

# JAS: Java Agent-based Simulation library.

## An open framework for algorithm-intensive simulations.

Michele Sonnessa<sup>1</sup>  
sonnessa@di.unito.it

### Abstract

This paper shows how agent-based modelling technique is a suitable approach for social scientists to model complex adaptive systems, using computer as experimental environment. Unfortunately advanced tools are lacking as well as an unified language supporting its development. We present JAS, a new agent-based simulation tool, developed with the aim to improve AB models designing. We give a brief methodological introduction to its use and a short description of its architecture.

**Keywords:** complex adaptive systems, agent-based simulation, open source software

### Introduction

Since the birth of the quantum physics, the science of the twentieth century has been characterized by the explorations of alternative modelling languages.

Ilya Prigogine (Prigogine 1997) suggested the use of probability within the traditional mathematical models :

In recent years, a radical changes of perspective has been witnessed in science following the realization that large classes of systems may exhibit abrupt transitions, a multiplicity of states, coherent structures or a seemingly erratic motion characterized by unpredictability often referred to as *deterministic chaos*.

[...]

Non-linearity and instability force us to adopt a different point of view concerning the formulation of the laws of nature. They now express possibilities instead of certainties.

The successes in the studies of the fractal mathematics (Mandelbrot 1975) is the basis for one of these new alternative perspectives: the experimental mathematics, in which the computer becomes the favourite instrument of research (Barrow 92). Particularly in the field of complexity, the massive presence of non-linear phenomena, the adaptive topological interactions and the evolving behaviours of system elements, makes the formal mathematics harder and harder to be managed.

It is well known (Gilbert *et al.* 2000) as scientific modelling could be carried out using different kind of languages and their effectiveness deeply depends on the choice of the most appropriate one.

A complex adaptive system (CAS) may be defined as an open system, made of several elements which interact, driven by non-linear behaviours, forming a unique organized and dynamic entity, able to evolve and adapt to the environment (Gandolfi 1999).

---

<sup>1</sup>Department of Computer Science, University of Torino.

Such kind of systems seems to be well studied with the ABM formalism. They are modelled as a collection of rules and algorithms, which can be numerically simulated. The attention is concentrated on the single agents' behaviour. It is represented in terms of reactive system, often determined by IF..THEN..ELSE conditions, typical computer language construction. So, with no doubt, the computer language becomes an alternative way to design dynamic systems.

ABMs and classical dynamic linear models are also different from a purpose point of view. Scientific modelling can be carried out with two different perspectives: the direct and inverse problem (Tikhonov *et al.* 1987). The attempt to reproduce empirical phenomena, starting from a given model, is considered a direct problem, while the observation of the empirical data in order to infer the cause-effect laws, which are subsequently used to forecast the future behaviour of the system, is called inverse problem. The ABMs are mostly used to represent CAS with the direct problem perspective, differently from linear dynamic modelling which is used to "compress" knowledge about a system in equation based laws.

Agent-based simulation models are experiments of minds, with the aim of understanding the internal adaptive processes of a system (Lavoie *et al.* 1990). The forecast of its future dynamics is not a real objective for this models, also because in presence of chaotic laws the long term forecasts are mathematically uncertain.

## Swarm

Building a computer simulation model requires a good skill in computer programming. In addition, the code plays the role of the modelling language and it must be also used for model validation, because revolutionary outcomes might be caused by code bugs rather than by emerging properties of the model.

The programming languages are too generic to describe a scientific model and they are full of technicalities due to computers' architecture<sup>2</sup>, so it is necessary to define a language subset and a description formalism with a specific semantic for agent-based models. In other words, we need a common language for ABM.

The most known toolkit for building ABM simulation is Swarm<sup>3</sup> (Minar *et al.* 1996), developed at the Santa Fe Institute<sup>4</sup>.

Through the experiences of the Swarm user community, we have today a clearer idea of the potentialities and the limits of agent-based simulation models.

The advantages of this methodology have been shown in different subjects like the game theory (the Prisoner's Dilemma, Axelrod 1984), the biology (Kauffman 1993), the epidemiology (Bagni *et al.* 2002), the financial applications (Terna 2000 and LeBaron 1996).

The Swarm Development Group (SDG) gave the community three main contributions.

*A simplified approach to the ABM development.*

Simulations always need a piece of software driving the time evolution, random number generators, statistical tools, plotters. They are common to every simulation model. A library of tools simplifies its designing, reduces the programming time and improves the code reliability.

---

<sup>2</sup> For instance, let's think about the complex notation used to manage reference pointers to the memory...

<sup>3</sup> <http://www.swarm.org>

<sup>4</sup> <http://www.santafe.edu>

*A definition of a schema in model designing.*

Much important is a definition of a methodology for writing models. The SDG suggested, for instance, to keep a strong separation between the model (a piece of software which simulates the system) and the observer (a set of routines looking into the model, collecting statistics and showing them to the user). This approach is very much elegant and it has a similitude with the real world, where things happen and the researchers look at them from the outside, without participating to the system evolution.

The separation between model and observer may have also a relation with the ontological distinction between the *design complexity* and *control complexity* made by John Casti (1986). The *design complexity* expresses the idea of complexity perceived by the observer of a system, while the *control complexity* is complexity of the observer perceived by the system itself. In other words, the complexity may be not an absolute property of the system, but something emerging by the interaction between observer and observed.

*The creation of a community of users.*

The aggregation of many people using Swarm, through mailing lists and web site (collecting their works), represents a sort of *agora* for the “ABM people” and gives an important contribute to the community.

Many useful features were not implemented in Swarm although, thanks to the open source world, powerful libraries may be included into the model implementation. However, we think this is an unpromising approach, because it increases the difficulty of writing and sharing AB models. In fact, Swarm is not only a simulation library but also a methodology, a reference framework to build models in such a way that anyone, knowing its interface, can easily understand the source code and check any model detail.

In learning to use a tool, the community of modellers also learns to “speak the same language”. On the contrary, when a modeller imports an external tool, he or she introduces a new dialect, reducing the communication possibilities.

## **JAS<sup>5</sup>**

JAS, Java Agent-based Simulation library is:

- an environment for simulation experiments,
- a framework for building agent based models,
- a Java library, rich of simulation-oriented tools.

It is an open source project, hosted by the SourceForge open source portal and consists of a collection of Java utilities composing a framework for building agent based simulation models.

The library has been developed adopting the same Swarm philosophy of model-observer frame, because with no doubts it has become a standard.

In order to let models to be further standardised, it is necessary to enrich the basic tool with further features. The use of external libraries is welcome, but they have to be homogenized with the features of the tool. JAS includes well tested and standard libraries<sup>6</sup>, but they appear as a part of JAS through some specific wrapper classes. For instance,

---

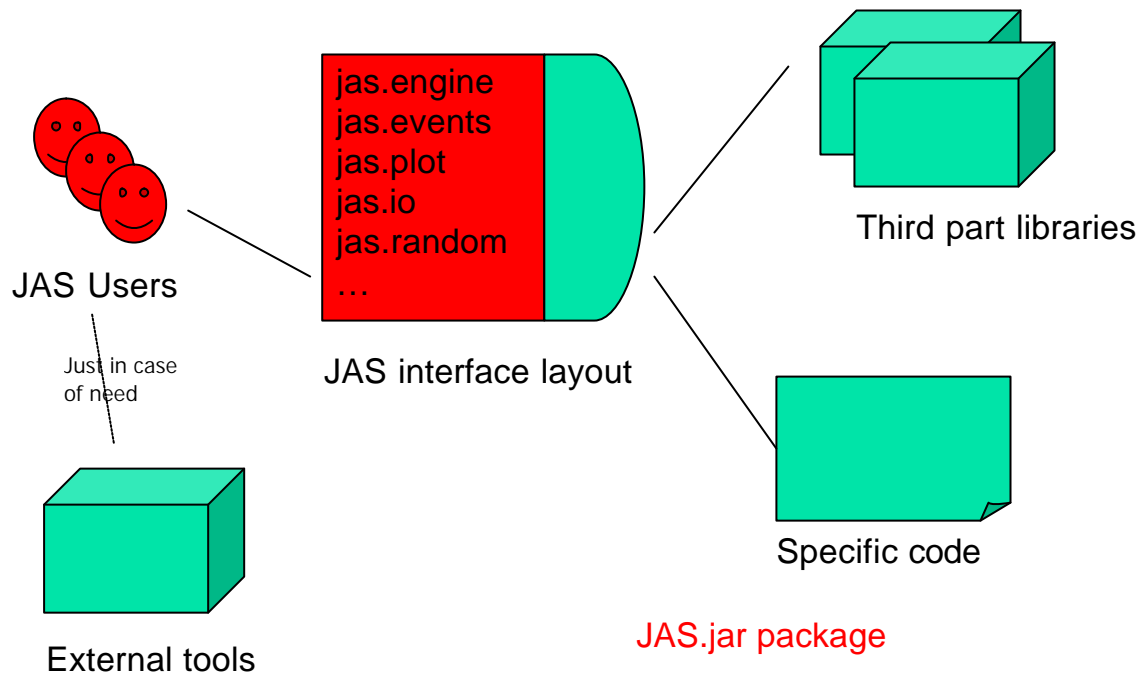
<sup>5</sup> <http://jaslibrary.sourceforge.net>

<sup>6</sup> See JAS web site or JAS about box for a complete list of the libraries included in the package.

we used the ptPlot<sup>7</sup> library as standard plotter, but its rich and complex interface has been filtered and now it is able to natively plot the statistical probes contained in the *jas.stats* package.

This way, the final user is not required to know their implementation, this remaining a problem of the developers.

We think this is a fruitful way to work with open source code, that facilitates continuous improvements of the tool.



**Fig. 1 The architecture of JAS**

The following list represents the main new features of JAS in comparison with Swarm.

- A pure Java code implementation, so it is easy to be installed and configured. No operating system dependent libraries have been used.
- The possibility to execute in parallel agents' actions.
- A network protocol (Sim2Web<sup>8</sup>) to publish simulations on the web and to allow humans interactions with the simulation through a web page.
- A reduced set of turtle's movement instructions taken from Starlogo<sup>9</sup> (Resnick 1994).
- XML file format support.
- A powerful random number generator and statistical functions taken from COLT library<sup>10</sup>.
- AI library supporting GA, ANN and CS to implement agent's intelligence.
- A discrete event time management, with real time emulation and different time unit representation.
- A dynamic class loader, to reduce the problems in configuring CLASSPATH variable for models execution.

<sup>7</sup> <http://ptolemy.eecs.berkeley.edu/java/ptplot>

<sup>8</sup> <http://wf.econ.unito.it/sim2web>

<sup>9</sup> <http://education.mit.edu/starlogo>

<sup>10</sup> <http://hoschek.home.cern.ch/hoschek/colt>

- A multi-run protocol to perform automatic parameter calibration.

JAS is not only a library but an application. After the installation procedure, in fact, it can be started like every other Java application. This could seem to be hazy.

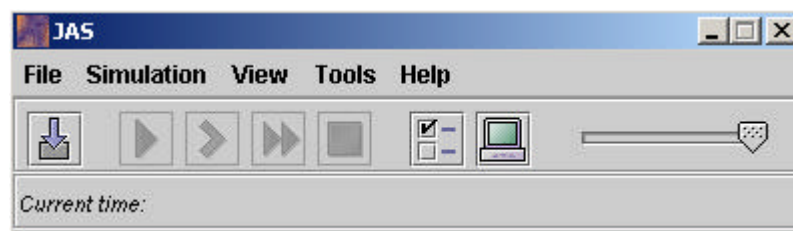
A JAS model is just a Java application mostly based on the classes defined in the JAS.jar package, compiled with a standard Java compiler (JDK, for instance). So the reader could imagine that in order to execute a model it is necessary to start it like every other Java application.

JAS is equipped with a custom Java class loader, which can load compiled class “on-the-fly”, without a previous CLASSPATH specification. Thanks to it, we can run JAS as an application and start simulation models as they were simple documents of the application (with the File/Open menu command). Moreover, it is possible to stop a simulation and restart it without shutting down JAS.

Thanks to this feature, we were able to define a multi execution protocol, useful to automate the parameter calibration.

The *Control Panel* (Fig. 2) is the main JAS window, through which is possible to:

- create a model project specifying the paths, libraries and parameters for the simulation engine, thanks to the project editor;
- load and run a compiled model (through the XML project file);
- edit seed random number, windows position and so on;
- open a console output window (useful when JAS is started with the javaw command)
- control the simulation engine status (event list, models currently running, windows, ..).



**Fig. 2 JAS Control Panel**

## Why Java

It has been asserted above that the choice of modelling language is crucial. At the same time the choice of the programming language determines the characteristics of the simulation tool.

Although programming languages are only formal collections of rules to express the same concept: the algorithms, which are always translated by compiler in the same machine code, they implies different styles of expression.

JAS is written in Java language, because we want it to be open source and platform independent. Java is with no doubt the most diffused and appreciated multi-platform language. The open source java community offers a very large set of ready-to-use libraries. It is suited for scientific applications since its arithmetic is based on 64-bit words, the best numeric precision nowadays available (Horstmann *et al.* 2001).

In addition, Java has two interesting features: the executable are never linked and it natively supports reflection.

Traditional compilers link each part of the compilation process into a unique executable file. This mechanism creates a shielded code which can be accessed only by the executable itself. On the contrary, every Java application is not linked by compilers (jar files are simple extractable archives). This means that each package may be at the

same time a library of functions and an executable program. Thanks to this feature JAS is both a library to write AB models and an application able to control simulation run.

The reflection mechanism is a technical, but very relevant, issue. It allows to take off “the lid” of object instances and look inside them. Doing it with other languages requires to access directly the computer memory<sup>11</sup> with the high risk to cause a system crash. Java has native methods to do it in a safe and easy way. The implementation of the probes has been much easier in JAS rather than in Swarm.

These advantages notwithstanding, Java has an unpleasant limit. It is slower than the native languages. This is a problem when code execution has to be fast, as it is often the case with simulations. With this concern in mind, we took care of using the fastest algorithms in looped actions execution, like scheduled operations, and particular attention was paid to memory management, developing customized garbage collectors<sup>12</sup>. Swarm is still faster, especially in graphics, but the difference is acceptable considering Java pros.

## 2 Simulation Environment

This paragraph shows the skeleton of any simulation model developed with JAS.

A JAS simulation model is a Java application with some specific constraints. A JAS-compliant simulation model must have a main class, inheriting from the *jas.engine.SimModel* template. The *SimModel* class represents a bridge between the user’s model and the simulation engine.

The *SimModel* interface provides a reference to the JAS’ centralized timer, the *eventList* variable, and a set of standard methods which are invoked by the JAS when particular events happen.

The *SimModel* class is abstract and force the model designer to override two methods:

- *setParameters()*
- *buildModel()*

and optionally *simulationEnd()*.

The first method is called by JAS just when the model is loaded into memory. At this time the model set (collection of agents and widgets) has not been instanced yet and the user can still modify the parameters of the simulation experiment. The *setParameters()* method allows the modeller both to set directly some constant parameters, loading them from a file, and to open a probe (or other kind of visual interface) to let the choice to the user.

Only when the user presses the “Build model” button, JAS calls the *buildModel()* method, which plays a key role: it represents the point when the simulation model is finally set into memory.

Usually the *buildModel()* method performs two tasks:

- creates instances of the agents and all the model stuff ;
- defines the event structure: which events will happen and when.

This implementation is left to the model designer.

If, for instance, at the end of the simulation data are to be written into an output file, the modeller might also override the *simulationEnd()* method, defining which operations are to be executed when an *EVENT\_SIMULATION\_END* event is raised or when the user clicks the “Simulation\End” menu item.

---

<sup>11</sup> The memory zone where compiled code is loaded. Accessing the code segment is a dangerous operation.

<sup>12</sup> The garbage collector is a software component which stores objects that are no more used, in order to recycle them when a new object of the same type is needed by the application.

In the Swarm protocol two special objects are defined with the aim of coordinating the experiment: the *ModelSwarm* and the *ObserverSwarm* classes. JAS generalizes this structure introducing only the *SimModel* interface as controller. So, an observer and any other kind of model controller must always override the *SimModel* class.

The reason for this choice is due to the fact that, from a technical point of view, an observer is not different from a model: it creates some objects (typically statistical and graphical widgets) and schedules their updating frequencies. In Swarm they are nested. So when the observer is started, it creates an instance of the model it wants to observe. In JAS, observer and model are executed independently (we can run in parallel as many model as we want) and the observer is put in the condition to look into the model retrieving its reference from the JAS engine. In other words, an observer class looking at the model is implemented simply like another model, running in parallel and observing its state variables.

One of the advantages of this architecture consists in the possibility to run a model silently, simply deactivating the observer in the project definition file.

Through an XML project file, in fact, we can define a simulation experiment, specifying one or more models which are executed all together (the swarm of swarms). The action sequence of each sub-model is coordinated by the JAS engine, with a unique timer.

The following XML code is used to start an example model. It has been visually designed with the JAS project editor.

```
<?xml version="1.0" encoding="UTF-8" ?>
<JAS projectName="SimpleBug">
  <ProjectParameters>
    <TimeUnit>7</TimeUnit>
    <MajorVersion>0</MajorVersion>
    <Seed randomSeed="true">1024304619192</Seed>
    <ProjectDescription>The simplest agent based example.
  </ProjectDescription>
  </ProjectParameters>
  <Model className="SimpleBugModel">
    <Window title="Simple bug model">9,128,414,403</Window>
  </Model>
  <Model className="SimpleBugObserver">
    <Window title="Space viewer">432,8,400,400</Window>
  </Model>
  <ClassPath>
    <Path>.\examples\SimpleBug</Path>
  </ClassPath>
</JAS>
```

It is interesting to notice that if the *SimpleBugObserver* element (<Model> XML tag) is present in the XML document tree, the observer will be executed, otherwise the model will run in a non-graphical mode.

## The package overview

Java allows the programmer to organize his or her code into packages. The package system is primarily used to avoid namespace conflicts so that two Java classes with the same name will not be confused.

JAS consists of 11 packages, which are shortly described below.

### *jas.ai*

The artificial intelligence package contains a simulation oriented implementation for the main evolving algorithms: artificial neural networks are based on the original implementation of the *bp-ct* package by Pietro Terna; genetic algorithms and classifier systems are based on the work by Gianluigi Ferraris<sup>13</sup>.

---

<sup>13</sup> <http://www.swarm.org/community-contrib.html>

### *jas.engine*

The engine classes are responsible for setting up and driving the simulation event-based engine. The simulation engine (SimEngine) is a typical discrete-event engine, particularly optimized to speed up the execution of looped schedules. In fact, ABM are often executed with a repeated sequence of events.

The SimModel is the template class for all models managed by JAS. The ControlPanel is responsible for handling user interaction with a simulation through a GUI. The engine package contains a custom java class loader, which is able to dynamically manage the java CLASSPATH.

### *jas.events*

The events are the core of the scheduling and message dispatching mechanism. The events package contains a rich collection of event, which might be scheduled for single agents (unicast), for entire collections (broadcast) and for restricted group (multicast) of them. In addition they are defined particular system events, understood by the JAS SimEngine.

It is implemented a garbage collector for reducing memory occupancy and the EventList and RealtTimeEventList are the threads managed by the SimEngine to schedule events and raise them in time. In particular, the real time implementation of the event list is able to fire event using a real timer for emulation purposes.

### *jas.io*

The input/output operations are very important for a simulation model. They are responsible for the input/output communications with the user and for reading from disk and writing to it.

The io package supports the comma separated values (CSV), the Microsoft Excel spreadsheet and the eXtensible Markup Language (XML) formats for disk I/O operations.

### *jas.net*

The net package is the repository for each network service of JAS. Particularly, all the interfaces for the XML-RPC communications are contained by this package. They have been used to realise the Sim2Web architecture, which allows to publish simulation models on the web, via Zope web server. In addition, they might receive commands from the remote users. See the website <http://wf.econ.unito.it/sim2web> for more details.

### *jas.plot*

All the graphic widgets are available in the plot package. There is a raster class, which is able to draw bi-dimensional overlapped surfaces. The TimePlot and the BarPlot are the standard plotters of JAS. They are based on the ptPlot library from the Berkely University.

### *jas.probe*

The probes are particular graphic objects, which inspects an object instance and shows the user current values for its state variables, allowing him or her to change some values or to invoke object methods.

The probes are very important to let the observer to watch inside the simulation model, checking the internal dynamic of a particular element of the system.

The probe is based on the java reflection library.

### *jas.random*

The pseudo random number generation is an essential tool for computer simulations. It represents a critical activity, because the algorithm generating pseudo random number could be affected by loops, when the generation algorithm is not complex enough. At the same time, this operation requires a lot of CPU resources. It is a difficult compromise and this is the reason why we used a well known and hi-level speed library: the COLT library, developed at CERN, Geneva. It is probably the best Java implementation for random and statistical functions.

### *jas.space*

The space package contain 2D grid surfaces for mapping topological spaces, particularly useful in model based on cellular automata. They are logical representation of space, while their graphical rendering is left to the classes of the *jas.plot* package.

### *jas.stats*

A simulation model does not give only graphic results, but many numerical series, which represent the basis of the latter statistical analysis. They are useful to understand both the overall behaviour of the system and of the single agents.

The stats package contains probes able to inspect agents' state and to collect statistics. While data are collected a rich set of statistical indicators is available. They are able to write data to disk, too.

They are based on the same COLT library used in the random package.

## **Conclusions**

Many researchers are upset when they discover that the only way to build an agent based model is to learn a programming language. Their first idea is to search for a visual application that allows them to build models in a couple of minutes. The reason for the absence of such applications or environments in the world of ABM is due to the difficulty in defining reusable building blocks for these kind of models. In fact, they are distributed, parallel and their relations are loosely coupled. Rarely they present a stable network of relations.

We said that an ABM is an algorithm-intensive formalism, because the classes representing agents have to drive their behaviour, using a rich set of algorithms. They are characterized by a list of things to do, with many conditions testing the external inputs in order to decide the right action to be taken.

These models are rich in behaviour routines and the visual editing of such complex algorithms may result in a very complicated graphical representation.

A proof of this comes observing the structure of available ABM tools: Starlogo uses its own language; Ascape, Repast, JAS are based on Java; Swarm is based on ObjectiveC.

A probably good way to resolve this impasse is the definition of a specific subset of instruction, a specific semantic and use the UML formalism to represent such kind of models. The model-observer paradigm introduced by Swarm authors could be a starting point.

In comparison with Swarm, JAS has some improvements and it is easier to use. The modeller has still to type code, but we tried to hide as many technicalities as we could. This was possible thanks to Java. The Swarm original source code was written in ObjectiveC, and this means that implementation of the probes and the Selector class has been very hard. Using the Java Reflection and other Java features, JAS code is much lighter and requires the user to write down fewer technical instructions than Swarm.

## References

- Axelrod R. (1984), *The Evolution of Cooperation*, Basic Books, New York.
- Bagni R., Berchi R., Cariello P. (2002), *A Comparison of Simulation Models Applied to Epidemics*, Journal of Artificial Societies and Social Simulation vol. 5, no. 3
- Barrow J. D. (1992), *Pit in the Sky. Counting, Thinking and Being*, Oxford University Press, London
- Beltratti, A., Margarita S., Terna P. (1996), *Neural Networks for Economic and Financial Modelling*, ITCP, London.
- Casti J. L., Karlqvist A. (1986), *Complexity, Language and Life*, Mathematical approaches, Springer, Berlin pp.146-73
- Gandolfi A. (1999), *Formicai, Imperi, Cervelli*, Bollati Boringhieri, Torino.
- Gilbert N., Terna P. (2000), *How to Build and Use Agent-based Models in Social Science*, Mind & Society 1
- Horstmann C., Cornell G. (2001), *Java 2*, vol. 1 & 2, Sun Microsystems press.
- Kauffman S. A. (1993), *The Origins of Order: Self-Organization and Selection in Evolution*, Oxford University Press, New York.
- Lavoie D., Baetier H., Tulloh W. (1990), *High-tech Hayekians : some possible research topics in the economics of computation*, Market Process, 8, Spring, pp.120-147
- LeBaron B. (1996), *Asset Pricing Under Endogenous Expectation in an Artificial Stock Market*, Santa Fe Institute Working Paper 96-12-093
- Mandelbrot B. (1975), *Les objets fractals*, Flammarion, Paris.
- Minar N., Burkhart R., Langton C. , Askenazi M. (1996), *The Swarm Simulation System: A Toolkit for Building Multi-agent Simulations*, Santa Fe Institute Working Paper 96-06-042.
- Prigogine I. (1997), *Non-linear Sciences and the Laws of Nature*, Journal of The Franklin Institute Volume: 334, Issue: 5-6, September 11, 1997, pp. 745-758
- Resnick M.(1994), *Turtles, Termites and Traffic Jams: Explorations in Massively Parallel Microworlds*, MIT Press.
- Terna P. (2000), *SUM: a Surprising (Un)realistic Market: Building a Simple Stock Market Structure with Swarm*, presented at CEF 2000, Barcelona, June 5-8.
- Tikhonov A.N., Goncharsky A.V. (1987), *Ill-Posed Problems in the Natural Sciences*, Mir, Moskva