AGSYSLIB – A SOFTWARE TOOL FOR AGENT-BASED PROBLEM SOLVING

Şerban IORDACHE[1], Florica MOLDOVEANU[2]

Algoritmii bazaţi pe agenţi au capacitatea de a se adapta la medii dinamice şi complexe şi sunt utilizaţi adesea pentru a rezolva probleme practice de mare dificultate. Există însă o serie de dificultăţi legate de proiectarea, implementarea şi ajustarea parametrilor acestor algoritmi. În prezentul articol ne propunem să identificăm aceste dificultăţi şi să arătăm cum pot fi ele depăşite folosind AgSysLib, o bibliotecă software pe care am creat-o în acest scop.

Agent-based algorithms are able to adapt to complex, dynamic environments and are frequently used to tackle difficult real-world problems. There are, however, a number of difficulties encountered in designing, implementing and tuning agent-based algorithms. In this paper, we identify these issues and we present AgSysLib, a software tool that we have developed in order to overcome them.

Keywords: agent-based problem solving, heuristics, stochastic algorithms, parameter tuning

1. Introduction

Many complex phenomena that arise in physical and biological systems can be naturally described using agent-based models [1][2][3][4]. These models are able to capture the complex behavior that emerges from the interactions between agents governed by simple rules. It is tempting to use agent-based approaches not only to describe existing phenomena, but also to solve complex problems that cannot be tackled with conventional techniques. Consequently, in recent years, there has been a growing interest in designing algorithms that take advantage of the emergent behavior exhibited by systems composed by interacting agents [5].

While conceiving a new agent-based algorithm, one can benefit from the various software libraries and frameworks available nowadays for agent-based modeling and simulation (ABMS) [6]. However, ABMS software is not able to assist in all aspects related to the design, implementation and tuning of agent-based algorithms. The aim of this paper is to identify the difficulties encountered during these activities and to discuss how a software tool can help in overcoming

[1] Eng., SCOOP Software GmbH, Cologne, Germany, e-mail: siordache@acm.org
[2] Prof., Faculty of Automatic Control and Computers, University POLITEHNICA of Bucharest, Romania, e-mail: fm@cs.pub.ro

them. We illustrate our ideas by presenting AgSysLib, a Java tool that we have developed in order to facilitate the tasks associated with agent-based problem solving.

The main difficulty in designing agent-based algorithms is to find the set of rules that produce the desired emergent behavior. At present, there is no generally accepted methodology for designing agent-based algorithms. Some of these algorithms are nature inspired. They adopt mechanisms found in biological systems such as colonies of ants [7] and bees [8], flocks of birds [9] or immune systems [10]. When no inspiration is found in nature, the task of finding an appropriate set of rules for the agents becomes more difficult, because the emergent behavior is often surprising and counterintuitive. Therefore, the design of an agent-based algorithm is frequently a trial-and-error process that could benefit from the help provided by a software tool.

While ABMS offers mainly qualitative insights, agent-based problem solving is concerned with producing high performance results in terms of both solution quality and running time. Turning a proof-of-concept simulation into a state-of-the-art implementation for solving a real-world problem is a very tedious task, for which ABMS software does not typically provide support. Since currently no tools are offering assistance with this task, we have developed AgSysLib in order to help algorithm designers and engineers in implementing agent-based solutions for complex problems.

2. The AgSysLib framework and library

AgSysLib is both a framework and a library. It is implemented in Java and it is available as an open source project at http://agsyslib.sourceforge.net/. AgSysLib offers an API (Application Programming Interface) that should be implemented by any agent-based algorithm and it provides a set of utility classes that help performing various tasks associated with agent-based problem solving.

AgSysLib features a component-based architecture, which permits to build algorithms in a modular way and facilitates the experimentation and analysis of different variants of an algorithm. A new variant of an algorithm can be obtained by simply replacing a particular component of a base implementation with another component. AgSysLib also allows executing batches of runs, in order to apply repeatedly an algorithm to the same problem, to different problems, or using different parameter values.

Configuration

Agent-based problem solving is typically a trial-and-error process requiring experimentation at various levels. One type of experiments involves assessing different variants of an algorithm. Frequently, switching to a new

algorithm variant is achieved by commenting out portions of the original code and replacing them with new code, or by using flags in various parts of the program in order to activate the portions of code corresponding to the given algorithm variant. Such a practice clutters the source code, making it hard to read and to maintain. A component-based architecture allows a clear separation of the portions of code specific to each algorithm variant, but it usually still requires some changes in the original code, in order to specify which component should be used. AgSysLib allows specifying all components of an agent-based system in a configuration file. This way, no code changes or additional flags are needed in order to switch between different algorithm variants.

Another type of experiments involves comparing the results obtained by a given algorithm with different sets of parameters or even with different formulas. If parameter values or algorithm formulas are hard-coded into the program, this leads again to code cluttering. Therefore, AgSysLib provides a large set of utility functions for reading various types of values and lists of values, as well as mathematical formulas from a configuration file. Moreover, AgSysLib allows providing lists of values for parameters and executing batches of runs for each combination of these parameter values. Finally, it is possible to specify in the configuration file that the value of a parameter should be computed based on a given formula using the values of other parameters.

AgSysLib does not use an interpreter to evaluate formulas, because this would have a negative impact on the performance of an algorithm. Instead, it generates on-the-fly a Java class for each formula and creates an instance of it. The dynamically generated class contains a method that accepts as arguments the variables present in the given formula and returns the value corresponding to its evaluation. The on-the-fly class generation takes place only once, at program start. This way, the evaluation of a formula is performed as fast as if it would have been hard-coded in the algorithm code. In a series of experiments aimed to assess the performance of the implementation of our formula evaluator, we have determined that the time needed to evaluate a formula using a Groovy interpreter is in average about 15 times longer than that needed by our evaluator.

The AgSysLib API

The framework controls the execution of all operations required to evolve an agent-based system implemented in AgSysLib. This way, the user is no longer concerned with general aspects regarding the working of an application. Instead, he can concentrate on the details of his particular problem. To this end, the user has to implement some of the interfaces specified by the AgSysLib API. For many of these interfaces, AgSysLib offers default implementations or abstract classes that can be easily instantiated, extended and customized.

The particular API implementations used by a certain application can be specified in a configuration file. Instances of the classes declared in this file are created through Java reflection. Since this operation is done only once, at program start, it has no impact on the performance of the application.

There are two high-level interfaces that must be implemented by any AgSysLib application: Initializer and AgentSystemEvolution. The Initializer interface has a single, parameterless method, called getNextAgentSystem, which is called before starting a new run in a batch, in order to retrieve the agent-based system for the new run. The AgentSystemEvolution interface declares methods that should implement the actions performed at the beginning and end of each run in a batch, and at the beginning, at the end and during each step in a run. It also declares a method that should implement the termination condition of a run. For both interfaces, AgSysLib offers abstract methods implementing the base functionality usually needed by any concrete implementation.

During experimentation with an agent-based algorithm, it is frequently necessary to inspect the evolution of various quantities handled by the application, or aggregate values of them, in order to gain insight about the behavior emerging in the system. Similar information is required during tuning and debugging activities. The statements needed to output this information usually clutter the source code. Moreover, they may affect the performance of the algorithm, although they are usually no longer needed in the final application. The AgSysLib API introduces evolution listeners in order to allow keeping these statements separated from the main source code, while also permitting the quick activation or deactivation of these portions of code.

The evolution listeners active during the execution of an application can be specified in a configuration file. This way, there is no need to make changes in the application code in order to add or remove a listener.

Evolution listeners are typically used to watch and monitor the evolution of an agent-based system. They provide methods that can be triggered by the following events:

- the start of the processing for a batch of runs
- the end of the processing for a batch of runs
- the start of the processing for a run in a batch
- the end of the processing for a run in a batch
- the start of an evolution step in a run
- the end of an evolution step in a run
- the end of the operations performed by an agent during a step.

The remote control console

Agent-based applications are usually very adaptive, thus being the best choice for solving real-world problems in complex, dynamic environments.

However, detecting flaws in such applications is more difficult, because in many cases, even a buggy implementation is able to solve the problem, although not as efficient as possible. In addition, when designing a new agent-based heuristic, one does not know in advance how efficient an implementation could be, therefore bugs that only affect the algorithm performance may remain unnoticed, since the developer has only limited knowledge about what to expect from the algorithm. Because many agent-based algorithms are stochastic, reproducing an unusual behavior may also prove difficult.

Investigating problems related to agent-based applications often requires a detailed analysis of the dynamic of the internal program state. AgSysLib offers a GUI component that allows connecting via RMI (Remote Method Invocation) to a running application and interrogating, watching or changing its internal state. This GUI component, shown in Fig. 1, is called the remote control console. Since it can establish a connection not only at program start-up, but at any moment, the remote control console can be also used to investigate unexpected behavior appearing in a program not actually under debugging.

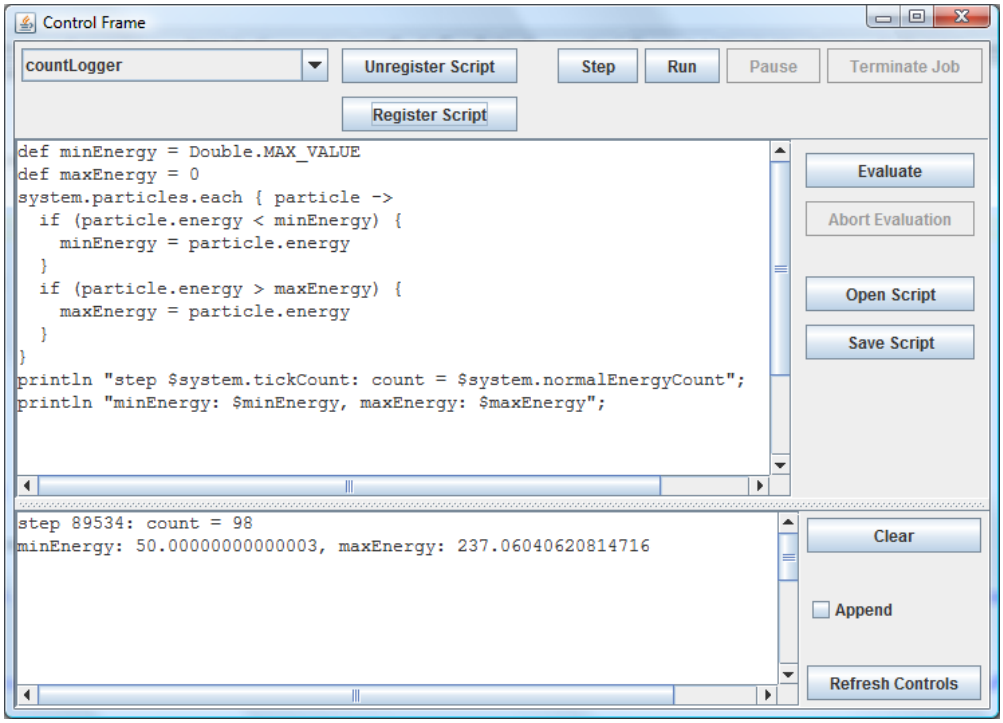

Fig. 1. The remote control console

Access to the internal state of a program is provided by means of scripts written in the Groovy programming language. Therefore, it is possible to make

complex queries on the internal program state or to make multiple state changes, such as altering in some way the state of each agent, by using only a few lines of code. The scripts can be also registered, in order to be executed at the end of each step. In addition, the results of internal state queries can be used to set complex conditional breakpoints.

After introducing the script code in the upper text area and pressing the "Evaluate" button, the results are displayed in the lower text area of the console. Scripts can be also saved to and loaded from a file. The script can be changed in order to log the output into a file and register it for execution. By pressing the "Run" button, the information queried by the string is written to the log file at the end of each step.

Tuning

As mentioned before, AgSysLib allows specifying lists of values in the configuration files, in order to run an algorithm with all possible combinations of these parameter values. Comparing the aggregate results obtained for each batch of runs, it is possible to select the best performing combination of parameter values. This basic form of parameter tuning can be used for simple problems or in the first stages of experimentation with a new algorithm. However, in order to obtain high-quality solutions, a more elaborated tuning procedure should be applied. One such tuning procedure offered by AgSysLib is based on genetic algorithms and it uses the JGAP library (http://jgap.sourceforge.net/). Additionally, AgSysLib enables an easy integration with specialized software packages for automatic parameter configuration such as ParamILS [11], F-Race [12] or SPOT [13].

Integration with ABMS software packages

Currently available ABMS software packages offer many features that are also useful for agent-based problem solving. AgSysLib does not try to provide yet another implementation of these features. Instead, it is concerned with those aspects of designing, implementing and tuning of agent-based algorithms that are not covered by ABMS software. However, in order to give users the possibility to still benefit from features available in ABMS software, AgSysLib can act as a wrapper for such libraries. This way, AgSysLib can transform an ABMS package into a tool able to assist in agent-based problem solving.

A plethora of ABMS packages has been developed in the last years, differing in their purposes and capabilities. In a recent survey, Allan [14] discusses 31 of the most commonly used toolkits for agent-based modeling and simulation, while in another survey, Nikolai and Madey [15] consider 53 such toolkits. AgSysLib is able to wrap around many of these packages, but in its default configuration, it integrates the MASON library. MASON (Multi-Agent

Simulator of Neighborhoods) [16] is a discrete-event multi-agent simulation environment implemented in Java, allowing models with a large number of agents to be executed fast a large number of times. Some of its main features include:

- checkpointing – simulations can be serialized to disk. A serialized simulation can be later recovered with or without visualization, and it can be migrated on a different platform.
- model / view decoupling – models are completely independent of their visualizations. Different types of visualizations can be defined for the same model and they can be dynamically added and removed.
- media – various 2D and 3D visualizations, charts and graphs are available, with the possibility to save them as snapshots (in PNG format) or as Quicktime movies.
- duplicability – MASON simulations are able to produce identical results across different platforms.
- high quality random number generator – many agent-based algorithm are stochastic and they need a robust random number generator in order to produce valid results. MASON provides an efficient implementation of the Mersenne Twister random number generator.

3. Conclusions

In this paper, we have identified a number of issues involved in the process of solving a real-world problem using agent-based techniques and we have introduced AgSysLib, a software tool developed by us in order to overcome these issues. AgSysLib is both a framework and a library and it facilitates many tasks related with agent-based problem solving. It can act as a wrapper around existing ABMS software packages, thus seamless integrating their capabilities.

AgSysLib has been already used in the development of new agent-based heuristics. Examples include open source packages like SwarmTSP, which implements consulting-guided search algorithms for the traveling salesman problem [17], or SwarmQAP, which implements consulting-guided search algorithms for the quadratic assignment problem [18]. The state-of-the-art performance obtained by these algorithms shows that AgSysLib is a valuable tool for all people involved in agent-based problem solving.

REFERENCES

[1] *A Troisi, V Wong, and M A Ratner*, "An agent-based approach for modeling molecular self-organization.," Proc. National Academy of Sciences, **vol. 102**, no. 2, pp. 255-260, 2005.

[2] *Leigh Tesfatsion*, "Agent-Based Computational Economics: Growing Economies From the Bottom Up," Artificial Life, **vol. 8**, no. 1, pp. 55-82, 2002.

[3] J*oshua M Epstein*, "Agent-based computational models and generative social science," Complexity, **vol. 4**, no. 5, pp. 41-60, May 1999.

[4] *Lars-Erik Cederman*, "Endogenizing geopolitical boundaries with agent-based modeling," Proceedings of The National Academy of Sciences, **vol. 99**, no. suppl. 3, pp. 7296-7303, 2002.

[5] *Susan Stepney, Fiona A Polack, and Heather R Turner*, "Engineering Emergence," in Proceedings of the 11th IEEE International Conference on Engineering of Complex Computer Systems, 2006, pp. 89-97.

[6] *C M Macal and M J North*, "Agent-based Modeling and Simulation," in Winter Simulation Conference, Austin, TX, USA, 2009, pp. 86-98.

[7] *M. Dorigo and T. Stützle*, Ant Colony Optimization.: The MIT Press, 2004.

[8] *Dervis Karaboga and Bahriye Akay*, "A survey: algorithms simulating bee swarm intelligence," Artif. Intell. Rev., **vol. 31**, pp. 61-85, 2009.

[9] *Riccardo Poli, James Kennedy, and Tim Blackwell*, "Particle swarm optimization," Swarm Intelligence, **vol. 1**, no. 1, pp. 33-57, 2007.

[10] *Leandro N. de Castro and Jonathan Timmis*, Artificial Immune Systems: A New Computational Intelligence Approach.: Springer-Verlag London, 2002.

[11] *Frank Hutter, Holger H Hoos, Kevin Leyton-Brown, and Thomas Stützle*, "ParamILS: an automatic algorithm configuration framework," J. Artif. Int. Res., **vol. 36**, pp. 267-306, 2009.

[12] *Mauro Birattari, Thomas Stützle, Luis Paquete, and Klaus Varrentrapp*, "A Racing Algorithm for Configuring Metaheuristics," , 2002, pp. 11-18.

[13] *Thomas Bartz-Beielstein, Christian Lasarczyk, and Mike Preuss*, "The Sequential Parameter Optimization Toolbox," in Experimental Methods for the Analysis of Optimization Algorithms, Thomas Bartz-Beielstein et al., Eds. Berlin, Heidelberg, New York: Springer, 2010, pp. 337-360.

[14] *R.J. Allan*, "Survey of Agent Based Modelling and Simulation Tools," Science and Technology Facilities Council, Daresbury Laboratory, Warrington, Technical Report DL-TR-2010-007, 2010.

[15] *Cynthia Nikolai and Gregory Madey*, "Tools of the Trade: A Survey of Various Agent Based Modeling Platforms," Journal of Artificial Societies and Social Simulation, **vol. 12**, no. 2, 2009.

[16] *S. Luke, C. Cioffi-Revilla, L. Panait, K. Sullivan, and G.C. Balan*, "MASON: A Multiagent Simulation Environment," Simulation, **vol. 81**, no. 7, pp. 517-527, 2005.

[17] *Serban Iordache*, "Consultant-Guided Search Algorithms with Local Search for the Traveling Salesman Problem," in 11th International Conference Parallel Problem Solving from Nature - PPSN XI. LNCS, **vol. 6239**, Krakow, Poland, 2010, pp. 81-90.

[18] *Serban Iordache*, "Consultant-Guided Search Algorithms for the Quadratic Assignment Problem," in Hybrid Metaheuristics - 7th International Workshop, HM 2010. LNCS, **vol. 6373**, Vienna, Austria, 2010, pp. 148-159.