The Multi-Agent Simulation Suite

Marton Ivanyi, Rajmund Bocsi, Laszlo Gulyas, Vilmos Kozma and Richard Legendi

AITIA International, Inc.
Czetz Janos u. 48-50
H-1039 Budapest, Hungary

Abstract

Agent-based modeling is a branch of computer simulation, especially suited for studying complex social systems. It models the individual, together with its imperfections (e.g., limited cognitive or computational abilities), its idiosyncrasies and personal interactions. The approach builds the model from 'the bottom-up', focusing mostly on micro rules and seeking to understand the emergence of macro behavior. Participatory simulation - a branch of agent-based simulation - is a methodology building on the synergy of human actors and artificial agents, excelling in the training and decision-making support areas. In participatory simulations some agents are controlled by users, while others are software governed. The Multi-Agent Simulation Suite is a software package intended to enable modelers to utilize the tools of agent-based simulation in various fields, without having to develop heavy programming skills.

Introduction

The Multi-Agent Simulation Suite consists of three applications built around a simulation core (see Figure 1). The simulation suite has its own core called the Multi-Agent Core (MAC), but it is also able to run on the popular Repast core (North, Collier and Voss 2006). Being multi-core enables modelers to verify that results are core-independent, thus we plan to further develop this option. The Functional Agent-Based Language for Simulation (FABLES) is a programming language and its integrated programming environment specially designed for creating agent-based simulations. The Model Exploration Module (MEME) is a tool that enables orchestrating experiments, managing results and has support for their analysis. MASS also has an element that does not translate to a stand-alone application. It is called the Visualization Package and consists of the various implementations of charts and visualizations used in the other programs in the simulation suite. The PET (Participatory Extension) is an optional web-based environment for multi-agent and participatory simulations that is not going to be discussed in detail in this paper due to space limitations.

- MASS offers an integrated modeling environment with a simple programming language that is similar to the mathematical formalism used in publications.
- The rich charting package that is embedded in all the applications is operated through user friendly interfaces thus simplifying the creation of visualizations.
- The web-based environment and participatory framework provides means for multiple users running, observing or participating in the simulations enabling interactions between human and non-human agents.
- With the help of MEME modelers can design experiments, manage and analyze the data produced and carry out sensitivity analysis while working with various simulation cores.

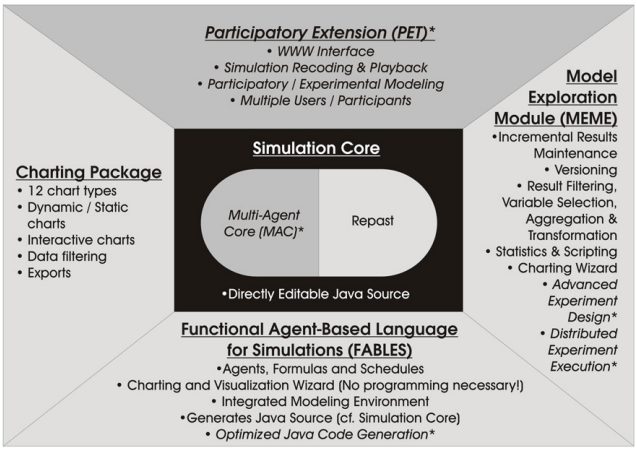

Figure 1: The Multi-Agent Simulation Suite consisting of four applications built around the simulations core.

Motivations

It is understood that the experts of the various fields affiliated with agent-based simulation would rather concentrate on the behavior emerging from their models instead of spending the bulk of their time and effort with the related programming (Gulyas and Bartha 2003). The CRC of Eotvos University and AITIA International Inc. cooperate on developing their platform independent simulation suite (MASS) to

meet this need on a high level. MASS offers an integrated modeling environment with a simple programming language that is similar to the mathematical formalism used in publications describing simulations. Although probably this is the most demanding part of the toolset as it requires some coding, the language is both very simple, focused on agent-based modeling and assisted by wizards, tutorials and help functions. In most cases creating visualizations for complex models with large datasets can be the most complicate part of model creation. The rich visualization package that is embedded in all the applications is operated through wizard-like intelligent interfaces thus giving a tool that can visualize simulation results in a few easy to do steps. The web-based environment and participatory framework provides means for multiple users running, observing or participating in the simulations enabling interactions between human and non-human agents in a way users are accustomed to, through web browsers. With the help of MEME modelers can design experiments, manage and analyze the data produced and carry out sensitivity analysis while working with various simulation cores.

History

In 2004 a participatory extension was developed for Repast at AITIA International, Inc. The application was used in the vBroker (Tatai, Gulyas, Laufer and Ivanyi 2005) stock market game - a project commissioned by the Hungarian Financial Supervisory Authority - where artificial agents and human users were trading along on a virtual stock market with close to real life properties. The knowledge and experience accumulated led to the development of the first beta version of the Participatory Extension (PET) and its simulation core, the Multi-Agent Core (MAC).

A different thread of development started at ELTE in 2004 where Sandor Bartha was working on the outlines of an easy to use language for agent-based simulation (Bartha 2005). By mid 2005 the Functional Agent-Based Language for Simulation was integrated into the MASS project and the development of an Integrated Modeling Environment (IME) was launched. The early versions of PET and FABLES IME were presented to the public at the Gartner ITEXPO 2005 in Orlando, Florida and at the Agent 2005 conference (Gulyas and Bartha 2005).

Upon developing these two applications the need for a universally applicable and easy to use visualization tool has formed. The Charting Package has been developed since mid 2005. Initially it was used in a joint data-mining project by AITIA International, T-Com Hungary, MTA Computer and Automation Research Institute (MTA SZTAKI) and Budapest University of Technology and Economics (BME). The Charting Package is now integrated in FABLES and MEME and is also being integrated into PET.

In 2006 we have identified niche in the field of ABM applications; there was hardly any software on the market being able to run a large number of simulations, collect, organize and visualize their data that could be operated by modelers with limited programming skills. Development of the Model Exploration Module (MEME) begun in mid 2006. First the toolset for collecting, organizing and visualizing simulation data was completed, then means for orchestrating experiments and running batch simulations were implemented. Currently the MEME development team is testing the extension enabling users to run simulations on variuos kinds of grid solutions.

FABLES

The Functional Agent-Based language for Simulations (FABLES) is a programming language and its integrated environment developed by AITIA, Inc. specially designed for creating agent-based simulations. It requires minimal programming skills, as its formalism is similar to the mathematical formalism used in publications in the subject.

FABLES has a whole range of functions intended to make the use of the language easier like the language assistant, syntax highlighting, bracket matching and the context sensitive help. An observation wizard is also provided with the language, which gives great ease to creating visualizations from simulations. With the above-described IME this constitutes what is called Integrated Modeling Environment.

Language

The language combines the strengths of functional programming with the object-oriented paradigm, providing unique means to implement agent-based simulations. FABLES is a dynamically typed, lazy, non-pure functional language. It contains higher-order functions, set and sequence expressions, and special language constructs for scheduling events.

FABLES' vision is an abstract formalism to describe agent-based models. Models defined in this language could be compiled both to Repast and the MASS' own simulation core. Such a description is ideal to publish concise definitions of agent-based models. Moreover, independently developed compilers to different modeling platforms in place, the formalism could also help making the replication/docking of computational models a routine task. When creating FABLES we have tried to eliminate as much unnecessary user side tasks and efforts as possible (Gulyas and Bartha 2005). In order to create a clear and simple language syntax, redundant type definitions are sorted out.

Models consist of standard text files with .fab extensions. Models begin with the keywords `model <ModelName>`, starting with { and ending with }. Definitions for functions, procedures, schedules, and classes go between the { and } characters. Besides the numerous built-in functions in the language defining functions by the user is simple. Also, every function definition may have local function definitions in any depth.

The relations in a model, or rather a *world* in this case, are defined by expressions. Expressions consist of sequences of numbers, constants and/or operations, resulting in arithmetic, logical and conditional expressions, sets and sequences and strings. FABLES is a non-pure functional language, it has variables or references (a general functional language does not operate with them). Everything in FABLES that is not a variable always denotes a non-changeable value. The variables are used to store the state of the model and the agents. Both the model and the individual agents can have variables.

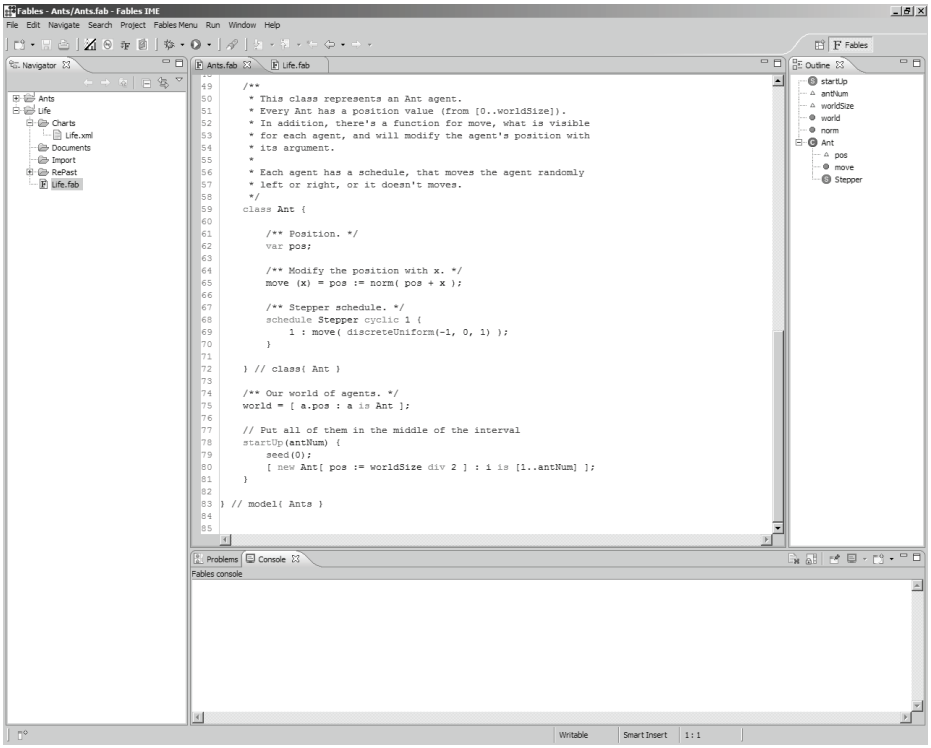

Figure 2: Screenshot of FABLES in operation. Navigation panel on the left, the actual model code in the middle, Outline in the right, Help and support on the far right panel.

Agents in FABLES are defined by *Agent Classes*. The classes are sets of agents (objects) of the same type. In FABLES a class definition is also a template for its instance objects. The syntax of classes is similar to models, beginning with the keywords `class <ClassName>` with member definitions declared in the block. A model or class may contain a `startUp` section with expressions to be evaluated upon starting the model or at the creation of an agent (object).

The execution of models involves scheduling events. Schedules can belong to the model itself or to individual agents (objects). Each event has a time variable, a non-negative integer the order of event activations is determined by. Events with lower time values are activated first. When two or more events should be executed at the same time, the order is decided by uniform random distribution, generated by the standard (deterministic) pseudo-random generator.

For a working example see the Appendix where the 'Ants' model is presented. It is a short, well commented model that provides a quick understanding of Fables.

Integrated Modeling Environment

The Integrated Modeling Environment (IME) makes model development in FABLES more effective by providing a modeler-friendly interactive editor based on the Eclipse platform. The user guide and tutorials are integrated in the environment as well as demo applications (see Figure 3) and the Fables specific help.

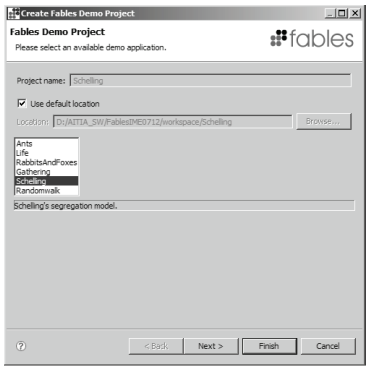

Figure 3: Browsing the example projects in FABLES.

The IME has numerous language support features for FABLES. When a function is typed in that is part of the standard library, intelisense identifies it on the fly, offers and checks for the proper syntax and parameters. It does syntax-highlighting, brackets matching and completes special characters or functions automatically. The outline panel - on the right side of the screen by default - is very useful as well, as it collects functions, constants and other special elements providing easy means of navigation in the source code. FABLES and the environment is also capable of automatic document generation in HTML, LaTex source, PDF, DVI and RTF formats.

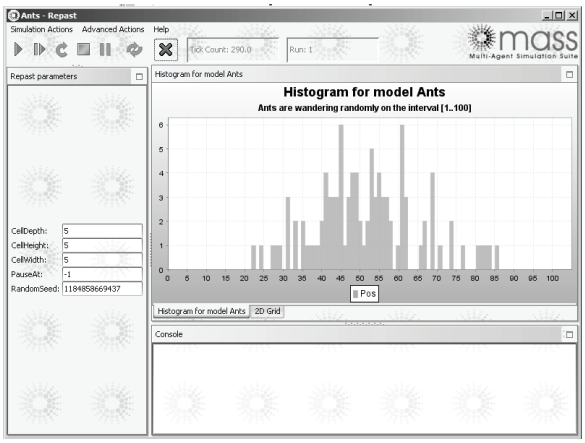

Figure 4: Running the model Ants with the MASS GUI interface from FABLES.

FABLES models can be run with or without the graphical user interface (GUI) through the environment by pressing a single button. See Figure 4 for the Ants model mentioned before running in the MASS GUI interface. Special speed-buttons can be set up for given projects with specific settings for GUI-generation, optimizing and debugging. The buttons, panels and the overall appearance of the environment can be customized according to the modelers needs. The IME has full Java support, and Repast models can be created in it as well, supported by almost the same features as coding in FABLES.

The *FABLES Charting Wizard* enables the modeler to create visualizations without any coding. The wizard based on the *MASS Visualization Package* collects and processes all the necessary and usable data from the model and leads the user through the creation of two- and three-dimensional displays interactively.

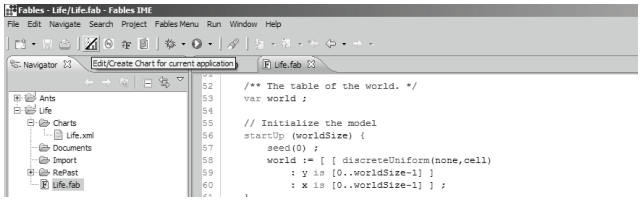

Figure 5: Creating visualizations in FABLES is just a matter of a couple of clicks.

Visualization Package

The MASS VISU is a supporting tool designed to be a general, user friendly visualization application. It can visualize data from arbitrary sources - such as MEME or FABLES - through its widely applicable components. Through its visualization wizard designing time series, function graphs, bar charts, histograms, area-color maps (also known as ”map of the market” or ”market carpet”), 2D grids, sequence and network diagrams, etc. is both easy and convenient. See Figure 6 a)-c) for working examples of the wizard.

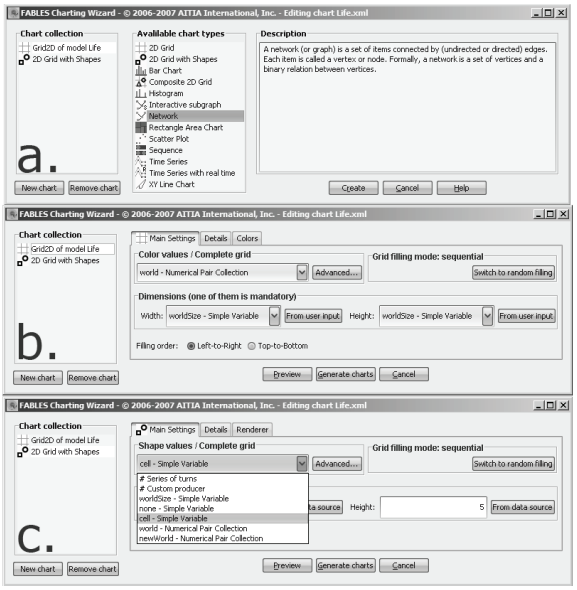

Figure 6: **a)** Browsing the available visualization types. **b)** Creating 2D grid through the charting wizard. **c)** Setting up the parameters of the 2D grid.

It is capable of drawing both static and dynamic charts, data filtering and has export options. The goal of the framework is to be able to visualize data from various sources and to provide widely utilizable, interactive components for modelers who wish to spend their time on developing their models rather than designing and coding visualizations. The package has been developed to be able to support network (HTTP for example) applications.

Components

The Visualization Package can be broken down to the following components:

- *Data model:* Data-interfaces and -objects representing the data to be visualized.
- *Data source:* This layer abstracts data sources, enabling the visualization of data from various sources. This way the source of the data to be visualized is opaque for the visualization component.
- *Visualization component:* The graphic components responsible for data visualization.

An important attribute of this visualization package is the separation of the data model and the visualization. During development our goal was to have tool for visualizing data from any source. To achieve this we introduced a data source layer responsible for generating the datasets ready to be visualized. The visualization component provides a constantly expanding list of interactive graph and visualization types. In fact one of the main features of the visualization component is its interactivity (eg. assigning tooltips, selection, resizing, etc.) available to the modeler.

Currently the following list of graphs and charts are available in the Visualization Package:

- *Time series:* Graphic component capable of visualizing one or more series on a chart. We interpret series as functions on a set of natural numbers.
- *Function graph:* The function graph component can be used to visualize a set of any given functions.
- *Bar chart:* Visualizing a set of numeric values in form of bars.
- *Histogram:* Visualization of numeric values' frequency distribution.
- *Area-based visualization:* Area-based visualization draws value pairs on a rectangle area. The available space is divided into pieces proportional to one of the values. The second value determines the coloring of the sub-areas.
- *2D grid:* Visualizing numeric values in a rectangular grid with coloring and in various shapes. This is probably the most common type of 2D visualization in agent/based simulations. See Figure 6. b) and c).
- *Sequence:* Visualizing ordered series of sets.
- *Network:* The network component enables the visualization of arbitrary graphs.
- *Sub-graph (of network):* Visualizing sub-graphs, a selected vertex and its neighbors in a given depth of a network. With the help of this component, huge, complex graphs can be walked through in a way where only one, relatively small sub-graph is visualized at once.

MEME

MEME stands for the Model Exploration Module of MASS. Models are generally used for studying, or exploring the behavior of certain real phenomena in different circumstances. The systematic study is done by observing various factors of the model during (or at the end of) unattended, usually long and/or large number of simulations or batch running of experiments in other words. These observations produce a huge amount of raw data that needs to be managed and analyzed.

MEME is a tool for dealing with the above described experiments and the data produced. MEME is not only able to run MASS/Repast simulations directly in batches and collect their results internally but it allows the user to design the experiments through wizards that intelligently sweep and organize parameters. See the wizard page on Figure 7 for importing a Repast model and all the related jars and classes. The modeler can store, filter and organize the raw data produced in databases. Processed view tables, to be visualized on graphs and charts or exported into other applications, are also created through automated wizards. MEME's export options already enable analyzing results in existing statistical software like R and Matlab but the program is developed to be able to interoperate with such applications.

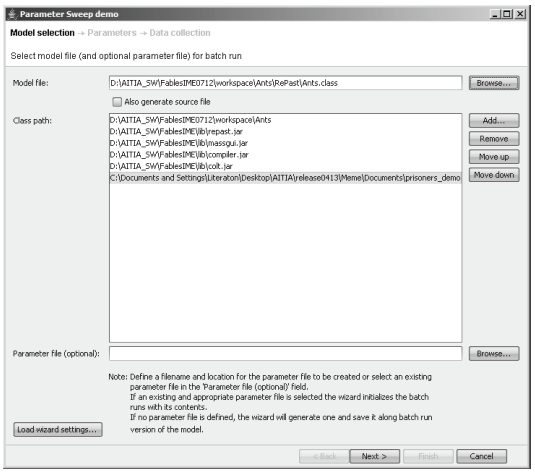

Figure 7: Importing a Repast model in the parameter sweep wizard.

Experiment Design and Execution

Agent based simulation usually involves studying, or we can say, exploring the behavior of model in different circumstances. This is done by observing various factors of the model during (or at the end of) unattended simulations, usually long and/or large number of simulations. These simulations are executed on parameter spaces that are either pre-defined or intelligently identified during the runs. These observations produce huge amounts of raw data to be dealt with.

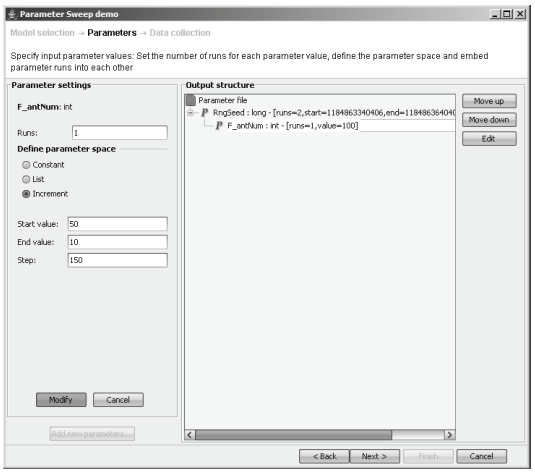

Figure 8: Setting up the parameter space in the parameter sweep wizard.

MEME offers solutions for designing experiments from Repast or FABLES models. Through its parameter sweep tool (see Figure 8) it identifies input parameters and offers them in the process of setting up the parameter space the simulations are to be run on. It also allows the user to define the input parameters in models where they have been not declared originally or to add to the already existing ones.

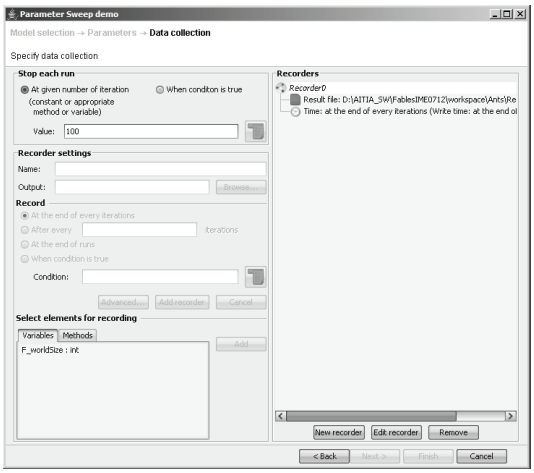

Figure 9: Configuring data collectors in the parameter sweep wizard.

After the parameter space is set up through these complex, embedded sets of parameter combinations, the user defines the data recording attributes (see Figure 9) such as saving periods, tick counts, elements to be recorded and can also declare new output variables for recording through scripting and calculations. These settings can be saved into separate files and loaded to be run or edited later. When executing these settings a new model is generated as a new java class file containing all the necessary information for the batch runs. Note that source generation is also possible. MEME launches the batch runs in a separate thread from, or rather in the background of the main application so observing the incoming results and overseeing the general state of the experiment is possible while executing the high number of simulations.

Storing, Organizing and Visualizing Simulation Data

MEME organizes simulation results in a 3-level hierarchy through its built-in database engine or the user's external database engine of choice. Every simulation result belongs to a particular version of a particular model. Here the "version" and "model" are arbitrary names entered by the user. These are the first two levels of the hierarchy. The third level organizes the batches, because usually a batch of runs is performed at once, generating a batch of simulation results. Therefore every simulation result belongs to a batch, too. Batches are identified by numbers that are incremented automatically by MEME.

Results are composed of values recorded during simulations or at the end of simulations. These values are usually associated with names that indicate parameters. MEME identifies and distinguishes between input and output parameters, ones that denote those values that describe/represent the circumstances in which the model is started, therefore never change during a run, and the factors that may change during the simulation and are observed for recording.

The raw simulation data, imported into MEME from

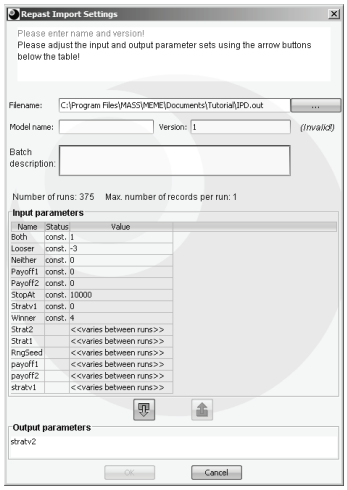

Figure 10: Importing a Repast result file into MEME.

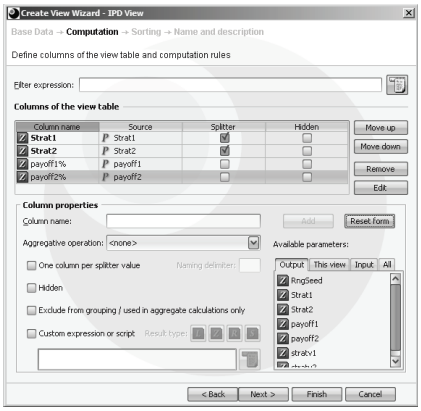

Figure 11: A processing raw data to computed view table through a wizard.

Repast result or CSV files or generated with the program through the batch running of Repast or FABLES simulations, is assembled into computed tables that provide input for visualizations and for sophisticated analysis (see Figure 12). View tables are created from batches of simulation results that can belong to any model and version. MEME supports custom computations, filtering, splitting, aggregating and reorganizing on the sets of value columns in the table (see Figure 11). View tables - as well as visualization configurations - are stored in independent files enabling editing, changing or if needed further developing them at any time while leaving the original data intact. The saving of filtering information in files also allows for the batch processing of view table generation.

Visualizations can be created from view tables described above, with the Charting Wizard which is built on the Visualization Package described earlier. Additional features provide further help and assistance for modelers who wish to visualize their results quickly in easy steps, but in a professional manner. Upon selecting a chart type to be created

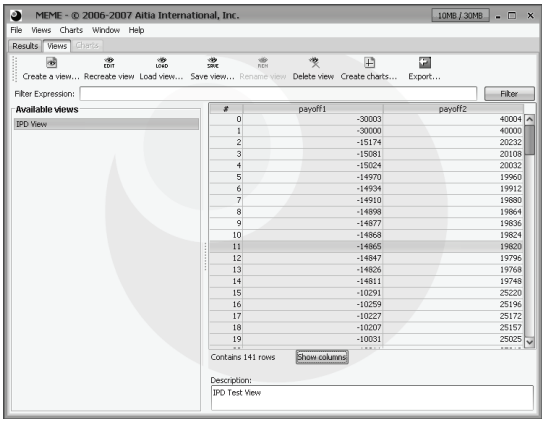

Figure 12: The resulting view table in MEME. Visualizations can be created from view tables or thez can be exported for further analysis.

MEME applies automatic conversions depending on the actual type of values in the data variables in question and the type the given chart requires. It also identifies and offers variables, sets of columns that are adequate for the selected visualization.

Further Developments

The MEME development team is currently working on implementing batch execution on clusters of computers and different types of grids. This development will further simplify and speed up modeling work by reducing execution times significantly without requiring modelers to develop specific grid knowledge or skills. An other ongoing project is introducing advanced design of experiment capabilities. While currently MEME batch execution is using brute force, executing all parameter combinations, support for design-of-experiments features are also planned.

MEME is not intended to be a professional statistical software. Rather, while it is currently capable to provide input for such applications in the form of CSV files, further plans of development include implementing interoperation with advanced statistical software. As the program is being developed modularly it is also possible to enable MEME to deal with additional languages and modeling environments based on Java (eg. Mason).

Conclusions and Further Work

The above described Multi-Agent Simulation Suite and its components - FABLES, MEME, PET and the Visualization Package - are under constant development but they have reached a state where we are already using them to help us in different simulation projects. We are in the process of organizing and evaluating user tests hence we hope to have the necessary feed-back to further improve their capabilities.

Concerning FABLES optimizing the compiler is one of the important tasks awaiting our development team. In the mean time we are designing template-models, general toolsets for different areas of agent-based simulations and evaluating the possibilities of introducing additional simulation cores in the application. As for the Visualization Package, the available assortment of charts and visualizations is constantly expanding trying to meet all the occurring user needs and requests. As already mentioned above grid solutions developed for MEME are being tested as of the moment, while interoperation with advanced statistical software is in preparation.

Acknowledgments

The partial support of the GVOP-3.2.2-2004.07-005/3.0 (ELTE Informatics Cooperative Research and Education Center) grant of the Hungarian Government and EC grants QosCosGrid IST FP6 033883 and EMIL IST FP6 033841 is gratefully acknowledged.

The work of Sandor Bartha developing the first versions of the FABLES language is also gratefully acknowledged.

Appendix: Ants

```
//Ants wandering randomly in 2D world.

model Ants {
/** The number of ants. */
antNum    = 100; // PARAMETER

/** The size of the interval */
worldSize = 100;

/* Please note if you change the size,
you should re-generate the chart with
the corresponding value */

/** Normalizing the world to make it
permeable. */
norm (x) = x mod worldSize;

/**
 * This class represents an Ant agent.
 *Every Ant has a position value
 *(from [0..worldSize]). In addition
 * there is a function for movement,
 * that is possible for each agent and
 * that modifies the agent's position
 * with its argument. Each agent has a
 * schedule moving the agent randomly
 * left, right or leaving it alone for
 * the turn.
 */

class Ant {

/** Position. */
var pos;

/** Modify the position with x. */
move (x) = pos := norm( pos + x );

/** Stepper schedule. */
schedule Stepper cyclic 1 {
```

```
1 : move( discreteUniform(-1, 0, 1) );
}

} // class{ Ant }

/** Our world of agents. */
world = [ a.pos : a is Ant ];

/** Put all of them in the middle of
the interval. */
startUp(antNum) {
  seed(0);
[ new Ant[ pos:=worldSize/2 ] : i is
[1..antNum] ];
}

} // model{ Ants }
```

References

Samuelson, D. and Macal, C. 2006. Agent-based Simulation Comes of Age, *OR/MS Today* 33(4): 34-38. Marietta, GA: Lionheart Publishing.

Axelrod, R. and Tesfatsion, L. 2006. A Guide for Newcomers to Agent-Based Modeling in the Social Sciences in Tesftasion, L. and Judd, K.L. (Eds.) *Handbook of Computational Economics Volume 2: Agent-Based Computational Economics, Handbooks in Economics Series*: Appendix A. Elsevier/North-Holland, Amsterdam, the Netherlands.

Gulyas, L. and Bartha S. 2003. How Much Is Too Much? – What Programming Skills Are Really Needed to do ABM? In Proceedings of Seventh Annual Swarm Users/Researchers Conference. Notre Dame, Indiana: Swarmfest 2003.

North, M.J., Collier, N.T. and Vos J.R. 2006. Experiences Creating Three Implementations of the Repast Agent Modeling Toolkit. *ACM Transactions on Modeling and Computer Simulation* 16(1): 1-25. New York, New York: ACM.

Gulyas, L. 2002. On the Transition to Agent-Based Modeling: Implementation Strategies From Variables To Agents. *Social Science Computer Review* 20(4): 389-399.

Tatai, G., Gulyas, L., Laufer, L. and Ivanyi, M. 2005. vBroker: Artificial Agents Helping to Stock Up on Knowledge. In *Pechoucek, M., Petta, P. and Varga, L.Zs. (eds.). Multi-Agent Systems and Applications IV.* Proceedings of the 4th International Central and Eastern European Conference on Multi-Agent Systems (CEEMAS 2005): 336-345. Springer-Verlag

Gulyas, L. and Bartha, S. 2005. FABLES: A Functional Agent-Based Language for Simulations. In *Proceedings of The Agent 2005 Conference on: Generative Social Processes, Models, and Mechanisms*. Chicago, IL: Argonne National Lab.

Gulyas, L., Ivanyi, M., Szabo, V., Adamcsek, B., Korompai, A. and Goldpergel, A. 2005. vBroker: A Participatory Stock Market Experiment with Heterogenous Agents. In *Proceedings of the 10th Annual Workshop on Economic Heterogeneous Interacting Agents*. Essex, UK: Univ. of Essex.

Bartha, S. 2005. Formalizing and Programming Multi-Agent Simulations. Thesis *(in Hungarian)*, Faculty of Informatics, Eotvos Lorand Univ., Budapest, Hungary.