INFORMATION SOCIETY TECHNOLOGIES
IST PROGRAMME

RTD PROJECT

Key Action II (Action line II.1.1)

SIMWEB

Exploring
Innovative *e*Business Models
Using
Agent Simulation

State-of-the-art of software tools for agent-based simulations

Proposal Number: **IST-2001-34651**

July 2001

TABLE OF CONTENTS

1. INTRODUCTION

The agent-based approach to simulation endeavours to replace individual actors or groups within any particular system with software representation of those actors (or groups). These software agents may interact – either with one another or with the environment in which they are placed and from the interactions, collective behaviour(s) may emerge. A multi-agent model may be defined as one that includes agents embedded in a simulated environment. The agents are autonomous (they act independently of any controlling intelligence), social (they interact with other agents), communicative (they can communicate with other agents explicitly via some language), reactive (they perceive and respond to changes in the environment) and pro-active (they are goal-driven) [Wooldridge and Jennings, 1995; Ferber 1999; Gilbert and Troitzsch, 1999].

> *Agent-based modelling [is] the set of techniques [in which] relations and descriptions of global variables are replaced by an explicit representation of the microscopic features of the system, typically in the form of microscopic entities ("agents") that interact with each other and their environment according to (often very simple) rules in a discrete space-time [Gross and Strand, 2000: 27]*

Multi-agent systems have several advantages over more conventional simulation models (such as those based on the system dynamics paradigm) (see [Axtell, 2000]):

- Explicit representation of the actors involved (the 'stakeholders'). This makes it relatively easy to validate the model against data about the behaviour of the actors.
- Opportunities for scaling and analysis at several levels. For example, a model may be composed of several sub-models at the organisational level, which in turn can be composed of objects modelling individual actors (such nested models are sometimes called swarms, after the concept popularised in the Swarm modelling system).
- Ability to detect and analyse 'emergent properties', that is properties of interacting entities that are not obvious from considering the behaviour of the individual entities alone. [Epstein and Axtell, 1996]
- Ability to model complexity: that is, behaviour of the system as a whole that is non-linear and sensitive to initial conditions.

Other simulation methods may have some of these characteristics, but the multi-agent approach is especially powerful because it encompasses them all.

There are a number of different tools or frameworks that can be employed to support the implementation of agent-based simulations, avoiding the task of programming simulations from scratch. This review is aimed at both presenting a general survey and offering a recommendation on which agent-based simulation tool appears to be more convenient to found subsequent developments authored by the Simweb consortium. It is based on papers, manuals, tests of available models, analysis of the architecture of the tool, and discussions held with developers of the considered tools besides further discussions with members of the concerned communities.

2. CRITERIA

In order to compare frameworks for agent-based simulation we must firstly establish a number of comparison criteria. In what follows we introduce such criteria intending to encompass both frameworks' common features and desirable features.

2.1 Development facilities.

One of the main aspects when considering choosing a particular software tool for agent-based simulation is the variety and utility of the provided development facilities. Next we classify and dissect such facilities along three major lines:

- *Supported programming language(s)*. We refer to the programming languages that the developer must know in order to build up simulation models. Besides we must take into account whether the particular features of the supported programming languages ease development. Thus, for instance, whereas Java guarantees ease of programming, other languages such as C or Objective C require a more intricate knowledge of the language in order to achieve similar development performance.
- *Availability of a simulation development framework* that eases the rapid creation of models to be subsequently run. Ideally a development framework might allow to graphically define, in a drag-and-drop manner, new models composed out of various pieces (pre-defined models, agents, analysis components, etc.). In addition to this, any desired behaviour not included in the pre-defined components could be specified using some high-level specification language, much simpler than the supported programming languages, particularly designed to integrate well with the embedding software tool. Therefore, users are allowed to do most of their implementation via some relatively easy to use graphical interface or by employing some high-level specification language. This feature is particularly important for users with limited programming experience whose actual interest narrows down to using their computer simulations to obtain results.
- *Extensibility*. It is not surprising that software tools for agent-based simulation are not functionally "complete". They may not offer some particular functionality strongly required by users. For instance, some users might be interested in incorporating genetic algorithms or neural networks into the learning features of their agents. In such cases, it would be desirable for users to be able to extend the chosen tool with additional libraries that provide further functionality. This capability is of particular importance for the Simweb project since it is intended to develop additional, component-based libraries for agent-based simulation, which complement and extend the chosen tool.

2.2 Flexibility.

Does the software tool provide an adequate way to express our model and program the simulation? Although there is no easy way to answer this question, a thorough analysis of publicly available models can help identify commonalities to gain an insight into the appropriateness of the software tools under analysis.

2.3 Compatibility.

This criterion is concerned with the portability of the developed model to other simulation frameworks. It would be valuable to have the chance to easily share our findings, and in particular, our code. Not only is code sharing a major benefit but also migration capability. In the latest case, we refer to the possibility of migrating the model to a different software tool in the eventuality of coming up against serious limitations that prevent us to successfully complete our simulations. The ideal situation would depict all the simulation platforms utilising the same general approach to model development. For example, if a given software tool uses an object-oriented approach then it is easier to translate the code to another tool which also uses an object-oriented approach than it is to translate the code to a tool using a procedural based approach.

2.4 Portability.

Although some simulation frameworks are available for different hardware platforms, models developed for different hardware platforms are no interchangeable. In other words, a given model developed in a Linux might not run on a Mac. Nowadays, most software tools tend to support the Java programming language in order to guarantee portability to the largest number of hardware platforms.

2.5 Facilities to publish simulations on the Internet.

Another aspect to consider is the dissemination of the simulation. In the past, researchers have used video clips to demonstrate their simulations (e.g. making QuickTime movies etc.). Nowadays there is an increasing trend to make Internet simulations more interactive, allowing people to experiment with the simulator (changing parameters, etc.). Some of the simulation tools allow the developer to create easily interactive Internet simulations in just a few easy steps.

2.6 Support.

An important aspect to take into account when evaluating software tools is the sources of support (manuals, newsgroups, support groups, etc.). If the developer gets into trouble when implementing, it is very useful to be able to draw on outside help and examples.

2.7 Performance and scalability.

In our view, this is one of the major issues when considering choosing a simulation framework. This issue becomes capital when attempting at developing large models involving hundreds, or probably thousands of agents. If this is the case it is important to count on a simulation framework that does offer reasonable performance for large agent populations, particularly when run over lengthy simulations. Both performance and scalability shall strongly depend on the supported programming languages' features (simulations for models developed in C are expected to be faster than those for models coded in Java), the parallelisation capabilities offered by the simulation framework (if any), and the distribution capabilities also offered by the simulation framework. Notice that distribution becomes appealing when dealing with models whose agents do a large amount of independent computation without much interaction. In other words, support for distribution is expected to be helpful to handle data intensive instead of communication intensive agent models.

2.8 Model availability.

The availability of models contributed by community users of a given software tool is particularly important to help developers gain understanding as to how to develop actual models. A large variety of tested models may help developers to find similar models from which they can depart. In addition to this, available source code also may help ease development. Furthermore, a rich library of publicly available models appears as a good indicator of the stability of the software tool.

2.9 Display facilities.

Whether the software provides facilities to graphically display data generated by the simulation. It is important for both developers and users to count on means of graphing grids, cells, statistics, etc. Not only is important to graph the data produced through the simulation, but also the structure, activity and evolution of the multi-agent system defined by the model as a whole. In the latest case, it is interesting for the modeller to count on structures such as graphs, cells, grids, networks, etc., that mimic the model in order to better identify and understand the activity of the system through its visual display.

2.10 Data export.

Whether the tool is capable of formatting simulation data in some sort of standardised scientific format. This makes it possible to share data easily, and also to build and share tools for accessing and analysing such data, particularly via languages and environments for statistical computing and graphics.

3. SOFTWARE TOOLS FOR AGENT-BASED SIMULATION

In what follows we dissect in detail a selected number of software tools for agent-based simulaiton. Our selection process aimed at choosing general-purpose multi-agent simulation frameworks currently supported by ongoing projects, non-profit organisations or companies. By general purpose we mean that the selected framework shall ideally support the development and deployment of agent-based models of varying characteristics in a wide range of application domains.

3.1 Swarm

Swarm [Swarmwww; Swarm Online Doc] is arguably the best known of all the selected frameworks in this section. It has a long tradition, serving as inspiration to other frameworks such as RePast and Ascape (both analysed below). Swarm was originally developed at the Santa Fe Institute and subsequently continued and maintained by the Swarm Development Group (SDG), a not-for-profit organisation dedicated to advancing the state-of-the-art in multi-agent based simulation through the continued advancement of the Swarm Simulation System and support of the Swarm user community.

Swarm is a multi-agent software platform for the simulation of complex adaptive systems. In the Swarm system the basic unit of simulation is the swarm, a collection of agents executing a schedule of actions. The swarm represents an entire model: composed of agents as well as the representation of time as depicted in Figure 2. Swarm supports hierarchical modelling approaches whereby agents can be composed of swarms of other agents in nested structures. In this case, the higher-level agent's behaviour is defined by the emergent phenomena of the agents inside its swarm. This multi-level model approach offered by Swarm is very powerful. Multiple swarms can be used to model agents that themselves build models of their world. In Swarm, agents can own swarms themselves, models that an agent builds for itself to understand its own world. Swarm allows a user to build a simulation as a state machine in which all the changes to the state machine occur through a schedule. A simulation program based on Swarm schematically contains three types of objects:

- A **model** that creates and controls the activities of agents in the model.
- An **observer** that collects information from agents (observations) to either write it out to a file or display it in real time.
- Objects corresponding to different **agents** in the model (firms, consumers, producers, etc.) and for some aggregate agents (markets, industries, economies) that regroup actions and aggregate properties (market price, concentration index of the industry, etc.).

Swarm provides object-oriented libraries of reusable components for building models and analysing, displaying, and controlling experiments on those models. Swarm effectively provides a very complete set of libraries for managing agents, the spatial structures for their environment, their activities, and the aggregation of these activities and the analysis of their results. It is written in Objective C, a programming language across Smalltalk and C. There is a Java Native Interface available on which a Java-based version of Swarm has been developed. The Java version makes Swarm developments highly portable. Swarm has a very steep learning curve. It is necessary to have experience of Java (or Objective C), be acquainted with the object-oriented methodology and be able to learn some Swarm code. For this purpose there is no development framework available to ease up the intricate task of building, running and analysing models.

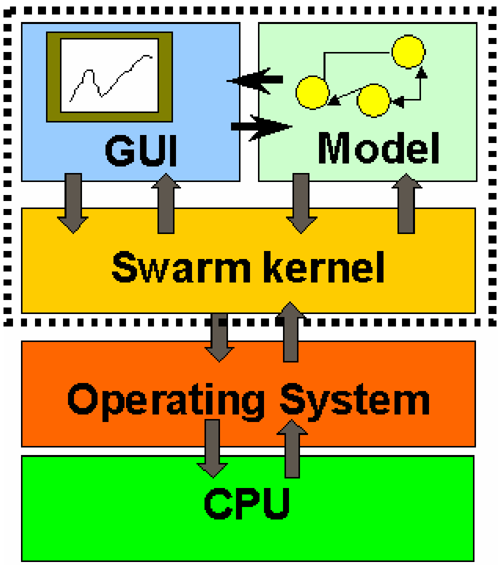


Figure 1. Swarm as virtual multicomputer

The actual model and the task of observing the model are separated in Swarm. There are special 'observer' objects whose purpose is to observe other objects via the probe interface (see Figure 2). These objects can provide both real-time data presentation and storage of data for later analysis (Figure 3 and Figure 4 respectively show how to parameterise the Artificial Stock Market model and some real-time data produced during a simulation). In fact, the observer objects are swarms. Combining the observer swarm with the model swarm gives an observable representation of the model to be subsequently run by the Swarm infrastructure as depicted in Figure 1 . Other simulation tools don't show such clear distinction between the actual model and the code needed to observe and collect data from the model, making models too sensitive to changes.

Major issues with Swarm have to do with performance and scalability [Daniels, 2000]. The Swarm community seems to agree on the capability of the Objective C of Swarm to handle large multi-agent models. However, there is no such agreement on the Java version of Swarm mainly due to Java rather than Swarm. Swarm users have reportedly experienced serious Java memory problems. It seems though that the support provided by the Swarm Development Group and the new versions of the virtual machine are contributing to overcoming such problems. And yet Objective C models perform and scale much better than Java models.

As to data export, Swarm offers explicit support to store simulation data in HDF5 [HDF5www], a general-purpose library and file format for storing scientific data, to be subsequently processed by statistical packages (e.g. GNU R [GNU Rwww], a language and environment for statistical computing and graphics).

In terms of support, there are excellent support mailing lists and a remarkable amount of publicly available Swarm code[1] contributed by a large community of Swarm users which have also contributed to extend the framework with third-party libraries (e.g. for genetic algorithms and neural networks). A distinguishing feature for Swarm models is that they can be easily shared with users of Swarm-like simulation frameworks such as RePast or Ascape. There is also an annual meeting of the Swarm Users Group, the SwamFest, where researchers

[1] Notice though that the number of available models in Objective C is much greater than the number of models in Java.

from diverse disciplines present their experience with multi-agent modelling and the Swarm package. Swarm models can already run inside a web browser, specifically Netscape 6. However, a future development goal is for Swarm to be a complete interactive, browser-based development environment for agent-based models.

Finally, the Objective C version of Swarm is available for quite a large number of platforms, namely Windows 9x, Windows NT, Windows 2000, Debian Linux, Red Hat Linux, SuSE Linux, and Solaris.

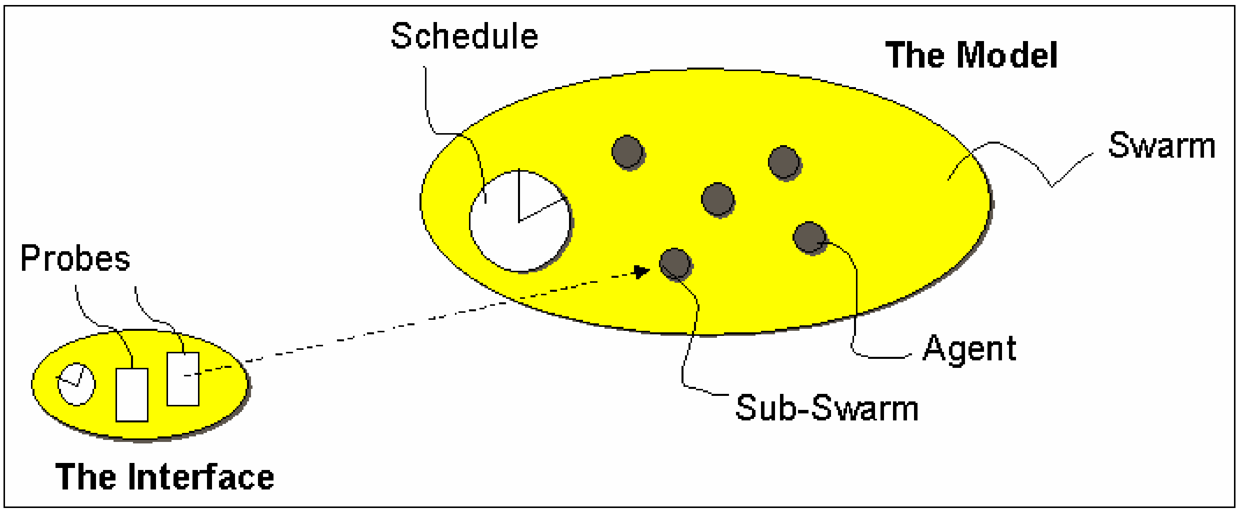
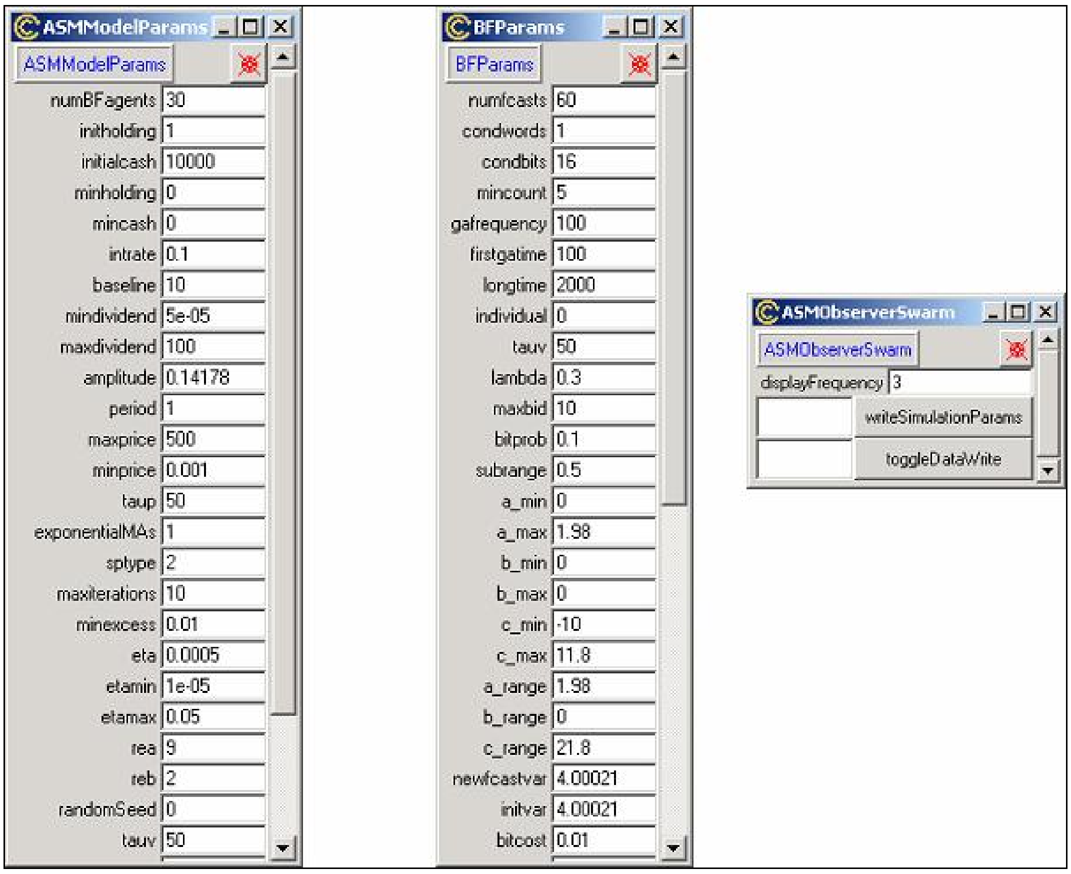


Figure 2. Swarm simulation structure



ASMModelParams

numBFagents	30
initholding	1
initialcash	10000
minholding	0
mincash	0
intrate	0.1
baseline	10
mindividend	5e-05
maxdividend	100
amplitude	0.14178
period	1
maxprice	500
minprice	0.001
taup	50
exponentialMAs	1
sptype	2
maxiterations	10
minexcess	0.01
eta	0.0005
etamin	1e-05
etamax	0.05
rea	9
reb	2
randomSeed	0
tauv	50

BFParams

numfcasts	60
condwords	1
condbits	16
mincount	5
gafrequency	100
firstgatime	100
longtime	2000
individual	0
tauv	50
lambda	0.3
maxbid	10
bitprob	0.1
subrange	0.5
a_min	0
a_max	1.98
b_min	0
b_max	0
c_min	-10
c_max	11.8
a_range	1.98
b_range	0
c_range	21.8
newfcastvar	4.00021
initvar	4.00021
bitcost	0.01

ASMObserverSwarm

displayFrequency	3
	writeSimulationParams
	toggleDataWrite

Figure 3. Model, agents' and observer parameterisation for the Artificial Stock Market simulation

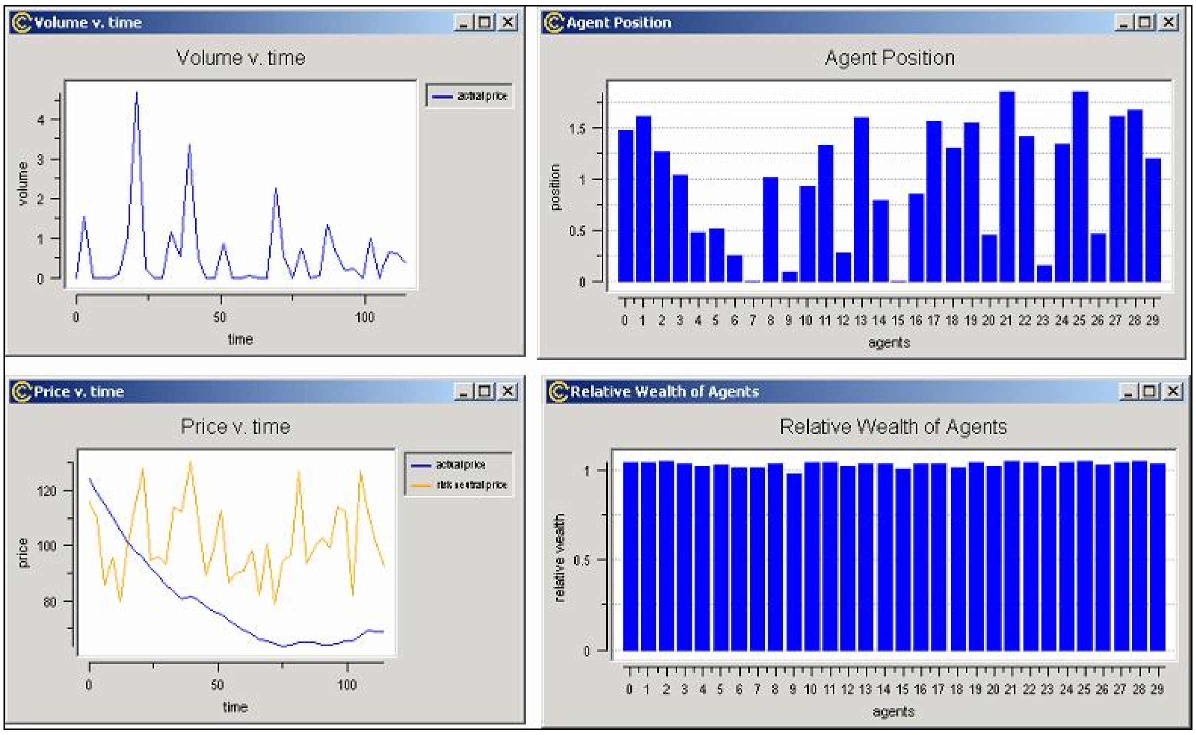


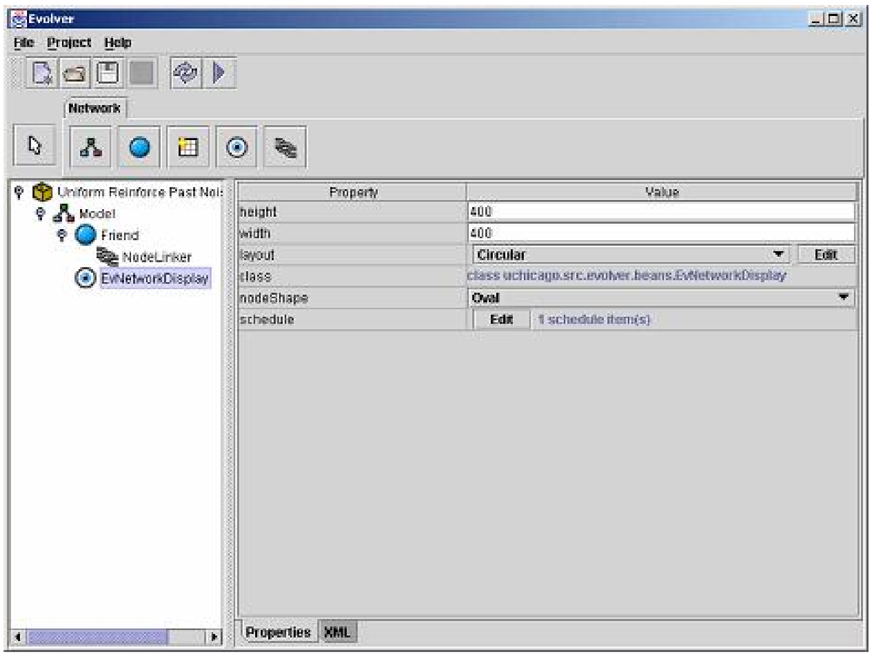
Figure 4. A sample of graphs produced by the Artificial Stock Market simulation

3.2 RePast

RePast [Collier; Repastwww] is a software framework developed by the University of Chicago's Social Science Research Computing for creating agent-based simulations using the Java language. RePast is rapidly gaining popularity. It provides a library of classes for creating, running, displaying and collecting data from an agent-based simulation. RePast borrows much from the Swarm simulation toolkit and can properly be termed "Swarm-like".

RePast envisions a simulation as a state machine whose state is constituted by the collective states of all its components. These components can be divided up into infrastructure and representation. The infrastructure is the various mechanisms that run the simulation, display and collect data and so forth. The representation is what the simulation modeller constructs, the simulation model itself. The state of the infrastructure is then the state of the display, the state of the data collection objects, etc. The state of the representation is the state of what is being modelled, the current values of all the agents' variables, the current value of the space or spaces in which they operate, as well as the state of any other representation objects. In RePast as in Swarm, any changes to the states of the infrastructural components and the representational components occur through a Schedule object. In short, RePast allows a user to build a simulation as a state machine in which all the changes to the state machine occur through a schedule. This provides clarity and extensibility both for the simulation writer/user as well as the software designer seeking to extend the toolkit.

Several other parts of RePast follow the Swarm paradigm and should be familiar to users of Swarm. This feature is particularly important for compatibility purposes. Thus models developed for RePast can be easily migrated to Swarm and shared with Swarm developers.



Property	Value
height	400
width	400
layout	Circular Edit
class	class uchicago.src.evolver.beans.EvNetworkDisplay
nodeShape	Oval
schedule	Edit 1 schedule item(s)

Figure 5. Evolver development environment

RePast offers *Evolver*, a rapid simulation development environment for creating network simulations (see Figure 5). Using a drag-and-drop model, a simulation can be graphically composed out of various pieces (pre-defined models, agents, analysis components, etc.). Any desired behaviour not included in the pre-defined components can be specified using NQPython (Not Quite Python), a Python-like[2] language specifically designed to integrate well with RePast and much simpler than Java. Notice though that the environment is limited to supporting the specification of network simulations (see Figure 6).

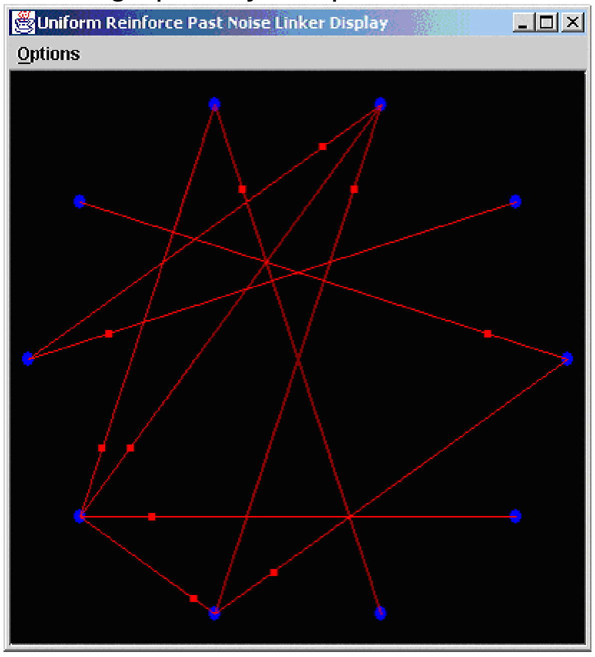


Figure 6. Example of network simulation with Evolver

RePast abstracts most of the key elements of agent-based simulation and represents them as a Java class or classes [Repast HowTos]. These classes cooperate to make a framework for creating agent-based simulations. RePast provides a ready-to-use class or classes for most of the common infrastructural abstractions of an agent-based simulation (e.g., scheduling, display, data collection, and so forth) and a variety of generic components for constructing representational elements. These generic components include such things as agent spaces (grids, torii, “soups,” etc.) and a few generic agent types. RePast is particularly strong in its support for network (social and otherwise) simulations. This support is both infrastructural (graph layouts, network generation, saving and loading) and representational (default node and edge classes).

One major RePast’s aim is to move beyond the representation of agents as discrete, self-contained entities in favour of a view of social actors as permeable, interleaved and mutually defining, with cascading and recombinant motives. RePast is intended to support the modelling of agents, organisations and institutions as recursive social constructions (the name RePast is an acronym for *REcursive Porous Agent Simulation Toolkit*). Nonetheless the current version does not offer explicit support for organisations and institutions as higher-level entities, though it does fully support social networks.

Extensibility is provided by the use of Java as an implementation language. Object-oriented languages easily lend themselves to the creation of extensible frameworks through the use of inheritance and composition. Furthermore Java ensures the portability of the models to all hardware platforms capable of running the Java virtual machine.

2 http://www.python.org

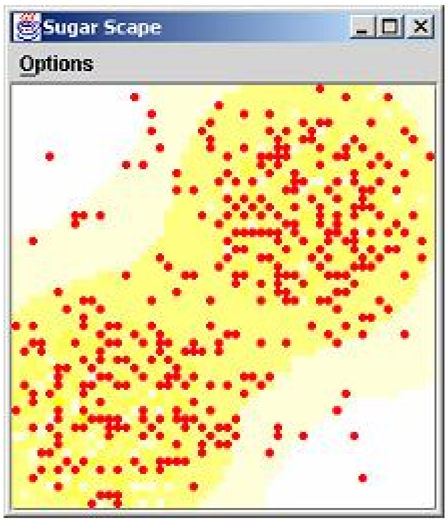


Figure 7. Sugarscape simulation. Population map

RePast provides a few, well-known models such as *Sugarscape*, *Heatbugs*, *Mousetrap*, *The Game of Life*, etc (see Figure 7). Apart from the models accompanying the software distribution there are very few other publicly available models. The community of users is still small compared to more popular simulation frameworks such as Swarm, and so the number of models to resort to at development time. However, some remarkable have been developed with the aid of RePast. Here we should mention the integrated model of the electric power and natural gas markets implemented by Michael J. North at Argonne National Laboratory [North].

Although performance optimisations were not part of the initial design, RePast reportedly offers good performance according to a survey conducted among some community users. RePast is said to support parallelisation in the future. So far, it has proven to behave well when running large models by distributing simulations via RMI. Although this technique is highly valuable when running large, data-intensive models (involving agents that do a lot of individual computation), it is not so convenient to handle large, communication-intensive models.

As to display support, RePast includes such features as run-time model manipulation via gui widgets. The display mechanism is responsible for the graphical animated visualisation of the simulation as well as providing the capability to take snapshots of the display and make QuickTime movies of the visualization as it evolves over time. QuickTime movies can be easily published on the Internet. It also includes support for the probing of the displayed objects. The display mechanism also contains the graph layouts used to visualise networks and extensible classes that can be used to build custom displays.

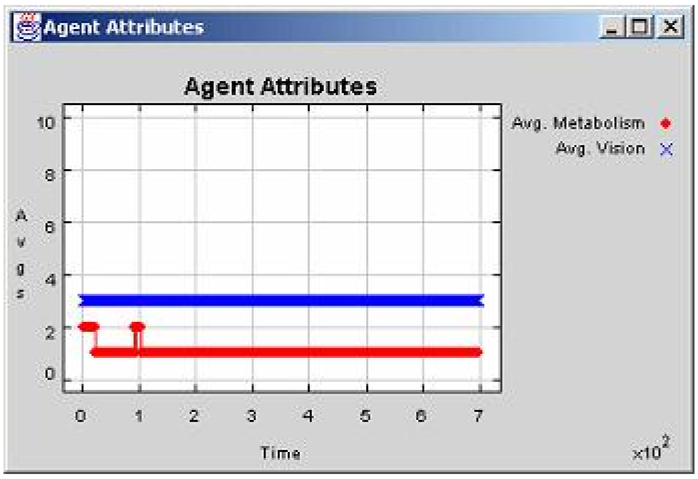


Figure 8. Sugarscape simulation. Dynamic features of agents

Although RePast offers a complete suite of display mechanisms, there is no explicit support to format the data produced by simulations into some sort of standardised scientific format.

As for support, RePast has an institutional presence behind it. It is expected to be supported in the future. There are currently two developers working on and supporting RePast, Nick Collier and Tom Howe. They provide support through the RePast-interest email mailing list. Furthermore, RePast is well documented with papers, user guides and javadoc-generated API help.

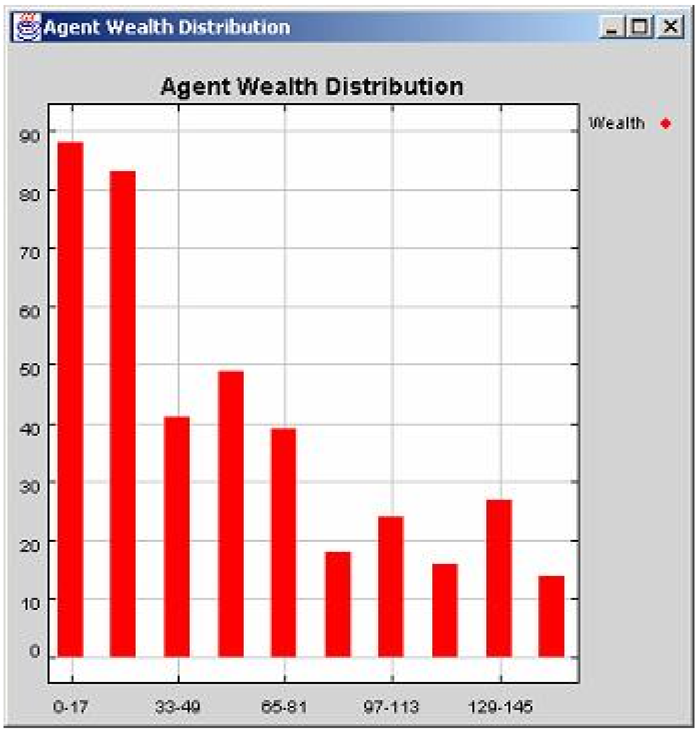


Figure 9. Sugarscape simulation. Agetn wealth distribution graph

3.3 Ascape

Ascape [Parker, 2001; Ascapewww] is a framework for developing and analysing agent-based models developed by the Brookings Institute, the Center on Social and Economics Dynamics.

Ascape is written entirely in Java and models are written in Java as well, using the API the framework provides. Particularly, models can be programmed using object-oriented techniques. The framework provides classes with the usual roles, though as of this writing there is no scheduler, so communication between agents might be somewhat limited for our purposes. Since there is no development environment, coders use their preferred programming editor.

Tools for easy generation of graphs are provided. The framework manages graphical views and collections of statistics for scapes and offers mechanisms for controlling and altering parameters for scape models. Also model data can be exported in CSV files for further analysis or visualisation.

Ascape runs in any platform for which there is an implementation of the JDK (or just of the JRE in case of model execution). This includes Solaris, MacOS, Win32, Linux, and the BSDs, for instance. Moreover, since the runtime engine can be put in a Java applet simulations can be executed from most web browsers.

According to the information in their home page, the Brookings Institute does not give direct support for Ascape, but there is a mailing list more or less active. Nonetheless BiosGroup, Inc. DC has taken over the Ascape development for which a new release is planned. There is almost no user-level documentation. There are better docs for developers, but in form of Javadoc: The API is somewhat documented, there is a brief overview of how to implement models in the documentation of the class `edu.brook.ascape.model.Scape`, and the code of the model `edu.brook.pd.PD2D` is commented to some extent. Besides, for about 20 models come with source code in the Ascape distribution.

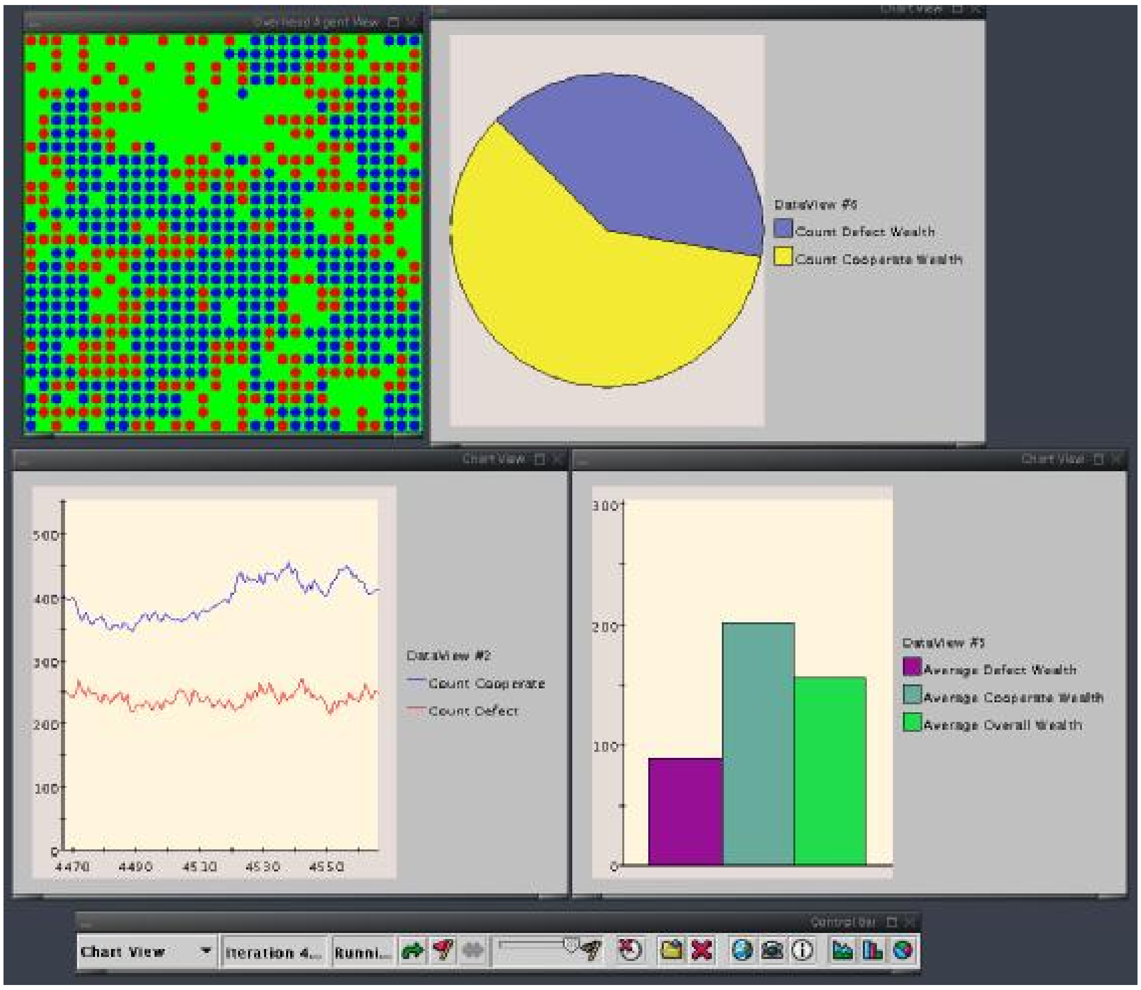


Figure 10. Ascape simulation of Prisoner's dilemma

3.4 NetLogo

NetLogo [Wilensky, 2001; NetLogowww; NetLogo Manual] is the next generation of a series of multi-agent parallel modelling and simulation environment developed at the Center for Connected Learning and Computer-Based Modeling, Northwestern University. NetLogo is the multi-platform successor of the Mac-only StarLogoT [StarLogoTwww], which in turn is a superset of StarLogo [StarLogowww], the programmable modelling environment developed by the Epistemology and Learning Group at the MIT Media Laboratory.

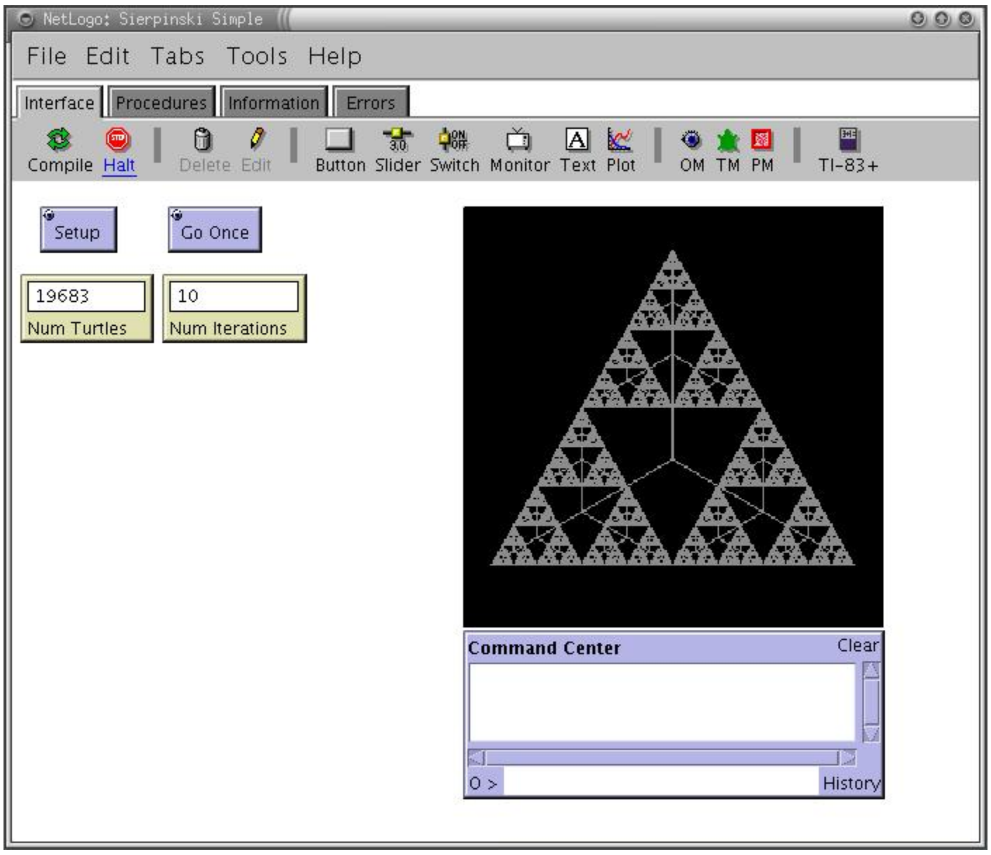
NetLogo is written in Java hence runs in any platform for which there is an implementation of the JDK (or just of the JRE in case of model execution). This includes Solaris, MacOS, Win32, Linux, and the BSDs, for instance.

NetLogo can be executed standalone or as a Java applet within any Java capable web browser, but models are coded using NetLogo's own programming language, kind of a Logo dialect extended to support agents and parallelism, which is appropriate for learning, but maybe too simple compared with a general-purpose, object-oriented programming language as Objective C or Java.

NetLogo is intended to allow non-programmers and students to develop quite complex simulations by providing a range of end-user tools, though professional developers would surely prefer more expressiveness in our opinion.

The framework comes with an editor with syntax highlighting (see Figure 12) for its language, a command shell (see Figure 11), and a shapes editor (see Figure 13). Documentation is excellent. NetLogo has extensive documentation and tutorials for all of its features. It also comes with a models library, which is a collection of about one hundred pre-written simulations that can be used and modified. These simulations address many areas in the natural and social sciences, including biolog, medicine, physics, chemistry, mathematics, computer science, economics, and social psychology.

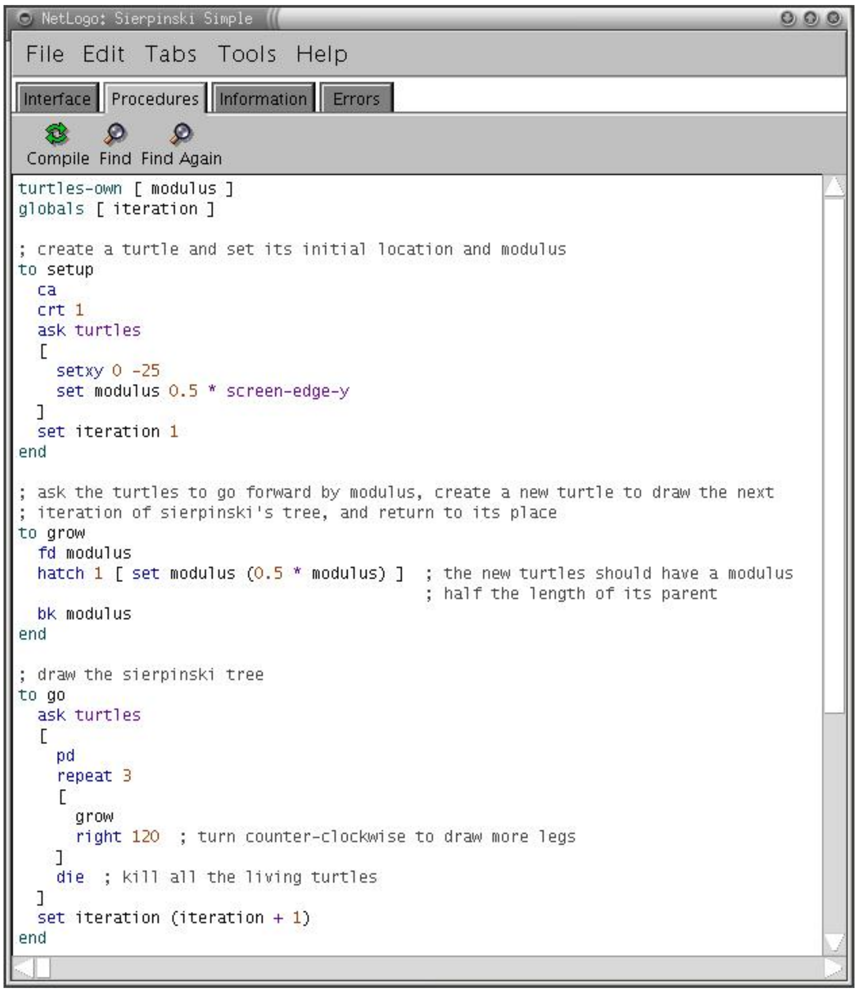
Graphs of different kinds are easy to generate as models evolve using language commands, the framework manages graphical views and collections of statistics for turtle sets and offers mechanisms for controlling and altering parameters of models. Also model data can be exported in CVS format for further analysis or visualization.


NetLogo: Sierpinski Simple
File Edit Tabs Tools Help
Interface Procedures Information Errors
Compile Halt Delete Edit Button Slider Switch Monitor Text Plot OM TM PM TI-83+
Setup
Go Once
Command Center
Clear
O >
History

19683
Num Turtles

10
Num Iterations

Figure 11. Netlogo simulation of Sierpinsky's triangle


NetLogo: Sierpinski Simple
File Edit Tabs Tools Help
Interface Procedures Information Errors
Compile Find Find Again

```
turtles-own [ modulus ]
globals [ iteration ]

; create a turtle and set its initial location and modulus
to setup
  ca
  crt 1
  ask turtles
  [
    setxy 0 -25
    set modulus 0.5 * screen-edge-y
  ]
  set iteration 1
end

; ask the turtles to go forward by modulus, create a new turtle to draw the next
; iteration of sierpinski's tree, and return to its place
to grow
  fd modulus
  hatch 1 [ set modulus (0.5 * modulus) ]  ; the new turtles should have a modulus
                                           ; half the length of its parent
  bk modulus
end

; draw the sierpinski tree
to go
  ask turtles
  [
    pd
    repeat 3
    [
      grow
      right 120  ; turn counter-clockwise to draw more legs
    ]
    die  ; kill all the living turtles
  ]
  set iteration (iteration + 1)
end
```

Figure 12. Netlogo's development environment

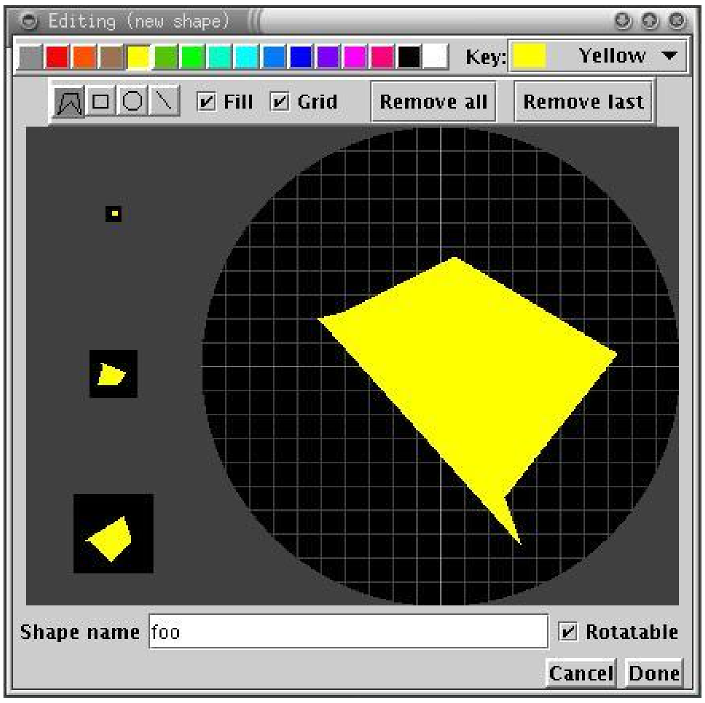


Figure 13. Netlogo shapes editor

3.5 AgentSheets

AgentSheets [Dugdale] is a commercial agent-based simulation tool specifically aimed at non-programmers. AgentSheets users express their models with the Visual AgentTalk tactile and rule-based language, which provides an easy to use, visual development framework. AgentSheets uses the visual programming paradigm meaning that there is no actual text based coding and all the development is done via a graphical interface (dragging and dropping elements from toolboxes, etc.). Indeed, the ease of use of AgentSheets is its greatest advantage. Agents are created in a window called a “gallery” and have an associated behaviour specified by sets of rules (called methods) and events.

The way that AgentSheets operates is intuitively easy to understand which makes it very quick to develop simple simulations. For this reason, it is widely used for teaching the principles of simulation to students. However, once the simulation models begin to become more in-depth, then the weaknesses of AgentSheets become apparent. With regard to simulation in the social sciences, two particular limitations of AgentSheets may cause problems: an agent cannot send information to another agent, being problematic when modelling the communication of information between human agents; an agent cannot change the attribute of another agent. No doubt, there may be ways to work around these problems, but if we need to model frequently these situations in our simulations, it would make the simulations complicated and inefficient. In addition, there is no 'long distance' vision making it impossible to examine the status of an agent elsewhere on the grid.

AgentSheets very easily generates Java applets and beans, which allow the simulation models to be interactively run through a Java capable web browser.

AgentSheets is available for the Mac and Wintel platform.

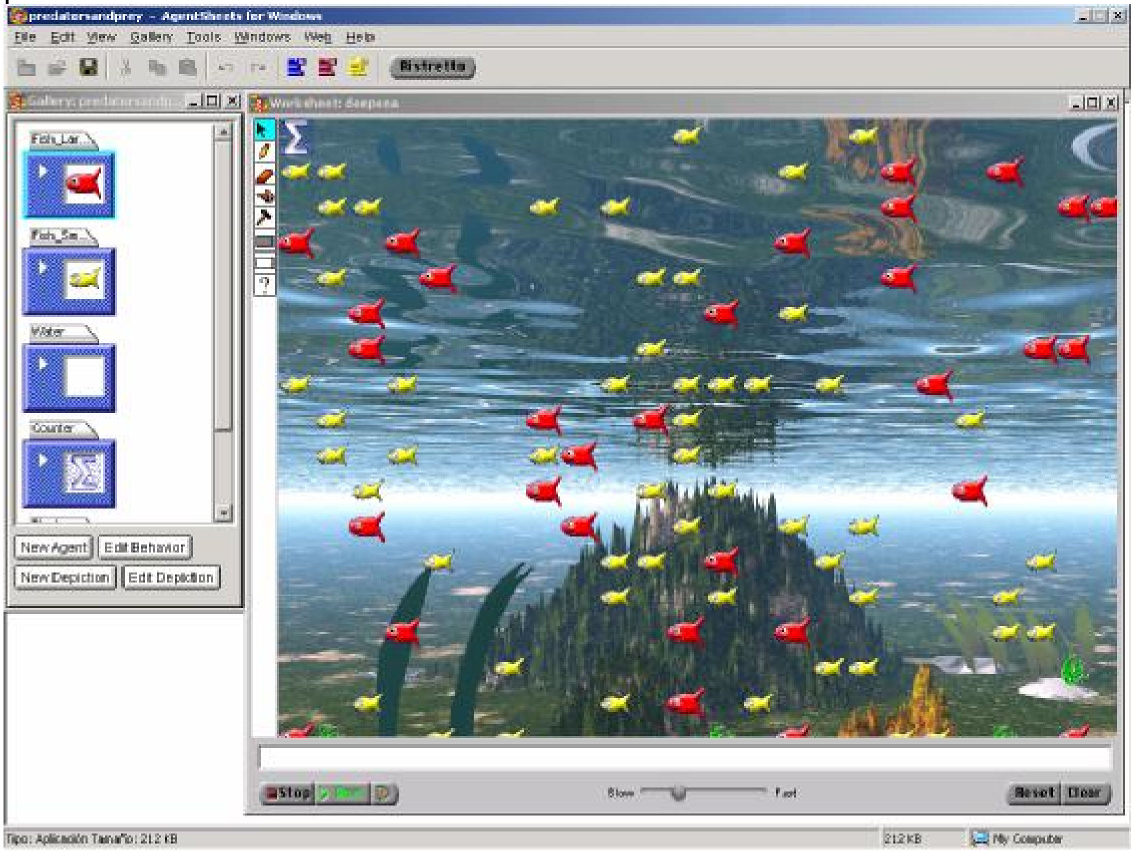
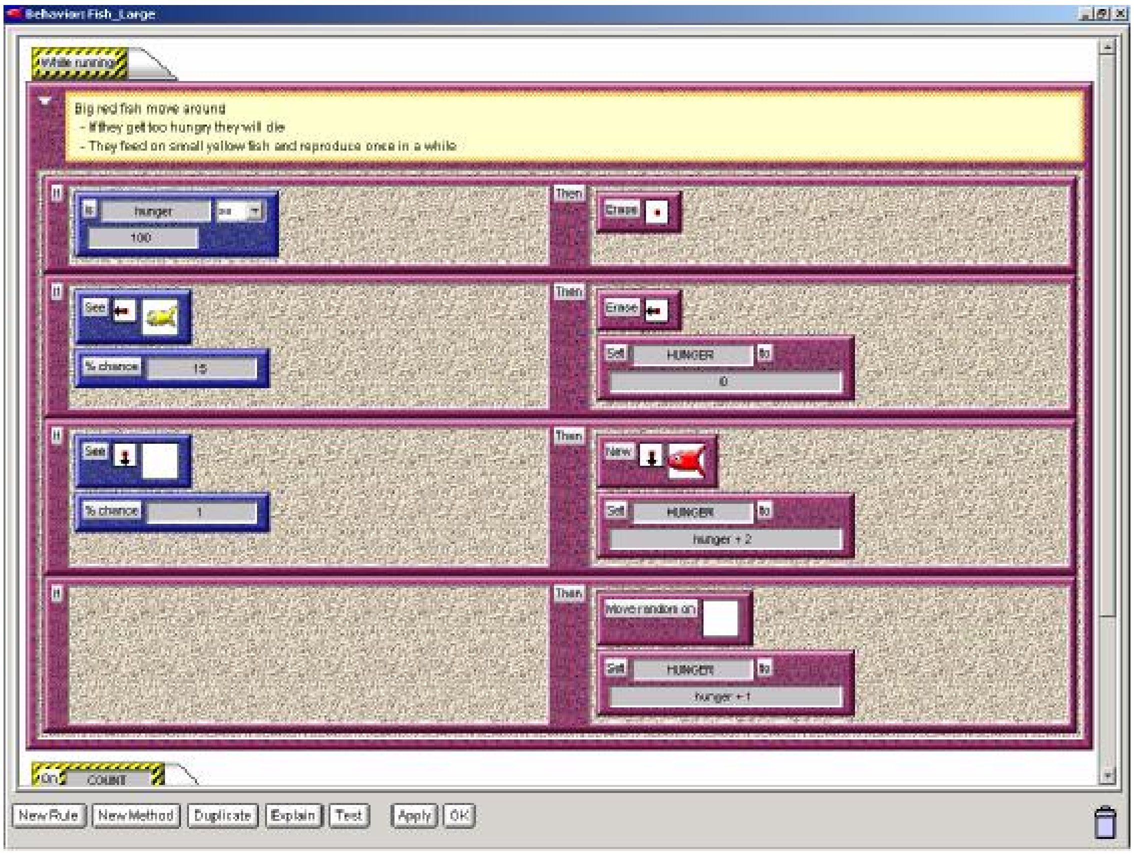


Figure 14. Agentsheets simulation of a predators and prey model



Big red fish move around
- If they get too hungry they will die
- They feed on small yellow fish and reproduce once in a while

If: Is hunger >= 100 — Then: Erase
If: See; % chance 15 — Then: Erase; Set HUNGER to 0
If: See; % chance 1 — Then: New; Set HUNGER to hunger + 2
If — Then: Move random on; Set HUNGER to hunger + 1

Figure 15. Rule-based agent behaviour definition

3.6 MAML

MAML[MAML; MAML Tutorial; Dugdale] was developed by the Complex Adaptive Systems Lab. at the Central European University in Hungary, which it seems it was interrupted for about a couple of years ago due to lack of funding. Further development would be unlikely.

The language was initially developed to help social science students with limited programming experience create agent-based models quickly. The ultimate goal of the project is to develop an easy to use environment (complete with a graphical interface). However, the present version of MAML is, as the name suggests, a programming language and not an environment and, according to MAML's web site, the current version of the language is indeed in an alpha stage.

MAML actually sits on top of Swarm and is intended to make Swarm easier to use by providing macro-keywords that define the structure of the simulator and access the Swarm libraries. MAML works at a higher level of abstraction than Swarm with clearer constructs. However, in addition to learning MAML, the developer would need to know Objective C and also Swarm. This point currently limits the usefulness the language to inexperienced programmers. Indeed, experienced programmers may actually prefer the added functionality of Swarm and the additional resources available. Programming using MAML requires the developer to create text files using a text editor since there is no developer's interface to MAML. Since MAML accesses the Swarm libraries, the interface of the developed simulation model is very similar as to what would appear if Swarm were used.

The code written using MAML is converted into a Swarm application via the MAML compiler (called xmc), which as of this writing has just an alpha version. The resulting application is then compiled in the same way as normal Swarm code by gcc. Currently the MAML compiler only runs on PCs under GNU/Linux (running the compiler on a Mac and PC/Windows NT is untested).

Some background documentation on MAML is provided: a tutorial with source code, reference manual, and a technical manual (knowledge of Swarm is needed). However, there is no support via mailing lists. The MAML home page contains some fairly simple examples of common simulations. Unfortunately, outside of the MAML home page there are very few examples of simulations using MAML.

With respect to the suitability of MAML to social science simulation, most of the effort in developing MAML has been devoted to simplifying the programming effort rather than providing facilities specifically geared towards modelling social science mechanisms.

3.7 SDML

SDML is a modelling language with the following features:

- Knowledge is represented on rule bases and databases
- All knowledge is declarative
- Models can be constructed from many interacting agents
- Complex agents can be composed of simpler ones
- Object-oriented facilities, such as multiple inheritance, are provided
- Temporal facilities are provided, including different levels of time
- Rules can be fired using forward and backward chaining

Steve Wallis implemented SDML in Smalltalk in consultation with other members of the Center for Policy Modeling. It evolved from a non-declarative modelling language implemented by Scott Moss.

SDML provides a simple development environment to put together the models written in the language. SDML is a declarative language, where problems a coded following the logic-programming paradigm. The state and behaviour is represented by means of rule sets and databases, and the model evolves by forward and backward chaining. Agents may be assigned rules that determine their behaviour and which can be shared with other agents. The fact that it is strongly grounded in logic allows formal proofs of the completeness of the model to be constructed.

Sophisticated simulations may be built using SDML involving complex interacting organizations, deeply nested levels of agents, and the ability for agents to possess limited cognitive abilities. However, as the developers admit, the language has a steep learning curve.

SDML was thought as a tool for developing simulations in the social sciences, however, most of the available models have to do with economic and market modelling, which is a good point for us indeed. Nevertheless, apart from the SDML Home Page at Manchester Metropolitan University, there are very few examples of simulations using SDML, as far as we can tell.

SDML does however provide features useful in modelling cognitive social agents. There is no inherent theory of cognition implemented in SDML so any agent cognition is represented as sets of rules. Communication between agents is achieved via databases: the result of a fired rule is written to an agent's database, which may be accessed via another agent. The accessibility of one agent's database to another agent's database can be restricted by assigning a status to the rule's clause (e.g. private or public). Agents may also evaluate each other as being possible 'collaborators' and endorse other agents as being a reliable, unreliable, successful, or unsuccessful collaborator.

SDML is available for MS Windows 3.1/95/98/2000/NT, GNU/Linux, PowerMac, and Unix ADUX/AIX/HPUX/SGI/Solaris.

4. CONCLUSIONS

Prior to offering a recommendation we must firstly state the features of a software tool for agent-based simulation that we deem more desirable.

- *Development facilities.* Firstly, we strongly prefer to count on an object-oriented programming language for the sake of clarity, abstraction and flexibility. Secondly, we do not regard the existence of a simulation development framework as compulsory. Nonetheless, such a facility might serve as reference and inspiration when developing the graphical simulation environments for building models that the Simweb consortium is committed to offer. Lastly, we do require the selected framework to be highly extensible in order to facilitate the addition of the extensions required in order to build the models proposed by the Simweb consortium.
- *Flexibility.* Since we plan to build agent-based models it would be interesting to count on a framework that offers explicit programming support for the notion of agent, and eventually for higher level structures (e.g. organisations, institutions, social networks, etc.).
- *Performance and scalability.* We shall ideally require a framework supporting both parallelisation and distribution in order to eventually run large simulations involving thousands of agents.
- *Model availability.* A large number of models, from toy models to actual models, of varying features are desirable as a reference to programmers at development time.
- *Compatibility.* High compatibility with other software frameworks will aid both to ease eventual migrations and to share source code.

- *Support.* A widely used framework with an extensive community of active users, good documentation, and active mailing lists will enormously help the sound development of the Simweb models. Active development of the framework, and the availability of the source code and the legal possibility of introducing changes in order to fix eventual bugs, is strongly preferred.
- *Portability.* Although portability is not a must, we'd rather prefer to have different choices when developing and deploying our models.
- *Display facilities.* An ample, complete set of display facilities will help create the most appropriate real-time visualisations to help users graphically interpret and analyse simulations.
- *Data export.* These facilities are particularly important to help us export data from simulations in order to conduct further, subsequent numeric analysis and visualisation.
- *Internet publishing.* The publishing of models on the Internet may help remote users to interact with the system by running their own simulations. This feature is particularly interesting so as to organise business games involving distributed participants.

Based on the desiderata above and the analysis compiled in Table 1 Swarm and RePast appear as the most powerful software tools for agent-based simulation. At this point it is tremendously hard to discard one of them as a candidate to found the Simweb developments. Thus we propose to carry out performance and scalability tests over: (1) common models available for both frameworks (e.g. Heatbugs); and (2) simplified, preliminary versions of the Simweb models.

Table 1. Comparison among the analysed software tools for agent-based simulation based on the criteria in Section 1.

	Development Facilities	**Flexibility**	**Compatibility**	**Portability**	**Internet publishing**	**Support**	**Performance/ Scalability**	**Models**	**Display**	**Data export**
Swarm	Medium	High	High	High	Low	High	Medium[3]	High	High	High
RePast	High	High	High	High	High	High	Medium	Medium	High	None
Ascape	Low	High	High	High	High	Low	Medium	Medium	High	Low
AgentSheets	Medium	Low	Low	Low	Low	High	Low	Medium	High	Low
MAML	Low	High	Medium	Low	Low	Low	High	Low	High	High
NetLogo	High	Medium	Low	High	High	High	Low	High	High	Low
SDML	Medium	High	Low	High	Low	Medium	Medium	Low	Low	None

[3] This valuation has been made on the Java version of Swarm. As to the Objective C version, our valuation is *High*.

5. BIBLIOGRAPHY

Axtell, Robert. (2000). *Why Agents? On the Varied Motivations of Agent Computing in the Social Sciences.* Brookings Institute. Center on Social and Economic Dynamics. Working Paper 17.

Epstein, J.M. and R. Axtell (1996) Growing Artificial societies: social science from the bottom up. MIT Press, Cambridge, MA.

Ferber, J. (1999) Multi-agent systems: an introduction to distributed artificial intelligence. Addison-Wesley, New York.

Gilbert, N. and K.G. Troitzsch (1999) Simulation for the Social Scientist. Open University press, Milton Keynes.

Gross, D and R. Strand (2000) Can agent-based models assist decisions on large-scale practical problems? A philosophical analysis. Complexity 5: 26-33.

Wooldridge, M and N.R. Jennings (1995) Intelligent agents: theory and practice. Knowledge Engineering Review 10:115-152.

Collier, Nick. *RePast: An Extensible Framework for Agent Simulation*.

Miles T. Parker (2001). *What is Ascape and Why Should You Care?*. *Journal of Artificial Societies and Social Simulation* vol. 4, no. 1, http://www.soc.surrey.ac.uk/JASSS/4/1/5.html.

North, M.J. (forthcoming) .*Technical Note: Multi-agent Social and Organizational Modeling of Electric Power and Natural Gas Markets*, Computational and Mathematical Organization Theory Journal, Kluwer Academic Publishers, Norwell, MA.

Daniels, Marcus G. (2000). *Writing fast models in Swarm*. SwarmFest 2000. http://www.santafe.edu/~mgd/swarmfest2000/performance.html

Wilensky, U. (2001) *Modeling Nature's Emergent Patterns with Multi-agent Languages.* Proceedings of EuroLogo 2001. Linz, Austria.

Dugdale, Julie. *An evaluation of Seven Software Simulation Tools for Use in the Social Sciences*. COSI Project Technical Report.

Swarmwww. http://www.swarm.org/

Swarm Online Doc. http://www.santafe.edu/projects/swarm/swarmdocs/set/set.html

RePastwww. http://repast.sourceforge.net/

RePast HowTos. http://repast.sourceforge.net/docs/how_to/how_to.html

MAMLwww. http://www.oasis-open.org/cover/maml.html

MAML Tutorial. http://www.maml.hu/maml/tutorial/index.html

SDMLwww. http://sdml.cfpm.org/

Ascapewww. http://www.brook.edu/dybdocroot/es/dynamics/models/ascape/

AgentSheetswww. http://agentsheets.com/

StarLogowww. http://el.www.media.mit.edu/groups/el/Projects/starlogo/

StarLogoTwww. http://ccl.northwestern.edu/cm/starlogoT/

NetLogowww. http://ccl.northwestern.edu/netlogo/

NetLogo Manual. http://ccl.northwestern.edu/netlogo/docs/

HDF5www. http://hdf.ncsa.uiuc.edu/HDF5/

GNU Rwww. http://www.r-project.org/

SDML Tutorial. http://sdml.cfpm.org/intro/html/

ANNEX

The following tables compile the resources we have made use of in order to elaborate this survey.

Multi-agent simulation software I

name	Description	language	model-design gui	news lists	examples	Tutorial	manual	pros	cons	review				platforms				latest version	src	license
										sss	us	irit	geneura	Linux	other UNIX	MS Win	Mac			
swarm	libraries	objective C, Java	-	x	x	x	user guide	very extensive, highly tested	difficult to learn	x	x	difficult to learn; powerful, flexible	fastest; difficult for beginners	x	irix, solaris, hp/ux	9x, NT, 2k	-	2.1.1	x	GNU GPL
Evo	framework for swarm	objective C		x	x			genetic algorithms		-	-	-	(x)	x	solaris	9x, NT, 2k	-	1.0.1		GNU GPL
MAML	"macro language" for swarm	objective C	alpha version	-	gen, sci	x	user guide, reference, technical	vrml		-	-	objective C needed; no mail list	(x)	x	-	NT	-		x	
SDML	modelling language	Visual Works Smalltalk	x	x		x	.hlp		1999?	x	x	few examples; communication via DB	-	x	adux, aix, hpux, irix, solaris	3.1, 9x, nt, 2k	Power Mac	4.1		GNU GPL
RePast	"java port of swarm"	Java		x	x		api doc, diag, howto, faq	built-in tools, movies		-	-	few examples	less complete than ascape					1.4 (24.1.02)		BSD

Multi-agent simulation software II

name	description	language	model-design gui	news lists	examples	tutorial	manual	pros	cons	review				platforms				latest version	src	license
										sss	us	irit	geneura	Linux	other UNIX	MSWin	Mac			
Ascape	new version of Sugarscape	Java	exploring	x			api	web, movies		-	-	end-user tools; little user doc	simplest; more complete (gui)	x	solaris, unix	x	MacOS	1.9.1 (12.10.00)		w/o fee for non-comm
Agentsheets	generation of Java applets		x		x	movies	manual, faq			-	-	spreadsheet approach; very simple use; simple simulations; teaching; poor agent interaction; applets generation	-	-	-	x	x	1.4/2.1		comm
Starlogo *	logo-like turtles	commands (Java)	x	x	x	x, x	getting started, commands	easy to use, applets generation	inflexible, slow	-	-	no OO; applet generation; easy to use; lots of examples; inflexibility; not for complex models	-	x	solaris	9x, nt, 2k, me, xp	x	1.2.2 (10.8.01)		
StarlogoT	logo-like turtles				lots				only Mac	-	-	-	-	-	-	-	PPC	2001 R2		
NetLogo	logo-like turtles	Java	x	x	lots	x	user			-	-	-	-	(java)				1.0 (1.4.02)		