insight, of the second author's work on logic programming and knowledge representation in the 90'ies is the following. While basic research flourished and produced many important results on the relation between various formal systems and on the complexity and expressivity of many

[11] `http://qt-project.org/doc/qt-5.0/qtqml/qtqml-index.html`
[12] `http://d3js.org/`
[13] `http://www.chartjs.org/`

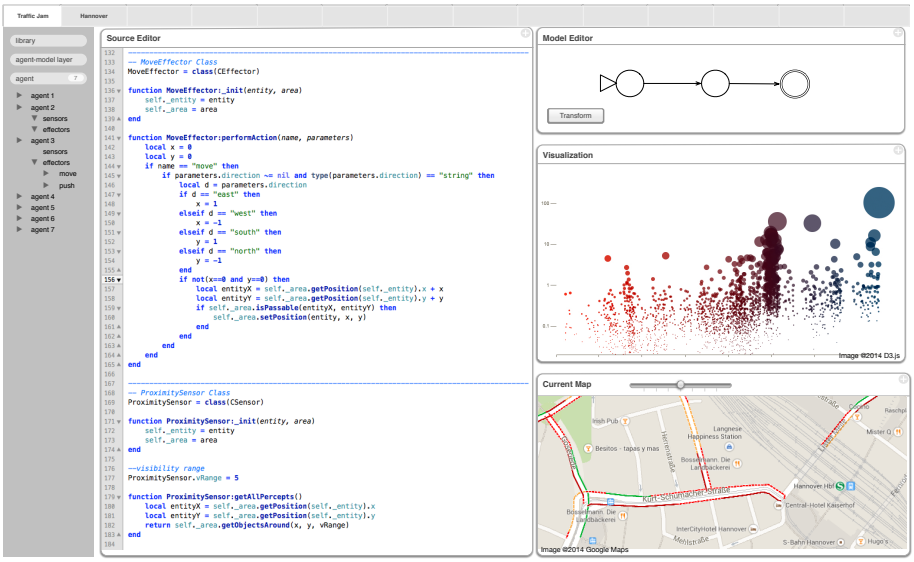

Fig. 7. UI Wireframe

semantical systems, it did not account for developing a methodology to apply these systems to the real world (or at least to nontrivial applications). The reason is simply that such methodologies are by many considered not *scientifically valuable* and thus it is difficult to get publications out of such work. The *engineering component*, which is invaluable for a potential serious implementation of a running system, is time-consuming yet the scientific content is low (given that the original theoretical results have already been published).

But the uptake of basic research in industry heavily depends on well-developed methodologies and seriously crafted software systems (as opposed to prototypes developed within PhD projects). Building such systems requires many person years and is almost never done within an academic environment.

The shift from logic programming semantics to *answer set programming*, seen as a paradigm to encode problems on the second level of the polynomial hierarchy and solve them with appropriate solvers, was of utmost importance. But without applications and ASP systems developed and improved along such applications it would have been nothing but an academic toy.

There are some similarities to the area of agent programming. As mentioned above, in the first few years agent programming languages developed in academia did only have premature (if any) debugging tools (and many more classical software tools were missing). The Multi-Agent Programming Contest helped, on a modest level, to improve some of the languages. But it addressed only relatively small problems/scenarios. As in the case of ASP, we need more *engineering* and we must take scalability seriously. This is what we have tried to do with *MASeRaTi*.

With an initial version of *MASeRaTi* including first scalability tests being available [2], future work in DeSIM will focus on optimizing the platform and increasing its runtime performance, e.g. by a more flexible distribution model, and by more sophisticated agent scheduling algorithms.

A key activity in this respect would be supporting collaborative modeling done by distributed teams of modelers including appropriate methodologies, tools, modeling abstractions, and libraries.

References

1. Tobias Ahlbrecht, Christian Bender-Saebelkampf, Maiquel de Brito, Nicolai Christian Christensen, Jürgen Dix, Mariana Ramos Franco, Hendrik Heller, Andreas Viktor Hess, Axel Heßler, Jomi Fred Hübner, Andreas Schmidt Jensen, Jannick Boese Johnsen, Michael Köster, Chengqian Li, Lu Liu, Marcelo Menezes Morato, Philip Bratt Ørum, Federico Schlesinger, Tiago Luiz Schmitz, Jaime Sim ao Sichman, Kaio Siqueira de Souza, Daniela Maria Uez, Jørgen Villadsen, Sebastian Werner, Øyvind Grønland Woller, and Maicon Rafael Zatelli. Multi-agent programming contest 2013: The teams and the design of their systems. In Cossentino et al. [8], pages 366–390.
2. Tobias Ahlbrecht, Jürgen Dix, Michael Köster, Philipp Kraus, and Jörg P. Müller. A scalable runtime platform for multiagent-based simulation. Technical Report IfI-14-02, TU Clausthal, February 2014.
3. Tobias Ahlbrecht, Jürgen Dix, Michael Köster, Philipp Kraus, and Jörg P. Müller. A scalable runtime platform for multiagent-based simulation. In Fabiano Dalpaiz, Jürgen Dix, and Birna van Riemsdijk, editors, *Engineering Multi-Agent Systems - Second International Workshop, EMAS 2014, Paris, France, May 5–7, 2014, Revised Selected Papers*, volume 8758 of *Lecture Notes in Computer Science*. Springer, 2014.
4. Tobias Ahlbrecht, Jürgen Dix, Michael Köster, and Federico Schlesinger. Multi-agent programming contest 2013. In Cossentino et al. [8], pages 292–318.
5. Tristan M. Behrens, Mehdi Dastani, Jürgen Dix, Michael Köster, and Peter Novàk. The multi-agent programming contest from 2005-2010 - from gold collecting to herding cows. *Ann. Math. Artif. Intell.*, 59(3-4):277–311, 2010.
6. Tristan M. Behrens, Koen V. Hindriks, and Jürgen Dix. Towards an environment interface standard for agent platforms. *Ann. Math. Artif. Intell.*, 61(4):261–295, 2011.
7. Viet-Hung Chu, Jana Görmer, and Jörg P. Müller. ATSim: Combining AIMSUN and Jade for agent-based traffic simulation. In *Proc. 14th Conference of the Spanish Association for Artificial Intelligence (CAEPIA)*, volume 1. AEPIA, 2011. Electronic Proceedings.
8. Massimo Cossentino, Amal El Fallah-Seghrouchni, and Michael Winikoff, editors. *Engineering Multi-Agent Systems - First International Workshop, EMAS 2013, St. Paul, MN, USA, May 6-7, 2013, Revised Selected Papers*, volume 8245 of *Lecture Notes in Computer Science*. Springer, 2013.
9. Koen Hindriks and Jürgen Dix. Goal: A multi-agent programming language applied to an exploration game. In Onn Shehory and Arnon Sturm, editors, *Research Directions Agent-Oriented Software Engineering*, pages 112–136. Springer, 2013.
10. Gerhard Weiss. *Multiagent Systems*. The MIT Press, 2013.