Department of Computer Science
George Mason University Technical Reports

4400 University Drive MS#4A5
Fairfax, VA 22030-4444 USA
http://cs.gmu.edu/ 703-993-1530

Future MASON Directions: Community Recommendations

Report of the 2013 MASON NSF Workshop
June 15–16, 2013, George Mason University Campus, Fairfax, Virginia

Nicolas Payette	Northwestern University
Marius Bujorianu	University of Birmingham
Glen Ropella and Ken Cline	Tempus Dictum
Jeffrey Schank and Matt Miller	University of California Davis
Sara Jonsson	Royal Institute of Technology, Sweden
Laszlo Gulyas and Richard Legendi	AITIA, Hungary
Olaf Bochmann	Oxford University
Luís de Sousa	Technical University, Lisbon
Vlasios Voudouris and Daniil Kiose	London Metropolitan Business School
Przemyslaw Szufel	Warsaw School of Economics
Steve Saul	NOAA
John McManus	University of Miami
Vittorio Scarano and Gennaro Cordasco	University of Salerno
Chris Hollander, Paul Wiegand, and Vera Kazakova	University of Central Florida
Brian Hrolenok	Georgia Tech
J. Daniel Rogers	Smithsonian Institution
Michael Schader	Yellow House Associates
Sean Luke, Kenneth De Jong, Mark Coletti, Paul Schopf, Claudio Cioffi-Revilla, Keith Sullivan, Khaled Talukder, Ahmed Elmolla, and Ermo Wei	George Mason University

Technical Report GMU-CS-TR-2013-9

Introduction

MASON is an open source multiagent simulation library geared towards simulating very large numbers of relatively lightweight interacting agents. MASON has been used for a wide variety of simulation tasks in robotics, the social sciences, biology, and animation.

On June 15 and 16, 2013, approximately two dozen invitees convened at George Mason University to discuss future directions for MASON and needs of the MASON community. This meeting formed the 2013 MASON NSF Workshop, sponsored by the National Science Foundation under CRI CI-P grant 1205626. The invitees responded to a call for participation on the MASON community mailing list, among others, and those selected for participation came from a broad spectrum of fields, organizations, and countries.

During the Workshop, participants identified nine areas of interest to the community where they felt MASON should be improved or extended. The participants then divided up into working groups to discuss issues with MASON improvements in those areas. This document details the reports of those working groups. The nine areas, and their basic proposals, were:

1. **Language and Development Support** Proposals: to enable MASON to be accessed from non-Java languages; and to develop plugin support for MASON in integrated development environments such as Eclipse or NetBeans.

2. **Output and Statistics** Proposals: to develop a plug-in architecture for simulation statistics and improved run-time analysis of models.

3. **Parallel and Distributed MASON Models** Proposals: to develop distributed facilities in MASON, and to examine extensions to MASON to better use massive multicore and GPU facilities.

4. **Modern Java Facilities** Proposal: to add various post-1.5 Java features to MASON.

5. **Testing** Proposal: to add unit and integration testing regimens to MASON.

6. **MASON and STEM** Proposal: to identify and develop facilities which make MASON more useful in a teaching context, with an eye towards STEM fields from Kindergarten clear through undergraduate.

7. **Collaborative Archives and Facilities** Proposal: to develop a facility (perhaps a website, or via MASON integration) for the community to share MASON models and code snippets to further community interaction and support.

8. **Validation via Optimization and Parameter Sweeping** Proposal: to develop optimization facilities for MASON models and for internal MASON agents, and facilities to enable large parameter sweeps of complex MASON simulations.

9. **MASON and GIS** Proposal: to improve GeoMASON's performance and its network support; and provide integration with distributed MASON facilities.

Following are the reports of the working groups in each of these areas.

1 Language and Development Support

Working Group Nicolas Payette, Matt Miller, Chris Hollander, Vera Kazakova, Ken Cline, Luís de Sousa, and Glen Ropella

Skilled workers in the near future will need to be able to program computers for a wide variety of tasks, from setting up computer-assisted machining to creating engineering software for scientific and industrial endeavors. Agent-based models are well-suited both to applications in many fields and to the instruction of those who will write software in the future. MASON is well-placed to address current use-cases for agent-based models and the education of tomorrow's programmers. However, there is a two-way barrier of domain specialization that must be overcome to optimize MASON's usefulness in these fields. Much of MASON's power accrues from its flexibility, but this flexibility requires explicit coding of basic structures assuming domain expertise with modeling, graphics, and MASON itself; nonetheless, much of this substructure often amounts to "boilerplate" code. On the other hand, agent-based models often provide elegant solutions to problems in a wide variety of disciplines, but the domain expertise within these disciplines is often limited to specific programming languages such as Python, R, MATLAB, or Lisp.

Proposal Two overarching goals will improve MASON's utility both as an applied and as an educational tool. We propose adding multiple language support to MASON and providing advanced development/integrated development tools for MASON.

Multiple language support entails providing bridges between MASON and other programming languages. This will be accomplished by providing interface libraries for languages that are compiled to run on the Java virtual machine (JVM), such as Jython (Python for JVM), Scala, JRuby (Ruby for JVM), Kawa (Scheme for JVM), Clojure (Lisp for JVM), and Rhino (JavaScript for JVM). This will allow programs to directly use MASON's methods with minimal implementation obstacles.

Development tools would encompass plugins for the Eclipse and NetBeans integrated development environments. Plugins would automate the production of boilerplate code and common coding tasks within MASON such as event engine setup, spatial representations, graphical portrayals, and data output facilities. These code generators would include customizable fields as well as opportunities for the developer to idiomatically code simulation elements that did not meet standard application parameters.

Justification Multiple language support brings domain expertise from multiple disciplines to MASON. Domain experts often are familiar with programming only in the language preferred in their domains; for example, psychology researchers frequently have exposure only to R and MATLAB. Furthermore, well-established models and libraries for specific disciplines are often written in a specific language; for example, some physics libraries rely on NumPy, which is written in Python. Finally, some users may need to use modules written in languages other than Java to achieve their goals; for example, one author of this white paper reports, "My model of liver function includes a module written by EPA developers in R. Any changes in that module render it non-functional."

Support for other languages and development tools both improve MASON's utility as a teaching tool. Agent-based models are often used to teach students in the sciences about programming and about emergent dynamics. The presence of individual software agents, as instances of encapsulated classes, often can be used to clarify concepts in object-oriented programming, communication between modules, and the emergence of unexpected patterns from non-linear dynamics. However, not all programming classes are taught using Java; thus, multiple language support would increase MASON's applicability to a wider range of instructors. Further, MASON models achieve power and flexibility by allowing users to implement highly customized spatial structures, scheduling mechanisms, graphical interface elements, and graphical representations of model runtime; however, the code for these structures for basic models and for almost any model as a teaching tool are basic themes on a pattern. This code, though following basic patterns, is generally beyond the ability of the beginning programmer; therefore, a development tool that simply provides the basic background structure would improve MASON's usability as a teaching tool in beginning stages. One white paper author relates, "When I teach introductory classes on modeling, the boilerplate code gets in the way of the essence of models and algorithms. Only after I've taught these basics, often including general programming, can I bootstrap a basic understanding of the setup code and graphics configuration."

Development tools will also accrue benefits to those implementing models for research purposes. Producing template code automatically will increase development speed and reduce opportunities for errors in basic model machinery. Furthermore, template code will provide more consistent models with more consistent interfaces and output. Finally, graphical development tools will allow contributions from team members with little or no programming experience and will facilitate model explanation to those lacking time, motivation, or ability to review the code in detail.

2 Output and Statistics

Working Group Mark Coletti, Paul Wiegand, Sara Jonsson, Vlasios Voudouris, and Daniil Kiose

Currently MASON provides a variety of mechanisms with which to display, monitor, and track quantitative values of simulations occurring within the system. These include visual monitoring of individual values using the GUI interface *inspectors* tied to simulation objects, graphical plotting of values via programmatic access to the JFreeChart API, and simple textual output to the standard output stream or a file.

However, there are severe limitations to these approaches. For one, JFreeChart has limited capabilities and does not produce production level plots, the inspectors are not useful for aggregate and statistical mechanisms, and almost any serious statistical output and reporting requires at least some additional programming above and beyond the programming of the simulation. Additionally, these existing mechanisms focus mainly on individual simulation runs and do not make it easy to collect aggregate information between multiple runs of the simulation.

On the reverse end, there is a desire from some users that MASON have better facilities for *interactive* simulation — for use of MASON as a *computational laboratory.* Inspectors are the only extant mechanisms for any kind of *interactive* access to simulation data during a run, but these are not sufficient for a deeper analysis of what is happening.

The majority of users with which the working group members are familiar (including themselves) choose to produce quantitative output to a text file or a database, where it is post-processed by some other tool (e.g., R) for statistical analysis and data visualization purposes. Indeed, this is so common that many users have written general classes for data output purposes in a lot of different simulations. This has been done, for example, by inserting *Steppable* objects into the scheduler whose sole function is to collect and report quantitative data about the simulation. Unfortunately, many different MASON users have designed and implemented their own individual versions of these kinds of facilities, which is wasteful of effort and unhelpful to newer MASON users.

This working group suggests the development of a plug-in architecture that provides users with more facilities regarding common workflows within MASON in terms of the collection and output of quantitative information about simulations. This architecture would allow users to output data regarding a variety of individual run quantitative information, as well as data regarding aggregate information over many runs. It would also support a number of common data formats, such as greppable text files, CSV files, or various SQL databases.

To support a *computational laboratory* point of view, we suggest MASON might benefit from providing a suite of GUI controls and widgets that can be embeded directly in the simulation or interface with a known dynamic visualization tool or service, such as Processing. One goal of such an interface would be to have to ability to pause the simulation, then follow some kind of link to a service where a deeper analysis of the data might be done. Another would be to improve data visualization such as allowing the embedding of legends and similarly informative widgets into the display.

3 Parallel and Distributed MASON Models

Working Group Vittorio Scarano, Gennaro Cordasco, John McManus, Przemyslaw Szufel, Richard Legendi, and Claudio Cioffi-Revilla

The current situation of MASON is that, in several different contexts, the limitations of being run on a single host appear evident as the model increases in size and complexity. As the size of the model increases, the memory bound of a single host quickly becomes a serious limitation, and the increasing computational complexity of the agents impacts on the execution speed of the simulation. On the technological side several computing options (not necessarily tightly coupled as High Performance Computing systems) are available to researchers, using multiple heterogeneous machines capable of massive simulations. At the same time the nature of the single host is changing, including multi-core (with several tens of cores to be packed in one processor) and many-core facilities in low-end machines; and Graphical Processing Units (GPUs) widely available for generic processing.

In this heterogeneous scenario, MASON should be able to leverage the distributed/parallel facilities now available to researchers, in order to speed-up simulations (allowing more extensive testing), to enlarge the size of the model (offering access to emergent behaviors that may appear only after a threshold), to increase agents' computational complexity (allowing more intelligence to be embedded into the single agent), and to offer facilities for parameter space exploration in parallel (significantly speeding up model tuning and validation).

Complications Several problems and complications are faced with along the road to making MASON distributed/parallel. As a premise, it appears that different architectures serve differently well different kinds of simulations. Among the possible architectures, we distinguish:

- Parallel architectures, where tightly-coupled, often homogeneous, centrally managed, dedicated computers are connected with dedicated, high-speed/bandwidth networks, and dedicated, high-speed storage facilities.
- Distributed architectures, where heterogeneous, loosely-coupled computers, are possibly spread in different local networks, with no centralized control and management, and often devoted only partially to simulations.
- Many-core architectures, where a single PC is equipped with several traditional cores and with a GPU (or more than one) that provides very large number of small cores, with some limitations on the memory size but with very high performance with respect to data movement and computation.
- Hybrid architectures, which combine all the preceding three architectures.

Several issues can be identified:

- **Flexibility and the Design of Architecture-Aware Models** Depending on the requirements in size, locality of communications, and complexity of the model, different architectures may behave very efficiently or not, or may not allow massive simulations to be run at all. A parallel/distributed version of MASON must address these scenarios, providing not only technological and scientific solutions but also templates and guidelines to model developers to guide them through the perils and hardships of making a simulation that is intended to be efficiently executed in a parallel/distributed environment.
- **Partitioning** Depending on the kind of architecture, different partitionings of the simulation can be used. In general, two main categories are common from the literature: a partitioning may be space-based (depending on an agent's location) or agent-based (ignoring physical positions). The former may be preferred when local (space-based) communication occurs frequently; but if communications are global, but seem to cluster in time, the latter can be a winning strategy. Different strategies must be possible in order to deal with the inherent diversity of simulations.
- **Load Balancing** When spatial partitioning is used, it may happen that the simulation produces an unbalanced usage of computational and communication resources, overloading some hosts while leaving others significantly underutilized. In this case, methods and techniques for dynamic load balancing must be employed. Examples of problems with these problems can be as simple as ant-generation and fish schooling, or as complex as disaster recovery simulations focused on a specific region.
- **Optimization for Different Communication Patterns** Mixed communication patterns offer particular challenges. When a simulation is consistently using only one particular local or global communication pattern, it may be possible to choose an appropriate architecture to increase the performance and scalability of the simulation. But the situation is complicated when communication occurs both locally and globally, or at in different moments of the simulations. A loss in performance can possibly be mitigated if the global interactions are not frequent and a space-based partitioning is used, but *general* methods and techniques to improve the performance of a parallel/distributed simulation are a challenging problem and would be a significant achievement for the agent-based modeling simulation community. Any parallel/distributed MASON must address different fields, such as 2D, 3D grids, networks with all their possible variants and overlapping.
- **Fault Tolerance** Especially in the distributed setting, fault-tolerance with respect to partial failures of the system must be supported and integrated

with mechanisms for automatic roll-back and recovery to meaningful simulation states. Solutions to fault-tolerance can also be adapted easily to provide for dynamic changes in the hardware configuration, when nodes are made available or made unavailable during the simulation.

- **Management and Control** In such a complex setting, a parallel/distributed MASON model becomes hard to manage and control. It is important to provide an easy-to-use console to researchers, in order to facilitate the management and make sustainable the effort of controlling such a complex computational facility with moderate technical skills.

- **Leveraging Hybrid Architectures** Hybrid architectures can be efficiently leveraged by models that are architecture-aware. As an example, when Agent-Based Models are combined with cognitive modeling, the simulation may require expensive single-agent computation that could be given to a GPU while agents are distributed on various CPUs. Another example: when equation-solvers are placed on GPUs and pulled up by MASON agents from time to time.

Conclusions We think that the proposed MASON work should first identify the commonality of challenges in biological, ecological, social, and other MASON models and how they could benefit from using a parallel/distributed MASON framework. Then, the issues outlined above may be driven by the needs of the experts in these domains, leveraging existing open-source solutions and freely available libraries that include D-MASON (an initial distributed MASON attempt), CUDA, Globus, and other parallel and distributed computational frameworks.

4 Modern Java Facilities

Working Group Przemyslaw Szufel, Chris Hollander, Ken Cline, Luís de Sousa, Vlasios Voudouris, and Matt Miller

The goal of this section is to discuss the modern Java facilities and language features introduced into the Java language from the version 1.5. MASON tends to be backward-compatible with Java 1.4. We discuss how these later facilities would improve a modeler's experience with MASON and code readability.

Several motivations support the introduction of these facilities into the MASON library. Firstly the MASON should evolve along its surrounding world — and new Java facilities are nowadays common in Java programs. New students are being taught Java with generics and annotations being there and they expect it in MASON. The new language facilities will also make it easier to integrate MASON with GUI tooling and will allow for easier integration with data import-export tools.

We believe that the following three types of new Java features should be included in the MASON framework.

- Generics
- Annotations
- enums

The document is divided into three sections — each section corresponding to discussion of a particular Java feature and its impact on the MASON library

Generics MASON does not use generics at all. Currently the explicit type casting occurring right now could be easily avoided through use of the diamond operator. This change can be carried out while preserving full backward compatibility in the MASON library and will greatly increase code readability. This change will require a rewrite of several MASON components including simulation engine and tooling libraries. However the change will be backward-compatible and will enable more concise and readable source code.

Annotations Annotations are another new feature introduced in Java 1.5, and MASON does not use them. Annotations can play important role in extending the MASON framework and in increasing developer productivity. The following areas can be enhanced by annotations in MASON:

- **Describing Java bean objects for the GUI** MASON employs an extension of the Java Bean protocol (set*Foo(...)* and get*Foo()* or is*Foo()*) to add additional functionality for inspection widgets. For example, MASON has a "dom*Foo()*" method which returns the domain of legal values with which Foo may be set. This can be used to define a slider or a pop-up menu which constraints possible user inputs to the Foo property. MASON has a number other gizmos as well. Currently in the worst case scenario up to six methods (six methods might be used to manage a single field. The introduction of annotations may lead to a cleaner code in this scenario, and code that will be less prone to developer mistakes. As Java Bean properties are very important and common in simulation visualization, we think this will significantly increase developer productivity.

- **Managing data collection processes** Annotations could easily allow one to assign data sinks to particular fields in a model and and to provide a logging framework for simulation models.

- **Model debugging** Debugging of multiagent simulation models is a complex multi-stage process. Simulation could be run in "debug" and "production" modes, where the debug mode triggers debugging annotations to be processed.

Enum support Enums are an example of various small-scale Java changes which MASON could support better at this stage, and unlike annotations or generics, many of them can probably be addressed fairly quickly.

As to enums: the current object inspector in MASON does not support the use of enums. This necessitates coding fragile getter and setter methods that involve using an integer variable. It would be preferable if the inspector natively handled enums. The get method currently operates as might be desired (returning the `toString` value for the enum) but enums are not supported for set methods. A set method that presented a JComboBox selection field would be desirable, both to allow for more robust coding of getter/setter methods and to avert the need of writing a separate getter method when the enum return type is desired. Additionally, extending this functionality with an optional dom*Foo()* facility (see earlier) to present friendlier strings for the JComboBox could be implemented.

5 Testing

Working Group Glen Ropella, Richard Legendi, and Daniil Kiose

Software is distinct from other engineered artifacts in that the languages used to form each layer of the stack are highly expressive. From general purpose hardware, through compiler back-ends, all the way up to the structures assembled by the end user, the opportunities abound for misunderstanding a component or allowing anomalous behavior to slip by unnoticed. MASON has been constructed according to very tight specifications. However, as yet, no automated testing infrastructure has been provided to ensure and advertise that each component adheres its specifications. Herein, we propose the development of a multi-scale testing infrastructure to cover both atomic and composite MASON components.

What is Proposed The tests for many components will be straightforward to implement, e.g. MutableDouble, Schedule, and neighborhood lookup operations. Particularly difficult to test are graphical visualization components like Portrayals and Movie Capture. Examples of testing infrastructure for such do exist, however. Netlogo takes checksums of graphical renderings for each release. Whether or not GUI interaction should be tested (with tools like Abbot or Jemmy) is an open question and any such testing will build upon the higher priority underlying test suite for the core components.

This proposal includes both internal consistency assertions and cross-release regression testing. And testing will be automatically triggered following a continuous integration server methodology where a high priority test suite is executed when code is checked into the source code repository. The test suite will be granular, allowing multiple testing patterns, perhaps including every check-in, nightly, weekly, or on demand execution of partial test suites. If such methodology proves infeasible due to project constraints, a compromise can be adopted where intensive test suites are only engaged at public releases or when new code or components are included.

Every attempt should be made to build upon standard testing tools, particularly for Java development, including JUnit, TestNG, and FindBugs. However, where such standards would interfere with established project objectives and needs, they will be abandoned or forked to provide MASON-specific functionality.

Justification Software verification provides methods to ensure adherence to a specification. MASON is designed as a hacker-friendly toolkit. I.e., it is intended to provide a modeler not only with high level constructs with which to easily construct models, but also with access to internal, perhaps obtuse, constructs that are easily used and misused. MASON provides the user with enough rope to hang yourself, as it were.

This project objective provides an interesting challenge in that a higher, more opaque wall between tool producer and tool consumer would provide tighter controls and, hence, a more straightforward testing strategy. It is easier to ensure the code lives up to its contract with the user if the tool developers maintain tight control. However, this conflicts with both the motivation behind open source and the pragmatic efficacy of open-ended, extensible tools where the user decides the most appropriate way to use the tool. As such, the testing framework will be as open and hackable as the code being tested, allowing the user to see into and build upon the testing idioms put in place by the MASON team.

This not only guarantees the usual benefits of a test suite (adherence to an implicit, evolvable specification, predictable effect and performance, and efficient development and debugging), but also provides a set of "best practices", a guideline for how MASON does testing. This helps both expert users transition to MASON, users who may be more familiar with how other tools do testing and novice users who may not be familiar with how testing is or should be done at all. In this sense, MASON's test suite will provide both a counterpart to other testing methods as well as educational content for those who are just learning.

Most importantly, the extent to which a layered toolchain can be trusted to achieve it's composite objectives depends fundamentally on the trustworthiness of each tool in that chain. Because agent-based models, the dominant application domain for MASON, are explicitly designed to target concrete, detail-oriented models of complicated and complex systems, it is critical for the trustability of those models that MASON have a transparently accessible, coherent, and complete testing framework. Such a framework will allow and facilitate explicit decision making and project tracking not only for MASON, but for any tool-chain in which MASON is used.

6 MASON and STEM

Working Group Vittorio Scarano, Claudio Cioffi-Revilla, Dan Rogers, Vera Kazakova, Sara Jonsson, John McManus, and Paul Wiegand

The United States is experiencing decreasing STEM literacy from kindergarten to university-level education. We believe that Agent-Based Models (ABMs), which are often exciting for students to build, can help reverse the trend by enhancing complex critical thinking, by encouraging multi-disciplinarity, and by making exploratory learning a common skill. Many misconceptions of scientific facts among students could be easily tackled by an exploratory approach via models and their simulation.

Complications Models in general are often intimidating for students, as they often have a steep learning curve. With a common lack of programming skills in many scientific disciplines, and the lack of multidisciplinary programs, this impacts on the popularity of modeling and simulation as a learning tool, since students (and teachers) often don't understand how models can help their students learn, especially in K–12. But we think that agent-based models can leverage kids' interests in simulation via playing simulation games, such as those many students are familiar with (SimCity, etc.). Even learning experts do not all realize that models are not used only for prediction, but are also used as a tool to understand, test assumptions, and look for missing information. ABMs may represent a 21st century way to explore the world and help understand it. Another substantial complication is the potential lack of high-quality visualization in simulations since it significantly affects the impact of models as teaching and learning aids.

Resolution We suggest developing teaching tools to help teachers to use MASON in a learning environment, in order to broaden educational usage of MASON in K–12 and to promote the benefits of modeling. MASON could be revised to fit educational needs for STEM education in order to allow learners to explore the dynamic world in first person. ABMs can be helpful in realizing that math and stats, alone, are insufficient to understand complex phenomena. By making MASON more user-friendly, i.e., by simplifying its interface, there could be a consistent effect on the STEM education, encouraging novices and young people to use (the educational version of) a professional simulation environment as teaching and learning aid.

To help the diffusion, first, there should by several ABM pilot programs in one or more high schools, in order to collaboratively design MASON around learning needs. Then, possibly, MASON developers might have a workshop and tutorials for enhancing the value of ABMs and MASON in non-modeling communities. Also helpful would be a catalogue of MASON simulations that correct the misconceptions of scientific facts,or that can be useful for motivation and engagement for learners in the class. ABMs (and MASON) could also be a way to teach both Java and ABMs to non-computer scientists' introductory programming course.

Use Cases An example use case is Thomas Jefferson High Schools, which has a collaboration with the MASON development team, and has in some cases had students contributing directly to MASON.

7 Collaborative Archives and Facilities

Working Group Nicolas Payette, Chris Hollander, Vera Kazakova, Sara Jonsson, Richard Legendi, Paul Wiegand, and Vittorio Scarano

MASON is a powerful environment, useful for solving complex scientific problems, but the system is hard to approach for new users. The new MASON manual mitigates that problem to a certain extent, but it is a reference work that can be daunting for a beginner. Examples are few, and there is no obvious, active community to which to reach out. The mailing list, while active and helpful, is often concerned with technical problems that can be intimidating for new users and provides limited interaction and search capabilities.

Because MASON is challenging to learn, it is less likely that people are going to use it to address their scientific problems. And those that do adopt it spend a lot of time overcoming the initial learning curve — time that they could be spending solving scientific problems.

Coming up with a precise plan regarding a collaboration space is outside the scope of this brief strawmen proposal. We have, however, identified several things that would need to be made easily sharable between MASON users. There are, of course, other existing tools that overlap with what we have in mind here (for example, StackOverflow, the OpenABM website, the Modeling Commons for NetLogo models), but there still needs to be a collaboration space that would be specific to the MASON community and bring it together: something that could serve as a bridge into the MASON world.

The main thing that people need to share is models. But these can range from whole scientific models to small and useful code snippets (the "cookbook" model was often mentioned in user group discussions). Examples that have pedagogical value (perhaps shared by MASON teachers) would be especially desirable. To be useful, this repository of community resources must be well structured. Two main forms of organization were suggested: by application domain and by level of complexity. These two are not mutually exclusive, and other dimensions could be added as well.

The social aspect is crucial for the community to maintain itself. Whatever artifacts end up being shared, people need to be able to comment on them, propose modifications (maybe fork them), share them, etc. That kind of interaction can also foster collaboration on larger scientific projects.

One issue that is related to the social aspect of the community is that of trust: we need to know if a proposed model or other artifact is safe to run or is scientifically valid. Having some expert (or team of experts) do this is unrealistic, as it is a time consuming job, but perhaps it could be done by the community, through a reputation system similar to that of StackOverflow or some other community-distributed mechanism for trust and certification.

8 Validation Via Optimization and Parameter Sweeping

Working Group Claudio Cioffi-Revilla, Keith Sullivan, Glen Ropella, Przemyslaw Szufel, Vlasios Voudouris, Sean Luke, and Daniil Kiose

Agent-based models are complex, involving large numbers of interaction rules, model parameters, and agent behaviors. This makes model validation and verification challenging, particularly when the proper settings of some of these rules or parameters are not known ahead of time. We identify four kinds of parameters/rules. First, there are those parameters/rules which are *known* to be true based on observation of real-world phenomena. Second, there are parameters/rules whose settings are based on the canonical theory or hypothesis which the model is designed to support or verify. Third, there are parameters/rules for which the model-developer does not have a good idea as to what the proper setting should be. We call these *hidden parameters*. Finally, there are parameters/rules over which the model designer wishes the model to be insensitive. We call these the *foil parameters*.

The question is how to validate such models. We believe one approach is to apply optimization techniques to attempt to optimize the hidden parameters regardless of settings of the foil parameters. The objective function for optimization is the difference in output statistics of the model when compared to expected results drawn from historical or real-world known behavior. Ideally the optimizer would find settings of the hidden parameters which optimize this objective function regardless of how the foil parameters are set.

The most obvious techniques for modeling complex models such as these are stochastic optimization or metaheuristics methods such as evolutionary algorithms. Evolutionary algorithms have the additional advantage of being highly distributable to multiple machines (important because models are costly to run and test); and so-called *coevolutionary* methods, among others, can find solutions which are insensitive to foil parameters.

An optimization package may also be useful *within* a model. Agent behaviors often involve a degree of bounded rationality and/or expectations. In other words, agent's develop rules by forming expectations and then optimize their decision space given their expectations. By developing an optimization package within MASON, applied modelers will be better positioned to develop "realistically rendered" agent-based models.

MASON also would benefit from a good parameter-sweep functionality, particularly over distributed machines. This will significantly enhance model validation, model calibration and systematic exploration of the decision space to discover unanticipated results in parts of the space which can have important implications and consequences on the underlying theory. It is also helpful to have dynamic parameter sweeping so that the parameters settings change as a function of the model performance. Therefore, the modeling of flexible conditional distributions is important in order to better capture the tails of the decision space.

For this reason MASON could benefit significantly from the development of external and internal model optimization tools, and from parameter sweep facilities, particularly ones which can be distributed or parallelized.

9 MASON and GIS

Working Group Mark Coletti, Ken Cline, John McManus, and Luís de Sousa

GeoMASON is an extension to MASON that makes it easier for practitioners to implement simulations that occur on the Earth's surface. Examples of these kinds of simulations include modeling wildfires, traffic, flooding, animal migration, refugee movement, disease propagation, and climate change effects. Writing MASON code to support this functionality without GeoMASON would be difficult, time consuming, and error prone; that is, practitioners would have to implement their own means of reading, writing, and manipulating geospatial data to support these kinds of simulations, which is the kind of functionality that GeoMASON provides.

Currently GeoMASON provides a number of primitives for querying and manipulating geospatial data. For example, a user could determine what political entity contains a given agent, find the nearest road, count the number of diseased agents in a country, and move agents along a path, among other things.

We feel that GeoMASON can be greatly improved. For one, the available geospatial operators are fairly primitive such that the user is left to assemble these operators to support higher level functions. For example, GeoMASON would greatly benefit from having better support for networks, such as finding the closest edge, shortest path, and moving along routes. Another area of improvement would be in performance, particularly with regards to geospatial operations. We would need to do a detailed assay of hotspots via profilers and address those as they are found. Another means of performance improvement would be better support for D-MASON, which would allow spreading geospatial-related computational burden across separate CPUs.