FOUNDATIONS OF COMPUTING AND DECISION SCIENCES
Vol. 38 (2013) No. 4

DOI: 10.2478/fcds-2013-0014 ISSN 0867-6356
e-ISSN 2300-3405

CONTROLLLING SIMULATION EXPERIMENT DESIGN FOR AGENT-BASED MODELS USING TREE REPRESENTATION OF PARAMETER SPACE

Przemysław SZUFEL*, Bogumił KAMIŃSKI*, Piotr WOJEWNIK*

Abstract. An important aspect of the simulation modelling process is sensitivity analysis. In this process, agent-based simulations often require analysis of structurally different parameter specifications – the parameters can be represented as objects and the object-oriented simulation configuration leads to nesting of simulation parameters. The nested parameters are naturally represented as a tree rather than a flat structure. The standard tools supporting multi-agent simulations only allow only the representation of the parameter space as a Cartesian product of possible parameter values. Consequently, their application for the required tree representation is limited. In this paper an approach to tree parameter space representation is introduced with an XML-based language. Furthermore, we propose a set of tools that allows one to manage parameterization of the simulation experiment independently of the simulation model.

Keywords: agents, simulation modelling, sensitivity analysis.

1. Introduction

The paper deals with the problem of agent-based modeling and simulation (ABMS). ABMS[1] is defined an approach to modeling complex systems composed of autonomous interacting agents [12]. It is assumed that the behavior of agents is described by a set of rules specifying the interactions between an agent and its environment or other agents.

In real life, the analysis of complex situations and the design of simulation model should be a structured process [7]; [10]. In particular, the authors agree that it should include at least the following steps: model validation, verification and sensitivity analysis.

* Division of Decision Analysis and Support, Warsaw School of Economics, ul. Madalińskiego 6/8, 02-513 Warszawa, Poland

[1] Macal and North [12] note that in the simulation literature two names for simulation of complex systems with agents exist: agent-based modeling (ABM) and agent-based simulation (ABS). In order to avoid ambiguity they propose a name agent-based modeling and simulation (ABMS) [11].

These three steps, and the sensitivity analysis in particular, require evaluation of a model with many different parameter sets. In a multi-agent simulation there is often a need to analyze model sensitivity with respect to structural changes in its source code. A typical structural change in such a case could be evaluating the influence of various agents' decision making algorithms on simulation outcomes. The structural sensitivity analysis problem becomes particularly important in simulation models having complex parameterization. During the last ten years the complexity of simulation models has rapidly increased and this calls for the development of new tools and architectures [21].

The simplest approach to simulation model parameterization and parameter space sweep is introduction of several nested *for*-type loops within source code that executes the simulation model for various parameter sets. This solution does not allow the separation of the model source code from the runtime environment and is difficult to manage when a parameter set dynamically changes during a simulation process. The parameter sweep through source code becomes even more complicated when someone other than the model's author wants to change the parameter space – she might even not have the access to model's source code. The source code problems have been noted already at early stages of simulation methodology development. Ziegler [23] introduced the concept of *experimental frame*, arising from the need to separate simulation source code from its parameterization. Daum and Sargent [5] further developed this idea, where authors pointed out the need to manage a model structure from a configuration layer. They also stress out the need to provide graphical tools for simulation process management – *Visual Interactive Modelling* (VIM).

Railsback et al. [16] present an overview of papers on agent-based modeling tools. Among other things they stress the role of simulation experiment design. Several popular multi-agent simulation environments have functionalities supporting parameter sweep – e.g. NetLogo BehaviourSpace [13], Repast Batch Parameters [4]. Another type of simulation scenario management is using some external environment (and programming language) to launch the simulation process and to manage its parameters. An example is RNetLogo, a tool which can be used to start a NetLogo multi-agent simulation and manage its parameters from GNU R programming language [18].

Each of the above presented approaches enables the consecutive launching of simulation models with different parameter values. In BehaviourSpace those parameters are presented as global values within a simulation model, while in Repast Batch Parameters a configuration is presented as a JavaBean object in an XML file. Finally, running a simulation from RNetLogo requires usage of GNU R language to write source code responsible for changing the parameter values.

Another approach is JABM library [15]. The author proposed a new multi-agent framework where the simulation configuration can be obtained from the Spring framework similarly to the approach proposed in this work. Our paper extends the JABM library in several areas: firstly, by providing configurable tools that can be used with various simulation platforms; secondly, by supporting parameter sweep and thirdly, by taking into consideration a tree parameter structure.

However, using any of the above tools requires choice of the variables to be parameterized *ex ante*. Moreover, these tools do not offer full control over the simulation process and do not provide any simple mechanism to repeat simulation experiments. Another issue is integration of a formal simulation experiment description with parameter

sweep management. For example Waltemath et al. [20] propose a simulation experiment scenario description, but do not consider controlling of parameter space.

A specific feature of multi-agent simulation is the need to consider the elements of the model as independent agents. Usually this is achieved through implementation in object oriented language and representation of agents as objects of appropriate classes. The approach considered in this paper also allows model parameters to be represented by objects. The objects can represent complex model features like decision rules used by a particular agent classes. Each of these complex configuration objects can be further internally parameterized, leading to *parameter nesting* that can represented as a *tree parameter space structure* (see Section 2.3). However, currently all leading agent-based modeling frameworks operate only on parameter spaces based on a Cartesian product of possible parameter values. When a parameter nesting occurs, the Cartesian product approach leads to increased computational complexity of the simulation model.

In this paper an approach to tree representation of nested parameter space is proposed together with a set of tools to process this representation independently of a simulation model. In Section 2, a review of existing approaches to simulation model parameter sweep is presented. Based on this review, in Section 3 a new approach to solving the problem is proposed and an illustrative example is described in Section 4. Discussion of the results is given in Section 5.

2. Management of parameter space in simulation models

In the literature authors agree that each simulation model requires verification and validation [10]; [7]. The goal of *verification* is to check whether a computational model behaves along the mathematical expectations – i.e. to verify whether the model has been programmed according to formal assumptions. The goal of validation is to check whether a simulation model behaves in similar fashion to a real system. An important tool supporting the verification and validation of a simulation model is sensitivity analysis. In this analysis it is observed how the model parameters change influences the relation between model inputs and outputs. The sensitivity analysis requires the running of a simulation model several times (often thousands of times) – with different parameter sets.

To evaluate existing frameworks and approaches to multi-agent simulation we have reviewed existing multi-agent software platforms. The initial list of software programs (almost 100 platforms) is based on review papers [14]; [19] and internet sources [22].

For the selection we have used the following criteria:

1. platforms should be general purpose (not focused on a single application domain) and provide a full simulation framework;
2. they have to be freely available, still being actively developed (we exclude frameworks that have not had updates since 2010) and be available for production use (not claim to be in alpha or beta development stage);
3. as we focus on *agent based modeling and simulation* in our analysis we do not consider frameworks aimed at modeling software agents[2].

The following[3] frameworks meet all the above criteria: Repast, NetLogo, Mason, and Swarm. Our choice of agent-based simulation software is consistent with the literature. The ABMS software review papers agree that the above four frameworks are the most important software packages for agent-based modeling and simulation [[1]; [3]; [8]; [12]; [16]]. Therefore, four software packages will be used in the paper as benchmark agent-based simulation platforms, i.e. NetLogo, Repast and Mason and Swarm. The first three are written in Java while the last was created in Objective-C.

Two popular approaches to management of simulation model parameter space can be considered: (1) specialized tools such as NetLogo BehaviorSpace or Repast Parameter Sweep and (2) writing own code to control the model parameter sweep process, possibly through batch or libraries provided with simulation library (e.g. Swarm Perl libraries). In this Section, the selected tools and approaches for both parameter management scenarios will be presented.

2.1. Tools for parameter sweep management in multi-agent simulation

Configuration of multi-agent simulation process in NetLogo (BehaviorSpace toolset) and Repast (Repast Parameter Sweep toolset) can be done simply through a graphic user interface (GUI). Moreover, both toolsets allow the replication of simulation results and the management of multi-threaded simulation. However, they both have significant drawbacks. Firstly, parameters are limited only to numbers; secondly, a parameter sweep is limited to the Cartesian product of possible values – no tree parameter structures are supported; thirdly, they do not provide full separation between a model and its configuration.

2.2. Programming parameter space management

The second approach to model configuration management is based on developing code specific to a particular simulation. This can be achieved in several ways:

- parameter sweep within source code;
- parameter sweep through batch processing;
- parameter files;
- mixes of the above.

[2] This is the reason why we do not consider frameworks such as JADE [2].

[3] In particular we exclude from the list: Janus and MadKit as these are more communication-oriented multi-agent frameworks and Ascape as it is only rule-oriented simulation engine.

The fastest way to implement the parameter sweep is to embed it into *source code*. Usually it takes the form of several nested loops. The main advantage of the source code approach is fast implementation and fast code execution. However, this type of parameterization requires the integration of the source code for a simulation model with the source code responsible for parameter sweep. This leads to a situation where any parameter space change requires changes in the model's source code. Such changes are error prone, requiring programming skills and in-depth knowledge of source code – which might turn out to be difficult when a different person implements a simulation model and a different person executes it.

Batch processing management of a simulation model is performed by running the model with different parameter sets, where configuration of each run is external to the model's source code (e.g. a separate script file is created). A batch executing model with different parameter sets can be implemented in scripting and shell languages or using tools constructed for simulation processing, such as RNetLogo. The main advantage of batch processing is its flexibility – it can be applied to any parameter set and any combination of parameter values. This approach is also present in the Swarm simulation framework where tools are provided to create batch scripts in Perl programming language. However, passing values as parameters in a batch file can have several disadvantages: it is error prone, requires some form of communication between a command shell and a simulation model, and might require more computing power (in case of starting a new process for each parameter set). The main disadvantage of the batch approach is that changes in model parameter space are usually complicated and hard to control.

Managing the simulation process through *parameter files* is achieved by introduction of an external flat file containing a set of keys and values. Consequently, the model parameterization is strictly separated from model implementation. However, this approach does not take into consideration complex data structures and makes it is difficult to include additional dimensions (such as time) in a configuration file. Moreover parameter files do not provide any tools that would allow validation of model specification.

The *mixed* approach is based on combining the former techniques in managing simulation with parts of a model written in dynamic programming languages such as Groovy or Jython (assuming that the simulation model is written in Java). The mixed approach is used in simulation models constructed with the MASON framework. The main advantage of this approach is its flexibility in constructing the various simulation parameter sweep scenarios. While its main disadvantage is increasing complexity of the source code and its interdependence.

Having presented various approaches to parameterization of simulation models we them compare in Section 2.4. However, prior to this comparison we describe the need for tree representation of parameter space in multi-agent simulations.

2.3. Tree representation of parameter space

Assume that an object `simC` of class `SimC` represents a configuration of a multi-agent simulation model. Let the class `SimC` have one numerical attribute `param1` and one object attribute `logic`. The numerical attribute `param1` can have one of two values: `1` or `2`. The object attribute `logic` of an abstract class `Logic` represents the agent's decision rule

algorithm. Assume that two possible logics are considered – object `logicA` of class `LogicA` and `logicB` of class `LogicB`, where the classes `LogicA` and `LogicB` are subclasses of `Logic`.

When an agent uses logic of class `LogicA` the model has to be further parameterized with parameters `paramA1` and `paramA2` (the parameters are actually fields of class `LogicA`). When an agent uses logic of class `LogicB` the model is further parameterized with parameter `paramB` (the parameter is a field of class `LogicB`). Possible values for `paramA1` are 3 and 4, possible vales for `paramA2` are 5 and 6 and finally possible values for `paramB` are 7 and 8. Thus, the model parameter structure can be represented as the following hierarchy:

- `SimC:`
  - param1 – `{1, 2};`
  - Logic – `{logicA, logicB};`
- `LogicA:`
  - `paramA1 – {3, 4};`
  - `paramA2 – {5, 6};`
- `LogicB:`
  - `paramB – {7, 8}.`

The full parameter space of the presented simulation model is presented in Table 1.

Notice that in Table 1 some of possible combinations of parameter values have been skipped. A full Cartesian product of all possible parameter values would turn 32 possible model parameterizations, out of which 20 parameterizations would be redundant. The redundancy would arise from the fact that when, for example, a model using logic `logicB` sweeping through parameters `paramA1` and `paramA2` would not extend parameterization space as these parameters would not be used.

Parameter space redundancy has two disadvantages. Firstly, it leads to unnecessary simulation runs which makes a simulation process take more time and resources. Secondly, when simulation results are analyzed, duplicated values should be identified and either duplication should be considered in the model interpretation or the redundant results should be discarded.

Table 1. Parameter space in an example multi-agent simulation

No.	param1	Logic	paramA1	paramA2	paramB
1	1	logicA	3	5	-
2	2	logicA	3	5	-
3	1	logicA	3	6	-
4	2	logicA	3	6	-
5	1	logicA	4	5	-
6	2	logicA	4	5	-
7	1	logicA	4	6	-
8	2	logicA	4	6	-
9	1	logicB	-	-	7
10	2	logicB	-	-	7
11	1	logicB	-	-	8
12	2	logicB	-	-	8

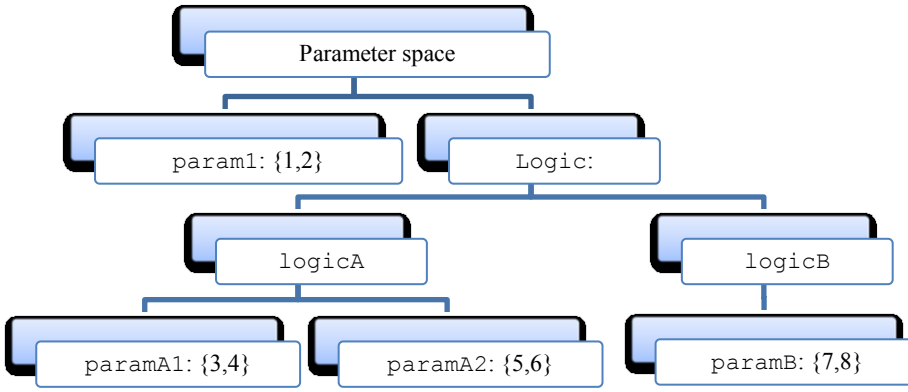


Figure 1. Parameter space of an example multi-agent simulation – tree representation

The solution to the above problems is representing the parameter space of a simulation model as a tree, see Figure 1. The nodes represent model parameters with lists of their possible values. When parameters are dependent on other parameters (especially when parameters' values are objects that can have their own values) a split in a tree is present. In the example considered the value of the `logicA` parameter determines whether the parameters `paramA1` and `paramA2` or `paramB` are used.

The given tree parameter representation can be further unfolded to consider all possible values of a parameter space for the example simulation model. Such an unfolded tree is presented in Figure 2. Leaf selection from the tree in Figure 2 unambiguously determines a particular point in the example model's parameter sweep space, where the values of particular parameters are represented by nodes on the path from the selected leaf to the root node.

Another advantage of using a tree representation of parameter space is the possibility to describe parameter structures in an XML format as proposed in Section 3.

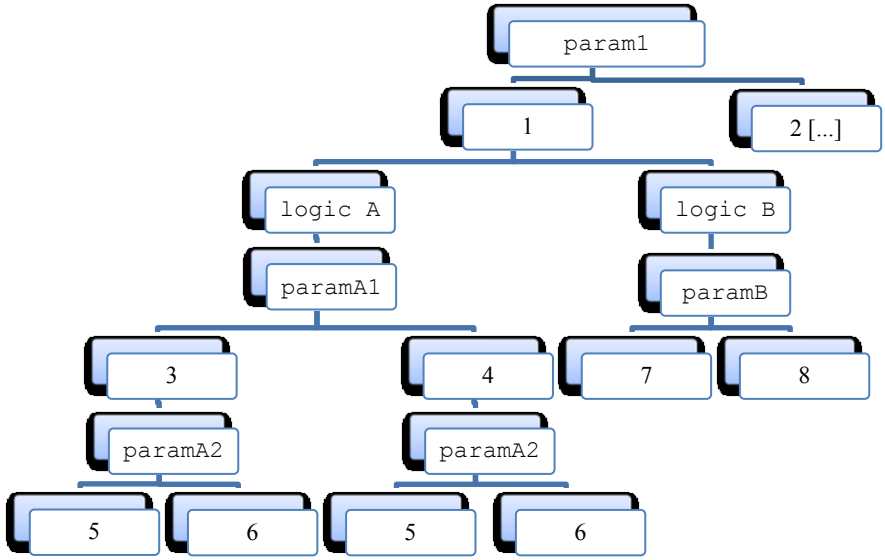


Figure 2. An unfolded parameter value tree for the parameter tree presented in Figure 1. The right site of the tree (for the parameter value param1=2) is identical to the left part (param1=1) and therefore is not visualized in the figure (marked with „[…]")

2.4. Comparison of existing methods for parameter sweep management

In Table 2 several methods for managing parameter sweep in a simulation process have been compared. The criteria were selected based on [5] and enhanced using the authors own experience in simulation models development. Now we shall describe the exact understanding of each criterion presented in Table 2:

- ease of change of parameter values is understood as the possibility of avoiding source code modifications when parameter values or the parameter space are altered;
- separating the model's configuration from its implementation means that model implementation and its parameterization are independent modules in the simulation environment;
- graphical support for configuration editing is understood as providing a GUI that allows the control of parameterization space and parameter values;
- independence from the parameter sweep algorithm means the possibility of changing the parameter space search algorithm (typical algorithms are grid search and *ceteris paribus* search);
- object configuration of parameters values means that parameters can be represented as objects of any class;
- support for the tree structure of parameter space is understood as defined in Section 2.3.

Table 2. Comparison of parameter sweep management methods

No	Criteria	**NetLogo BehaviorSpace**	**Repast Parameter Sweep**	**Batch File**	**Parameter file**	**Source code**	**Mixed**
1	Ease of change of parameter values	+	+	+/-	-	-	-
2	Separating model's configuration from model's implementation	+	+	+	+	-	+/-
3	Graphical support for configuration editing	+	+	-	-	-	-
4	Independence from parameter sweep algorithm	-	-	+	+	+	+
5	Object configuration of model's parameters values	+/-	-	-	+	+	+
6	Support for tree structure of parameter space	-	-	-	-	+	+

The tools presented in Table 2 that facilitate simple and graphical parameter editing have very limited flexibility in experiment design. On the other hand, more flexible methods involve programming and thus have poor usability by researchers other than the creator of the model.

In particular, only methods that are based on source code modification (directly or mixed) allow one to take into account a tree structure of model parameters. However, it requires a separate implementation for every simulation model.

The above review uncovers a gap in presently available methods and tools for model parameterization. An ideal solution should provide a full separation between source code and configuration, allow graphical configuration of simulation scenarios and have enough flexibility to allow complex parameter structures – particularly structures represented as trees.

In Section 3 a methodology that solves the above problems is described together with a prototype implementation called Parameter Sweep Library (PSL).

3. Representation and management of a tree parameter space

The goal of this Section is to propose a method for representing and processing tree parameter spaces of simulation models in object oriented languages – particularly Java[4].

[4] Java is currently the most popular language in multi-agent simulation. In particular, it is used in simulation libraries Repast and Mason. Moreover, the NetLogo simulation environment is also implemented in Java. It is also possible to migrate solutions presented in the paper into C# programming language.

3.1. Technical assumptions

As stated earlier in this paper our focus is on the multi-agent simulation. Macal and North [12] point out that in the multi-agent approach, agents are represented as objects of any kind and as a result agents are usually implemented in object-oriented languages. A configuration of a multi-agent simulation model can take into consideration dependency between objects (e.g. assigning to some agent group a particular decision rule as it was presented in Section 2.3). This leads to the conclusion that a tool for multi-agent model configuration should also support object-oriented parameterization of a simulation model.

Moreover, the review of methods and tools for parameterization management in Section 2.4 leads to the following postulates regarding a tool for simulation scenario management: ease of parameter space change, separating parameterization from source code, graphical support for configuration editing, independence from parameter space search algorithms, object representation of parameters, support for tree parameter structures and the possibility of searching tree parameter spaces. On the basis of these assumptions a parameter description language was created and is described in Section 3.2.

3.2. Language for description of tree parameterization structure of simulation models

The need to manage object oriented configuration of information systems was addressed in various tools. One of these tools is The Spring Framework [9] that is currently developed as an open source project[5]. The framework defines an XML-based language for object dependency description and provides an application container based on inversion of control (IoC) and dependency injection (DI) design patterns [6].

In simulation parameter sweep management the IoC pattern is implemented by giving the control of model's parameter space to an external library rather than managing it within the source code of a simulation model. The DI design pattern means that simulation model parameterization is done not directly in source code, but through an external XML file. Moreover field values that are injected can be both primitive values (such as integer or double) as well as complex objects with dependencies.

Law and Kelton [10] point out that working with a simulation model necessitates performing many experiments,. An example scenario of a simulation experiment could include achieving a steady state[6] by a model and subsequently introduction of a shock to observe how a simulated system reaches a new steady state level.

The Spring Framework allows the implementation of object-oriented configurations for simulation model parameterization. However, the Spring Framework does not provide any tools to manage parameter sweep. This problem is solved in the next Section – an extension to the framework is proposed that enables configuration and management of the parameter sweep process.

---

[5] The web page of the open source project Spring Framework is available at http://www.springsource.org/.

[6] A multi-agent simulation model of a dynamic system reaches a steady state after the model has run for a defined time – initial simulations (warm up) are usually discarded.

3.3. Implementation of tree parameterization description language

The goal of this Section is to present an extension for the Spring framework that will allow configuration and management of the parameter space of a multi-agent simulation model. The software tools involved will be subsequently referred to as Parameter Sweep Library (PSL).

According to the assumptions presented in Section 3.2, a parameterization of a simulation model will be presented as a set of dependent objects. Using the open source Spring Framework means that configuration will be presented in the form of an XML file and can be edited by graphical tools provided by the Spring project.

The proposed PSL library extends Spring Framework's functionality by adding the facility to manage parameter space in a simulation model. The PSL library is targeted for the Java platform and consists of the following components:

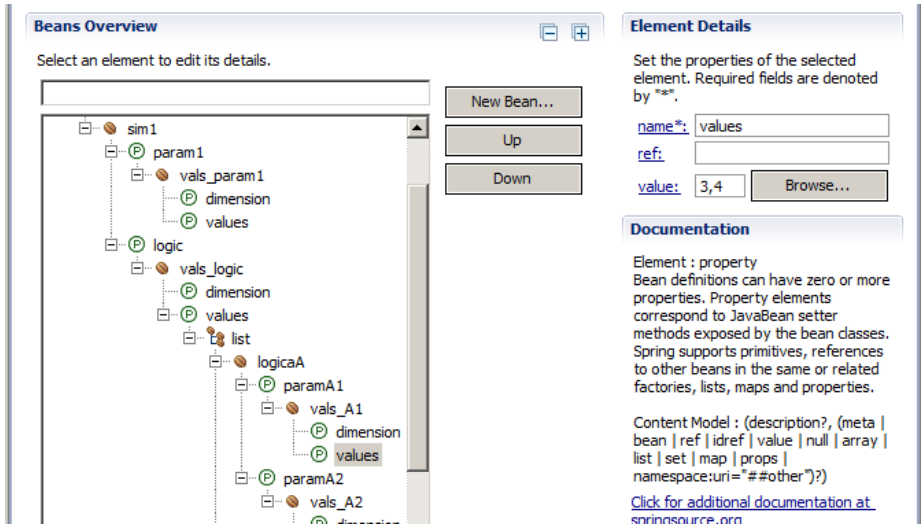
- classes to define manageable attributes in a simulation model;
- attributes that can be potentially parameterized, marked by an annotation;
- classes representing parameter spaces of the simulation model;
- a value converter for the Spring Framework;
- an application programming interface (API) for developers for creating simulation scenarios and managing parameter spaces.

The Java language does not provide any tools for managing object pointers (as it is e.g. in C++). To provide the means to externally control a model's configuration, a hierarchy of enveloping classes has been created for managing parameter values during a simulation process. Appendix 1 presents a class containing a sample simulation configuration for the Parameter Sweep Library. The annotations identify the fields that can be parameterized and a simulation can be run several times for their different values.

The information on generic classes (within the diamond <> operator) is in Java language removed during the code compilation process. As this information is needed during runtime an additional annotation `@ManagableValue` has been created to store information on a data class for data stored within an attribute, see Appendix 1.

As mentioned earlier the crucial feature of the PSL library is the independence of model parameterization from parameter space search algorithms. It is assumed that any parameter search scenario (e.g. grid search or ceteris paribus) can be applied to the same parameterization. Each parameter allows one to define information on a parameter search algorithm.

An object of the `SimC` class presented earlier in Figure 2 and described in Section 2.3 is created by the Spring Framework. Figure 3 presents a graphical configuration of a Java bean of class `SimC`. For each model parameter a set of feasible values (`values` attribute) has been assigned as well as the dimension used in parameter sweeping (`dimension` attribute). The PSL library supports the definition of two parameter space search algorithms: grid search and ceteris paribus. Grid search means that all possible combinations for all values are simulated, while ceteris paribus search is understood as analyzing deviation of each single attribute from standard values while leaving default values of all other attributes.



Source: screenshot from Eclipse IDE

Figure 3. Graphical configuration of a parameter space of a multi-agent simulation model with Spring IDE Eclipse plugin

Support for multi-value configuration of simulation parameters has been achieved through an implementation within the PSL library of a conversion mechanism via the Spring Framework API[7] . During the conversion process the mechanism gathers information on simulation model parameterization configuration and search dimensions, which later can be used to change the configuration state through the PSL API.

The use of a PSL library within a simulation model requires fulfilling of two conditions. Firstly, some parameters should be marked through annotation as configurable. Secondly, simulation managed by the PSL library requires calling the PSL API to create an object representing a simulation parameterization state.

An example of the source code controlling the simulation process can be found in Appendix 2. Firstly, a Spring application context is processed (here it contains simulation configuration). Secondly, a component managing simulation process is created. Thirdly, the component provides objects (here of class `SimC`) that represent simulation configuration. The PSL library calculates the number of possible scenarios (equal to number of leaves) in a parameterization tree. Each parameter set (leaf) can be obtained by calling its number. The PSL library handles the parameter sweeping process by providing configurations on request. The number of available parameter sets is calculated depending on search scenario and parameter dimensions.

The concept of parameterization space management presented above is illustrated in the next Section by a typical example in multi-agent simulations of economic systems.

[7] Documentation for the Spring Framework converter mechanism can be reached at http://static.springsource.org/spring/docs/3.0.x/javadoc-api/org/springframework/core/convert/converter/GenericConverter.html

4. Example – multi-agent artificial market simulation

The goal of this Section is to present an example usage of tree parameter representation in multi-agent artificial market simulation.

Let us consider a multi-agent model with two agent classes: `Customer` and `Firm`. Firms supply products at some `price` while customers observe the prices and present their demand.

Assume that agents – `Customers`:

- analyze prices offered by `Firms`;
- declare their demand to particular `Firms`.

Similarly, let us assume that agents – `Firms` are characterized by the following attributes:

- `price`;
- `demand` declared for their product (as an aggregated value of all customers interested in a firm's products);
- `marginal_cost`.

Using the information above a `Firm` can calculate its profit. For the given artificial market, rules (logic) for agents can be defined. The `Firms` can have one of the following decision rules:

1. `Firm-ANN` – decides on price adjustments based on comparing generated profit measured in historical prices and potential profit measured in the last transactional price. The most profitable price becomes a new price of `Firm-ANN`, but the price is additionally `disturbed` (to reflect imperfect market information) with a normally distributed random value with standard deviation of `Range`;
2. `Firm-DIF` – adjusts prices in similar way to `Firm-ANN`, but a new price results from adjustment by the value of `Range` – the price can be randomly increased or decreased (with probability 0.5) by this factor.

In the similar fashion `Customers` can use one of the following possible decision making rules:

1. `Customer-IMP` – searches an entire market for the lowest offer and chooses it but her information on market prices is imperfect – the price is a normally distributed random variable with `Randomness` as standard deviation. In the result, `Customer-IMP`'s decision is suboptimal and optimality distance depends on value of the `Randomness` parameter;
2. `Customer-VLT` – some group of cases defined by the `Randomness` parameter `Customer-VLT` chooses a random `Firm`, and in other cases chooses the cheapest offer.

It is worth noting that the `Range` and `Randomness` parameters have the same name in different agent classes but have completely different meaning (and different sets of possible values). Such situations are difficult to handle in standard multi-agent frameworks that only provide a grid approach to parameter sweep.

Figure 4 contains a tree representation of parameter space of the given model taking into account different possible logics for different agents classes.

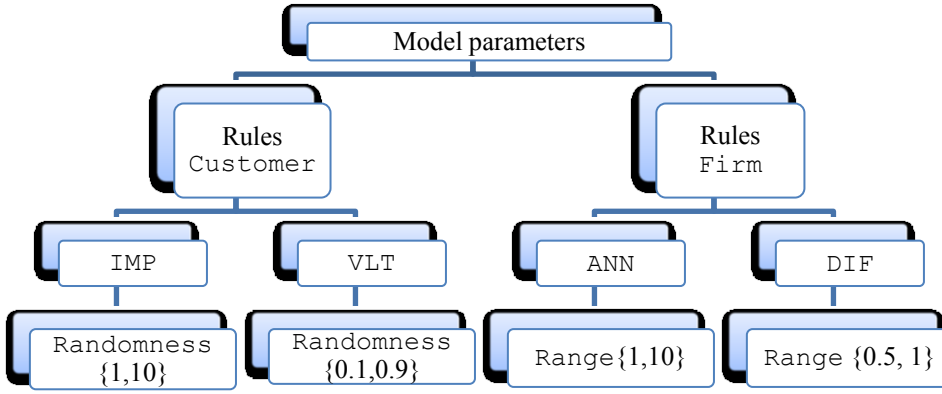


Figure 4. Tree representation of an example multi-agent artificial market model

Table 3. Parameter values for Cartesian product parameter sweep in the presented artificial market example. The selected rows are values used in tree representation

Customer	Random	Firm	Range
IMP	0.1	ANN	0.5
IMP	0.9	ANN	0.5
IMP	1	ANN	0.5
IMP	10	ANN	0.5
IMP	0.1	ANN	1
IMP	0.9	ANN	1
IMP	1	ANN	1
IMP	10	ANN	1
IMP	0.1	ANN	10
IMP	0.9	ANN	10
IMP	1	ANN	10
IMP	10	ANN	10
VLT	0.1	ANN	0.5
VLT	0.9	ANN	0.5
VLT	1	ANN	0.5
VLT	10	ANN	0.5
VLT	0.1	ANN	1
VLT	0.9	ANN	1
VLT	1	ANN	1
VLT	10	ANN	1
VLT	0.1	ANN	10
VLT	0.9	ANN	10
VLT	1	ANN	10
VLT	10	ANN	10

Customer	Random	Firm	Range
IMP	0.1	DIF	0.5
IMP	0.9	DIF	0.5
IMP	1	DIF	0.5
IMP	10	DIF	0.5
IMP	0.1	DIF	1
IMP	0.9	DIF	1
IMP	1	DIF	1
IMP	10	DIF	1
IMP	0.1	DIF	10
IMP	0.9	DIF	10
IMP	1	DIF	10
IMP	10	DIF	10
VLT	0.1	DIF	0.5
VLT	0.9	DIF	0.5
VLT	1	DIF	0.5
VLT	10	DIF	0.5
VLT	0.1	DIF	1
VLT	0.9	DIF	1
VLT	1	DIF	1
VLT	10	DIF	1
VLT	0.1	DIF	10
VLT	0.9	DIF	10
VLT	1	DIF	10
VLT	10	DIF	10

With two possible `Customer`'s rule sets (`IMP` and `VLT`), two possible `Firm`'s rule sets (`ANN` and `DIF`), four possible values of the `Randomness` parameter (0.1, 0.9, 1 and

10) and three levels of the `Range` parameter (0.5, 1 and 10) we could expect 2x2x4x3=48 market types for Cartesian product parameter space, see Table 3.

However, the Cartesian product parameter sweep leads to redundant and ambiguous parameters in comparison to parameters presented in the parameter tree in Figure 4. For example, for `Customer-IMP` analyzed values include 0.1 and 0.9 for the `Randomness` parameter, while the parameter for this case only makes sense for values between 1 and 10.

More specifically, in the Cartesian product there are 48 possible parameter sets while in the parameter tree there are only 16 (it is the number of leaves in the tree presented in Figure 5). In the presented example the tree parameterization can decrease the number of calculations by three times in comparison to a Cartesian product search. Additionally, it allows for proper processing in cases where parameters with the same names but from different leaves and of different sets of allowed values exist.

Example source code for parameter sweep management for the above example has been presented in Appendix 3.

Figure 5 presents a tree parameterization structure of an artificial multi-agent market model, where each path from the leaf to the root node represents one complete model parameterization. There are 16 leaves in the tree i.e. 16 possible market configurations. While the tree presented on Figure 4 presents groups of values in the model, the tree in Figure 5 allows one to calculate exactly the number of possible parameterizations.

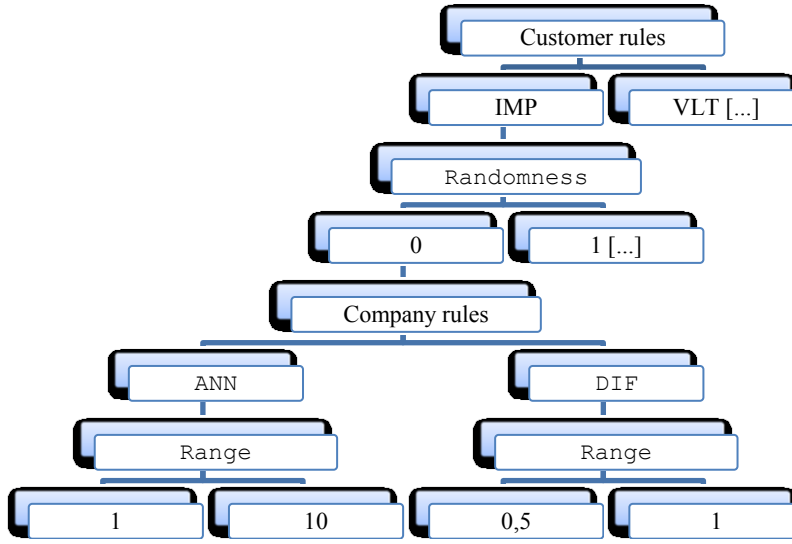
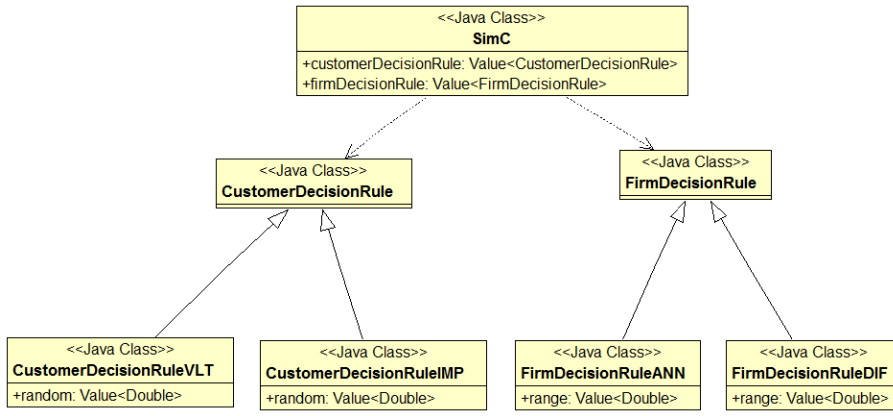


Figure 5. Part of a tree representing full parameter space of the example artificial market simulation model. Symmetrical parts of the tree which are not visualized have been marked with „[…]"



Source: generated with Eclipse IDE

Figure 6. Class diagram for the example parameterization

The presented parameterization can be described as a set of objects, see Figure 6. This means that it can be processed along with the approach presented in Section 3 – i.e. it can be represented in the Spring Framework XML and handled by the proposed Parameter Sweep Library.

The approach presented for representing and managing tree parameter structures enables the number of simulated scenarios for a parameter sweep to be limited – in the example shown from 48 scenarios that would require evaluation when using currently available tools to 16 obtainable when using the proposed parameter sweep library.

5. Discussion

Existing popular agent-based simulation platforms provide tools for managing parameter sweep only for grid-structured parameter spaces. In agent-based simulation object-oriented simulation configuration leads to a tree parameter representation. The tree parameter structure requires either performing redundant simulations, coding the parameter sweep or generating batch files. The proposed approach fills this gap by providing description of a tree parameter space representation and tools allowing for processing the tree parameter space.

The proposed approach and tools have significant advantages but also present additional requirements in terms of the simulation development process. In this Section both the advantages and disadvantages of the proposed approach will be discussed.

The advantages include: (1) universality, (2) standardization, (3) support for changing dimensions and (4) separation of a simulation's configuration from its implementation.

Universality is understood as ability of the approach to be used in any simulation model written in Java[8]. The library is particularly useful in multi-agent simulations when the

[8] The Spring Framework is also available for C#, so the proposed library can be also implemented for the .NET framework.

simulation framework is used along library-oriented simulation framework [12][9]. The proposed tree representation of parameter space can be applied to illustrate complex parameter sweep scenarios.

The second advantage is possible *standardization* of the simulation process. Integration with the Spring Framework allows for a standard XML-based description of simulation scenarios.

The third advantage of the proposed library is *support for changing the dimensions* of parameter sweep space. This is particularly important during the early stages of a simulation process where the model needs to be verified and validated, see Law and Kelton (2000). During the verification and validation phases different parameter ranges and parameter sets are altered several times. Moreover, XML parameter sweep configuration files can be generated by other tools which adds another level of configuration.

The fourth advantage is a complete *separation of a simulation's configuration from its implementation*. This enables the delivery of simulation models to end users in binary form (without source code) while still giving them complete control over model configuration. Moreover, simulation scenarios can be processed and stored independently of a model's binaries.

The framework proposed in the paper also has disadvantages, namely increased memory and resource usage, requirements for additional definitions in object-oriented simulation model configuration, and programming integration-code to attach the PSL library to a simulation model.

The *increased memory usage* is caused by the fact that the Spring Framework is a large set of libraries that adds a significant memory and start-up time overhead. This means that more RAM memory needs to be available to a Java Virtual Machine. However this problem can be overcome as RAM memory is relatively cheap and a single Spring container can be responsible for managing several threads, with each thread running several simulations in sequence. This means that when running many simulation repetitions in parallel threads the Spring overhead can be neglected.

A need for *additional definitions* in an object-oriented simulation model configuration arises from the fact that the proposed approach requires annotations in fields of Java beans that are used to represent the parameter space of a simulation model. The annotations tell the environment that a field is part of the parameter space. This is an additional step in defining beans representing an object-oriented configuration of a simulation model. The additional work makes the framework's implementation in existing simulation models more complicated. However, annotation of the definition process is still relatively little work in comparison to creating and updating a parameter sweep mechanism tailored to a particular tree-structured simulation model. Moreover, we are working to overcome this problem in the future by building a plug-in to the Spring Framework that allows the use of plain Java beans (i.e. POJO) for the simulation model's configuration and simulation scenarios could be defined entirely in a Spring XML file.

[9] Macal and North [12] review agent-based simulation tools and define three types of frameworks: library-oriented, IDE-oriented and hybrid. Library oriented means that a simulation process is managed by an library linked to a simulation code. IDE -oriented means that an IDE (Integrated Desktop Environment) i.e. graphical tools are used to construct a simulation model. A hybrid approach is a mix of both previous [12].

The PSL library needs to be linked with a simulation's code. *Programming integration-code* to attach the PSL library to a simulation model adds an additional programming overhead. However, the code for a parameter sweep is standardized and usually does not need be updated when a simulation model changes.

The advantages described overwhelm the disadvantages, which means that the PSL framework can streamline an agent-based simulation process in with popular tools for the Java programming language. It is also worth mentioning that the usability of the proposed approach has been verified in multi-agent simulation - in parameter sweep for a multi-agent model of educational markets [17].

6. Concluding remarks

The paper tackles an issue of managing the parameterization processes in multi-agent simulation. The review of existing software shows that the currently available tools are incomplete because they do not provide any support for tree parameterization structures. Moreover, these approaches exhibit a significant trade-off between configuration flexibility and availability of a graphical user interface.

In this paper a solution to the above problems has been presented. The Spring Framework enables the construction of an object-oriented simulation configuration. It has graphical tools for configuration editing, while the proposed Parameter Sweep Library (PSL) enables simulation process handling. The software has been implemented in Java – the most popular language in multi-agent simulation support software.

The proposed approach allows for construction of hierarchical parameter spaces through XML file and provides tools to manage such parameterization in simulation runtime. It also supports visualization of object structure in standard tools. A strict separation from a simulation source code allows for various parameter sweep strategies. Another advantage of the PSL framework is that it allows for standardization in simulation parameterization with techniques commonly used in business software – XML and the Spring Framework.

References

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

[2] Bellifemine F. L., Caire G., Greenwood D., *Developing Multi-Agent Systems with JADE*, John Wiley & Sons, 2007.

[3] Berryman M., *Review of Software Platforms for Agent Based Models*, Land Operations Division, Defence Science and Technology Organisation, Edinburgh, Australi, 2008.

[4] Bragen M., Altaweel M., *Repast Parameters Sweeps Getting started*, http://repast.sourceforge.net/docs/RepastParameterSweepsGettingStarted.pdf, accessed on 2012-03-12, 2012.

[5] Daum T., Sargent R.G., Experimental frames in a modern modeling and simulation system, *IIE Transactions* **33**, 2001, 181-192.

[6] Fowler M., *Containers and the Dependency Injection Pattern*, http://martinfowler.com/articles/injection.html, accessed on 2012-04-14, 2004.

[7] Gilbert N., Troitzsch K.G., *Simulation for the Social Scientist*, 2nd Edition, Open University Press, Berkshire, 2005.
[8] Gilbert N. Agent-based models, in: T. F. Liao (ed.) *Quantitative Applications in the Social Sciences* 153, SAGE, 2008.
[9] Johnson R., Hoeller J., *Expert One-on-One J2EE Development without EJB*, Wiley Publishing, Indianapolis, 2004.
[10] Law A. M., Kelton W. D., *Simulation Modeling and Analysis*, McGraw-Hill, Boston, 2000.
[11] Macal C., North M., Tutorial on Agent-based Modeling and Simulation, in: Kuhl M. E., Steiger N.M., Armstrong F.B., Joines J.A. (eds.), *Proc. 2005 Winter Simulation Conference*, Orlando, FL, 2005, 2-15.
[12] Macal C.M., North M.J., Tutorial on agent-based modelling and simulation, *Journal of Simulation*, **4**, 2010, 151-162.
[13] NetLogo, *NetLogo BehaviorSpace Guide*, http://ccl.northwestern.edu/netlogo/docs/behaviorspace.html, 2012.
[14] Nikolai C., Madey G., Tools of the Trade: A Survey of Various Agent Based Modelling Platforms, *Journal of Artificial Societies and Social Simulation*, **12**, 2, http://jasss.soc.surrey.ac.uk/12/2/2.html, 2009.
[15] Phelps S., *Applying dependency injection to agent-based modelling: the JABM framework*, CCFEA Working Paper #WP056-12, 2012.
[16] Railsback S.F., Lytinen S.L., Jackson S.K., Agent-based Simulation Platforms: Review and Development Recommendations, *Simulation*, **82**, 2006, 609-623.
[17] Szufel P., *On educational process cost efficiency*, doctoral thesis, The Collegium of Economic Analysis, Warsaw School of Economics, 2011.
[18] Thiele J.C., Kurth W., Grimm V., *RNetLogo: an R package for running and exploring individual-based models implemented in NetLogo*, Methods in Ecology and Evolution, British Ecological Society, Early Preview, 2012.
[19] Tobias R., Hofmann C., Evaluation of free Java-libraries for social-scientific agent based simulation, *Journal of Artificial Societies and Social Simulation* **7**, 1, http://jasss.soc.surrey.ac.uk/7/1/6.html, 2004.
[20] Waltemath D., Bergmann F.T., Adams R., Le Novere N., *Simulation Experiment Description Markup Language (SED-ML): Level 1 Version 1*, http://sed-ml.org/, 2011.
[21] Wainer G.A., Dalle O., Software Tools, Techniques and Architectures for Computer Simulation, *Simulation*, **86**, 2010, 267-269.
[22] Wikipedia, *Comparison of agent based modelling software*, http://en.wikipedia.org/wiki/Comparison_of_agent-based_modeling_software, accessed 2012-08-20, 2012.
[23] Zeigler B.P., *Theory of Modelling and Simulation*, Wiley, New York, 1976.

Presented at the XII Conference: Systems and Operational Research – BOS 2012, 17-19 September 2012, Warsaw, Poland

Appendix 1

Source code defining a Java Bean type class that allows to describe parameter space of example represented in Section 2.3.

```
public class SimC {
            @ManagableValue(clazz=Double.class)
                public Value<Double> param1;
            @ManagableValue(clazz=Logic.class)
                public Value<Logic> logic;
          public Value<Logic> getLogic() {
                return logic;
          }
          public void setLogic(Value<Logic> logic) {
                this.logic = logic;
          }
          public Value<Double> getParam1() {
                return param1;
          }
             public void setParam1(Value<Double> param1) {
                this.param1 = param1;
          }
}
```

Appendix 2

An example source code of the PSL library utilization to manage a tree parameter space of a multi-agent simulation model.

```
//initialization and application context creation
FileSystemXmlApplicationContext springCtx = new
  FileSystemXmlApplicationContext("sim.xml");
//creation of an object allowing to manage
//the parameter space
SimConfigService scs =
  springCtx.getBean("simConfigService",SimConfigService.class);
//parameter space managemnt;
//in case of multi-threaded parrarel simulation each thread
//is connected to different parameter space
DynamicSimConfiguration dsc =
  scs.getNewDynamicSimConfiguration();
//acquiring an object representing simulation state
SimC simC = dsc.getBean("simC", SimC.class);
//parameter sweep space size
int paramSweepSize =
  dsc.getSimulationVariantsSize(dsc.getMetaDimension("PARAM"));
for (int paramIx=0; paramIx < paramSweepSize; paramIx++) {
  //setting of a particular parameter combination
  //can be acquired by givem its number
  dsc.setSimIx(dsc.getMetaDimension("PARAM"), paramIx);
  //[model is parametrized and simulation starts]
}
```

Appendix 3

An example source code that is processed through the Spring Framework and the PSL library. The source code describes parameterization of the example model that was presented in Figures 4 and 5.

```
<bean id="sim1" class="example.SimC" scope="prototype">
    <property name="customerDecisionRule">
        <bean class="dynsimconfig.arrays.ArrayValue" id="vals_custDecRule">
                <property name="dimension" ref="SCEN" />
                <property name="values">
                    <list>
                        <bean class="example.CustomerDecisionRuleIMP" id="IMP">
                            <property name="randomness">
                                <bean class="dynsimconfig.arrays.ArrayValueD" id="vals_IMP">
                                        <property name="dimension" ref="SCEN" />
                                        <property name="values" value="1,10" />
                                </bean>
                            </property>
                        </bean>
                        <bean class="example.CustomerDecisionRuleVLT" id="VLT">
                            <property name="randomness">
                                <bean class="dynsimconfig.arrays.ArrayValueD" id="vals_VLT">
                                        <property name="dimension" ref="SCEN" />
                                        <property name="values" value="0.1,0.9" />
                                </bean>
                            </property>
                        </bean>
                    </list>
                </property>
        </bean>
    </property>
    <property name="firmDecisionRule">
        <bean class="dynsimconfig.arrays.ArrayValue" id="vals_firmDecRule">
                <property name="dimension" ref="SCEN" />
                <property name="values">
                    <list>
                        <bean class="example.FirmDecisionRuleANN" id="ANN">
                            <property name="randomness">
                                <bean class="dynsimconfig.arrays.ArrayValueD" id="vals_ANN">
                                        <property name="dimension" ref="SCEN" />
                                        <property name="values" value="1,10" />
                                </bean>
                            </property>
                        </bean>
                        <bean class="example.FirmDecisionRuleDIF" id="DIF">
                            <property name="randomness">
                                <bean class="dynsimconfig.arrays.ArrayValueD" id="vals_DIF">
                                        <property name="dimension" ref="SCEN" />
                                        <property name="values" value="0.5,1" />
                                </bean>
                            </property>
                        </bean>
                    </list>
                </property>
        </bean>
    </property>
</bean>
```