Proceedings of the Operational Research Society Simulation Workshop 2016 (SW16)
A. Anagnostou, K. Hoad, and M. Kunc, eds.

AN INTRODUCTION TO AGENT-BASED SIMULATION AS A DECISION-SUPPORT TOOL

Bhakti Stephan Onggo

Department of Management Science
Lancaster University Management School
Lancaster University, United Kingdom
s.onggo@lancaster.ac.uk

ABSTRACT

Agent-based simulation (ABS) has been used widely in various application areas. This paper provides a tutorial on ABS. In addition to the theoretical foundations, such as the definitions of key ABS concepts and the structure of an ABS model, this tutorial also explains the technical aspect, such as how an ABS model is implemented using a software tool and how ABS is used to solve a problem. The software tool used in this tutorial is Repast. The last part of the tutorial discusses the challenges that need to be addressed in order to increase the confidence of decision-makers, who may not be familiar with computer programming, when using ABS to help them make decisions. The challenges include conceptual model representation, validation and participatory modelling.

Keywords: Agent-based simulation, Repast, conceptual modelling, validation, participatory modelling

1 INTRODUCTION

Agent-based simulation (ABS) has been applied in various areas. In business, Bonabeau (2002) categorised the application of ABS into four categories: flow management (e.g. pedestrians, traffic), markets (e.g. stock, electricity), organisational behaviour (inter- and intra-organisation such as manufacturing and supply chain) and diffusion (the spread of something in a society). Macal and North (2010) provide a different categorisation which includes epidemics, markets and socio-technical systems. Although Bonabeau (2002) and Macal and North (2010) do not mean to conduct extensive surveys, their observations are largely supported by recent surveys. Diffusion in a society is one of the main business applications of ABS, as we can see from the review by Kiesling et al. (2012). This is understandable because an ABS model can model people and how they interact within their social networks. Similarly, using ABS to model a market is sensible because a market would not exist without the interaction of trading agents. An ABS model can model how trading agents interact within their business networks by making decisions such as buy, hold or sell. An example of a review paper for the electricity market is given by Sensfuß et al. (2007). For the same reasons as in the previous two applications, the application of ABS in a supply chain or pedestrian/crowd management is also popular. A number of examples on the application of ABS in supply chains can be found in Peidro et al. (2009), who survey the quantitative models used in supply-chain planning, which includes ABS. A review on the modelling techniques used in crowd simulation which include ABS is conducted by Zhou et al. (2010).

Given the increase in the popularity of ABS, there have been a number of tutorial papers written to introduce ABS. Some tutorials focus more on the theoretical foundations of ABS (e.g. Chan et al. 2010, Macal and North 2010). Others explain both the theoretical foundations and technical knowledge (i.e. how an ABS model can be implemented using a software tool); for example, Macal and North (2007) use Ms Excel and Terna (1998) uses SWARM. This tutorial paper also explains both the theoretical and implementation aspects of ABS. The ABS concepts described in this paper are implementation-

independent, so they can be applied to other software tools. As for the software tool here, this paper uses Repast (North et al. 2013) which can be downloaded from http://repast.sourceforge.net/. The second part of the paper discusses the challenges that need to be addressed in order to increase the confidence of decision-makers, who may not be familiar with computer programming, when using ABS to help them make decisions.

To make the explanation more concrete, this paper will use Schelling's segregation case (Schelling 1978) as an example. This case is chosen because of its simplicity although, at the same time, it is able to demonstrate the advantages of ABS. The aim of the study is to explain why segregation exists in a certain society. One possible explanation is that segregation may occur if each individual in society does not want to be in a minority in his/her neighbourhood. An ABS model is developed to prove that an individual's decision to move or stay in his/her neighbourhood is able to generate the pattern (i.e. segregation) observed at the population level. All the computer code used in this paper is taken from the sample code provided by Repast.

The remainder of this paper is structured as follows. Section 2 explains ABS and its main components (agents, environment and behaviours/interactions). The approach used in this tutorial paper is to explain a concept followed by its implementation in Repast. The objective is to help the reader understand the concept better, not only by explanation of the concept but also by looking at the implementation of it. Section 3 highlights the challenges that we need to address to encourage decision-makers to use ABS as a decision-support tool. The challenges include conceptual model representation, model validation and participatory modelling.

2 AGENT-BASED SIMULATION

There are a number of terminologies that are similar to ABS, such as multi-agent systems (MAS), agent-directed simulation and distributed agents. Some articles use different terminology, such as MAS and ABS, to refer to the same thing. On the other hand, different articles may use the same terminology, such as ABS, to refer to two different things. This is understandable given the fact that these related techniques have been applied in various application areas relatively independently.

This paper adopts the following definition: "An ABS model is a simulation model that is formed by a set of autonomous agents that interact with their environment and other agents through a set of internal rules to achieve their objectives." ABS is one of the paradigms in simulation modelling (others include discrete-event simulation (DES) and system dynamics (SD)). ABS modellers see the world as a set of interacting agents inhabiting in an environment. Their view is different from that of DES modellers who see the world as a set of interacting queues and activities, or the view of SD modellers who see the world as a set of interconnected flows and stocks. The agents in ABS are autonomous, meaning that we do not model a central controller who coordinates the behaviour of agents. Hence, agents are capable of making independent decisions. For example, if we simulate the movements of individuals in a city, each agent (i.e. individual) makes an independent decision whether or not to move to a new neighbourhood or stay in the current one. Agents interact with other agents directly or indirectly (through their environment). These interactions are driven by rules that are internal (or local) to each agent. These rules are followed because each agent wants to achieve its objectives (for example, an agent wants to be in a majority in his/her neighbourhood). The interactions between agents may generate a certain pattern that appears at the population level (known as emergence behaviour). For example, the interaction between agents may generate a segregation pattern. This segregation pattern emerges from the interaction of agents, rather than a central controller in the model. Arguably, the main advantage of ABS comes from its ability to generate emergence behaviour from local interactions between heterogeneous agents. Given the definition above, it is clear that the main components of an ABS model are a set of agents in an environment and their behaviours/ interactions.

2.1 Agents

There is no consensus in the ABS literature on the definition of an agent (North and Macal 2007). Instead, we find a spectrum of complexity in the definition of an agent. At one extreme, the composition of an ABS model can be as simple as a set of homogeneous agents with simple attributes (such as speed and direction) and simple behaviours (such as move and accelerate). This type of agent is referred to as a pseudo-agent by North and Macal (2007). At the other extreme, an ABS model can be formed by a set of heterogeneous agents with various complex attributes (such as memory) and abilities (such as communication, perception, planning and learning).

Consistent with the ABS definition used in this paper, an agent is defined as "an entity that can make an independent decision in order to achieve its objectives". An agent can be human or non-human. When we develop an ABS model, it is important to understand the difference between agent and agent type. An agent type is a modelling construct that represents all agents that can be identified by using the same set of characteristics and are able to perform the same set of activities. In a hospital model, the types of agent include patient, clerk and doctor. A patient is an agent type because all patients can be identified by using the same set of characteristics (such as patient ID/social-security number, name and severity level) and are able to perform the same activities (such as arrive at the clinic, complete a form). From an agent type, we can generate one or more agents of the same type. In other words, an agent is an instance of agent type. For example, from the agent type patient, we can generate two patients called Joe and Jane. Both Joe and Jane have a unique ID. They also need to perform the same activities, such as arrive at the hospital, but their arrival and method of transport can be different.

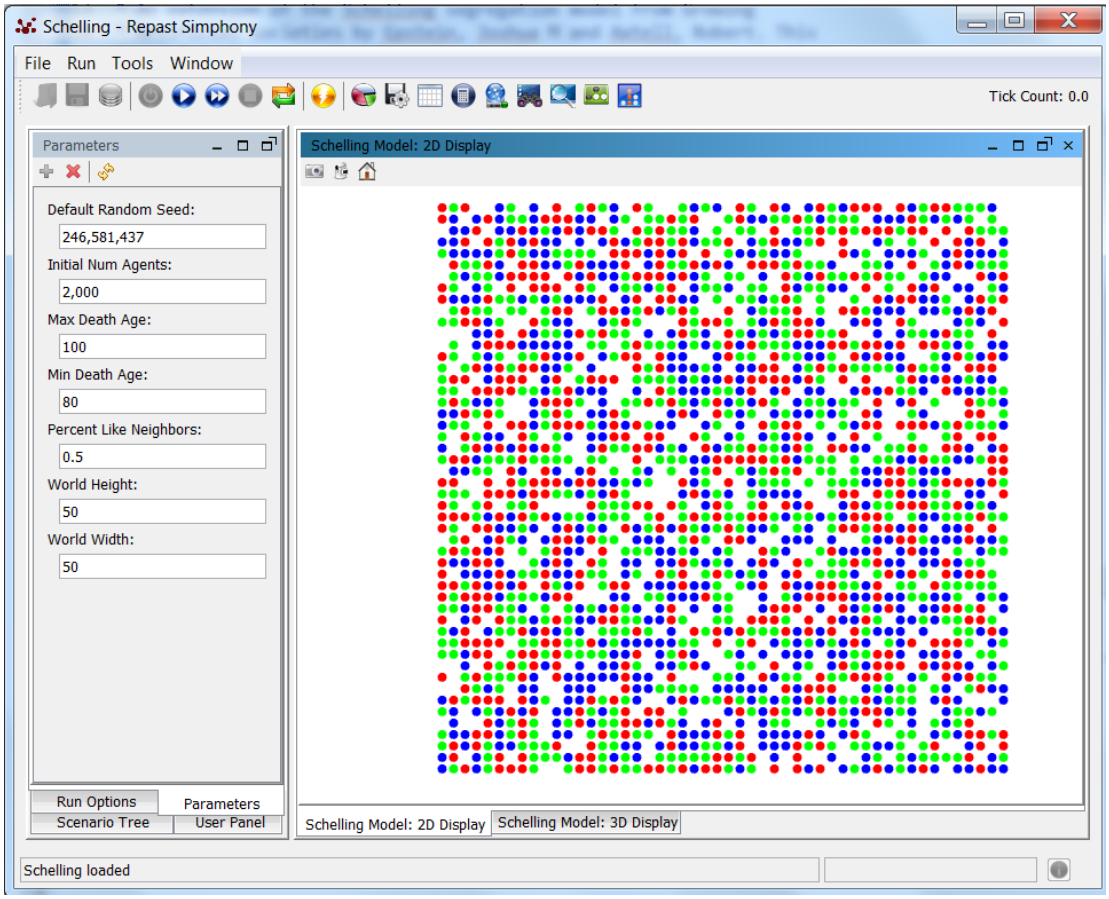

Figure 1 *Repast's user interface showing the Schelling's segregation model*

In Repast, we need to define an agent type as a class. The following code shows how to define an agent type used in the segregation model. Line 01 defines a class (i.e. agent type) called *Agent*. We can change this name to another name, for example *Individual*. This agent type has a number of characteristics/ properties as defined in lines 02–06. Line 07 is a constant to indicate the number of colours that may represent ethnicity or any group to which the agent belongs (Figure 1 shows that each agent belongs to one of three colours). Line 08 is the constructor for the agent type. This is where we should perform the initialization. Lines 09–10 define a one-agent behaviour called *step*, which is executed in every time step from time step 0. This synchronous time-advance mechanism is common to ABS where each agent performs an activity in every time step. In addition to this synchronous time-advance mechanism, Repast provides a very time-flexible scheduler that allows us to have advanced control over when an activity should be performed by an agent. Line 11 defines another behaviour called *die*, which is only executed when this method is called.

```
01. public class Agent {
02.   private int maxAge, currentAge, type;
03.   private double percentLikeNeighbors;
04.   private String id;
05.   private double numLikeMe;
06.   private double neighborCount;
07.   private int numberOfAgentTypes = 3;
08.   public Agent(String id) { /* TO BE ADDED */ }
09.   @ScheduledMethod(start=0,interval=1)
10.   public void step() { /* TO BE ADDED */ }
11.   public void die(){ /* TO BE ADDED */ }
12. }
```

The detailed code for the agent's initialisation is shown below. When an agent is created, it will be assigned a unique ID (lines 01–02). Line 03 is standard code to retrieve simulation parameters from the user interface (Figure 1). Each parameter shown in the user interface (such as "Min Death Age" and "Max Death Age") is linked to the code using a unique identifier. For example, the parameter "Min Death Age" is linked to the code using the identifier "minDeathAge" (line 04). Subsequently, we store the parameter value (80 in Figure 1) as the variable *minDeathAge*. The variables *minDeathAge* and *maxDeathAge* are used to generate the maximum age of an agent based on a uniform distribution (line 07). Line 06 assigns the colour of the agent that will be used to decide whether the agent will move to a new neighbourhood or stay in the current one. Line 08 retrieves the minimum percentage of neighbours with the same colour that will make the agent stay in the current neighbourhood. This means that all agents have the same threshold (homogeneous). We can make the agents heterogeneous by sampling the threshold values using a distribution function, such as the uniform distribution used in line 07.

```
01. public Agent(String id) {
02.   this.id = id;
03.   Parameters p = RunEnvironment.getInstance().getParameters();
04.   int minDeathAge = (Integer)p.getValue("minDeathAge");
05.   int maxDeathAge = (Integer)p.getValue("maxDeathAge");
06.   this.type = RandomHelper.nextIntFromTo(0, numberOfAgentTypes-1);
07.   this.maxAge = RandomHelper.nextIntFromTo(minDeathAge,maxDeathAge);
08.   this.percentLikeNeighbors = (Double)p.getValue("percentLikeNeighbors");
09. }
```

2.2 Environment

The second component in an ABS model is the environment where the agents live. The environment can be spatial or relational. The spatial environment refers to the physical location where an agent lives, such as a region, city or coordinates. The relational environment refers to the connections between agents, e.g. social network, communication network or friendship network.

The environment is an important part of an ABS model because it may affect agents and their behaviours. In the segregation example, an individual makes a decision to move or to stay based on the state of his/her neighbourhood. Furthermore, the environment can be used as a medium for the interactions between agents. In this case, an agent may change the environment, which consequently affects other agents. In the segregation example, when an individual moves, s/he changes the state of the old and new neighbourhoods. These changes will affect the decisions made by other individuals in the old and new neighbourhoods. Often, we need to model an environment to be dynamic, i.e. the environment changes even when all agents do nothing to it. In other words, an environment can have behaviours too. Hence, in the implementation, it can be implemented as an agent. All of these show that the environment is an important part of an ABS model.

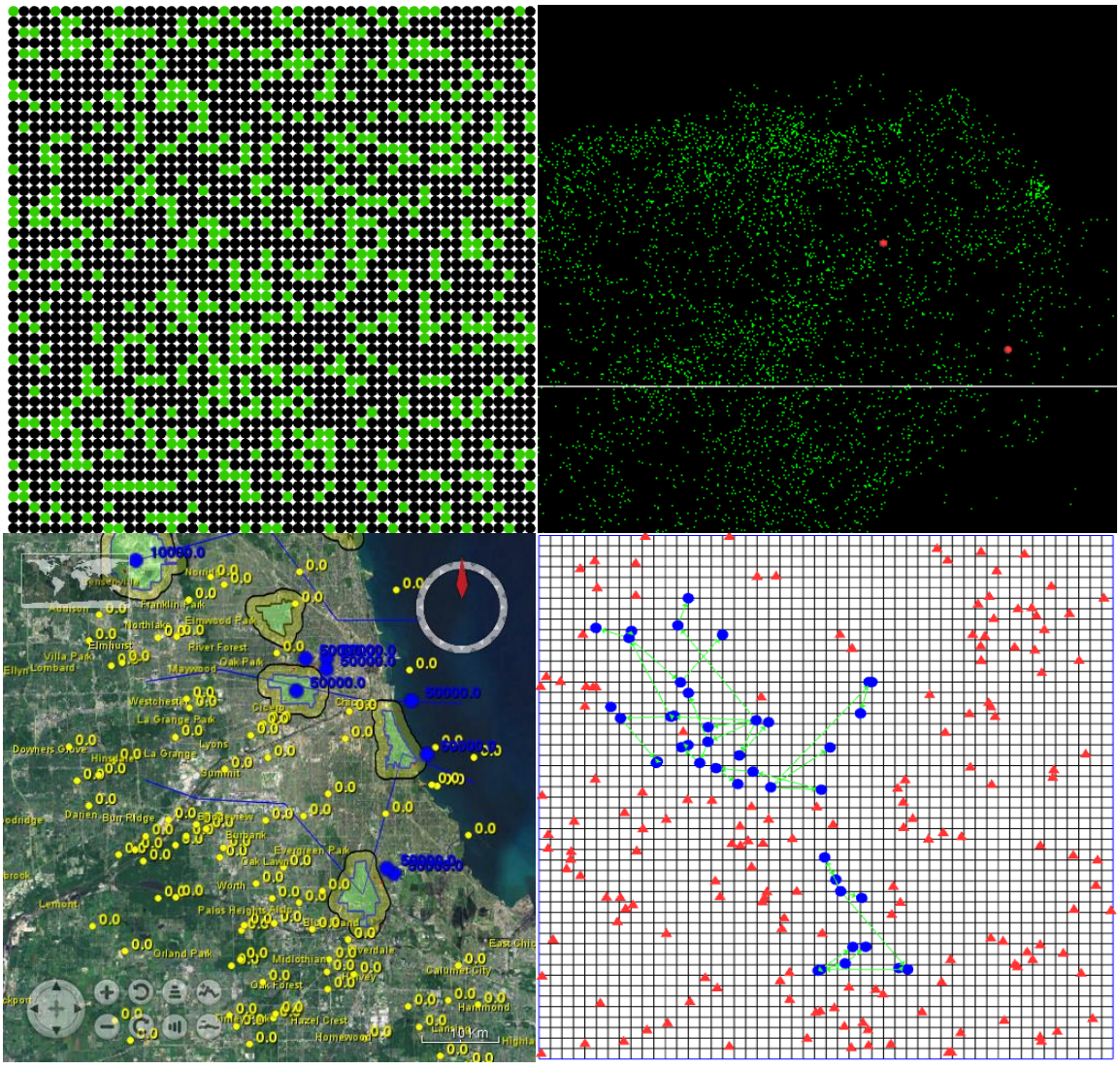

Figure 2 *Repast's projections (from top left clockwise: grid, continuous space, network on top of grid and space, GIS) – all taken from samples provided by Repast*

Repast implements the environment in an ABS by adding a *Projection* to a context. Hence, a context must exist before a projection is added. A projection imposes a certain structure on agents in a context. The

current version of Repast supports the following structures: Grid, Continuous Space, Geographical Information System (GIS) and Network, as shown in Figure 2.

A grid projection creates cells that are structured in one or more dimensions. For example, a 2D grid creates a chessboard-like environment for the agents. In this projection, each agent lives in a cell. This is a structure used in many popular ABS models including Schelling's segregation model. The code snippet below shows how to add a grid projection to a context. The approach follows the factory design pattern (one of the commonly used design patterns in software development). First, we need to create a factory (line 01). Then, we create a grid projection using the factory (line 02). To create a grid projection, we need to give a name to the grid projection, the context and the grid parameters. A description of all the parameters can be read from the Repast API document (it comes with the software). In the example below, the grid will have wrap-around borders (like a toroid), the agents will be added to the grid manually (hence, agents will not be allocated to cells until we do it ourselves), multiple agents can live in the same cell in the grid, and the size of the grid is 100×100 cells.

```
01. GridFactory gridFactory = GridFactoryFinder.createGridFactory(null);
02. grid = gridFactory.createGrid("a grid name", context,
      new GridBuilderParameters<Object>(
        new WrapAroundBorders(),new SimpleGridAdder<Object>(), true, 100, 100
      )
    );
```

In an *n*-dimensional continuous-space projection, the location of an agent is represented by a vector (x_1, x_2, .., x_n). The approach to add a continuous-space projection to a context is the same as in grid projection. The example below creates a continuous space projection in which agents will be added to the space in random locations, the space forms a toroid and the size of the space is 100×100.

```
01. ContinuousSpaceFactory spaceFactory =
      ContinuousSpaceFactoryFinder.createContinuousSpaceFactory(null);
02. space = spaceFactory.createContinuousSpace ("a space name", context,
      new RandomCartesianAdder<Object>(),
      new repast.simphony.space.continuous.WrapAroundBorders(), 100, 100
    );
```

A GIS projection enables us to model a more realistic-looking geographical area in which agents live. In this projection, each agent is associated with a spatial geometry, such as a polygon or point. This projection by default uses a longitude-latitude coordinate reference system. The approach to create this projection is the same as in grid projection, as shown in the snippet below.

```
01. GeographyParameters geoParams = new GeographyParameters();
02. Geography geography =
      GeographyFactoryFinder.createGeographyFactory(null).createGeography(
        "Geography", context, geoParams);
```

A network projection is used to create a relationship network between the agents. The relationship can be physical (such as a road network or a water-distribution network) or non-physical (such as a social network or a professional network). The approach to create a network projection is slightly different from the other projections. A network is created using *NetworkBuilder*, as shown in lines 01–02 below. The last parameter in line 01 specifies whether or not the network is directed. The code snippet below creates a vertex for each agent in the context. However, the edges between vertices (i.e. relationships between agents) are not created. We need to add relationships using an *addEdge* method later. It is possible to create a network with edges from an existing file or from an algorithm. In this case, we can insert a new line between lines 01 and 02 and call the method *load* to load a network from a file, or use *NetworkGenerator* to generate

a network from an algorithm such as small-world or random. We can also implement our own network-generator algorithm.

```
01. NetworkBuilder<Object> netBuilder =
      new NetworkBuilder<Object>("a network name", context, true);
02. netBuilder.buildNetwork();
```

Repast allows us to associate multiple projections with a context. This means we can build an ABS model with several layers of environment. For example, in a communications-network simulation, we can define three layers of environment, such as a grid, to model all the cell stations that serve the agents' mobile phones, a continuous space to model the physical location of agents and a network to represent the current active mobile communications between agents.

2.3 Behaviours

One of the main advantages of ABS is its ability to use agents' behaviours and interactions to generate (and explain) a pattern or behaviour observed at the system level. The behaviour of an agent includes an update to its internal properties and an interaction with other agents or the environment. This behaviour is often implemented as a computer code. Some tools (including Repast) provide a facility to implement behaviour using a graphical notation, such as UML State Chart. The behaviour of individuals in the segregation model is implemented in the method *step* (we can change the name to another). The detailed code is shown below.

```
01. @ScheduledMethod(start=0,interval=1)
02. public void step() {
03.   Context<Agent> context = (Context)ContextUtils.getContext(this);
04.   Grid<Agent> grid = (Grid)context.getProjection("Grid");
05.   boolean lookingForNewSite = true;
06.   while(lookingForNewSite) {
07.     VNQuery<Agent> query = new VNQuery<Agent>(grid, this);
08.     numLikeMe = 0;
09.     neighborCount = 0;
10.     for (Agent agent : query.query()) {
11.       if (agent.getType() == this.getType()) numLikeMe++;
12.       neighborCount++;
13.     }
14.     if (numLikeMe / neighborCount >= percentLikeNeighbors){
15.       lookingForNewSite = false;
16.     }
17.     else {
18.       int width  = grid.getDimensions().getWidth();
19.       int height = grid.getDimensions().getHeight();
20.       int x = RandomHelper.nextIntFromTo(1, width-1);
21.       int y = RandomHelper.nextIntFromTo(1, height-1);
22.       while(grid.getObjectAt(x,y) != null) {
23.         x = RandomHelper.nextIntFromTo(1, width-1);
24.         y = RandomHelper.nextIntFromTo(1, height-1);
25.       }
26.       grid.moveTo(this, x,y);
27.     }
28.   }
29.   currentAge++;
30.   if (currentAge >= maxAge) this.die();
31. }
```

Line 03 retrieves the context of the individual. This context is used in line 04 to retrieve the grid projection. The loop from line 06 to 28 is repeated until the individual is happy with the state of his/her neighbourhood. Line 07 retrieves all the neighbours around the individual. The loop from line 10 to 13

counts the number of neighbours with the same colour. Line 14 checks if the proportion of neighbours with the same colour is above a threshold value. If yes, the individual will stay in the current cell (line 15). Otherwise, the individual will move to a random cell that is empty (lines 18–26). Line 29 increases the age of the individual. Finally, line 30 decides if the individual has to leave the context (i.e. die).

The code for the method *die* is shown below. We retrieve the context (line 02) so that the agent can be removed from the context (line 03). A new agent is created using the same ID as the dead agent (line 04) and added to the context (line 05). This method shows that an agent can be removed and added dynamically during a simulation run.

```
01. public void die() {
02.   Context<Agent> context = (Context)ContextUtils.getContext(this);
03.   context.remove(this);
04.   Agent child = new Agent(this.id);
05.   context.add(child);
06. }
```

2.4 Building the model

In Repast, an ABS model is implemented as a *Context* created by *ContextBuilder* using the method *build*. When we implement the method *build*, we can take the following actions: (1) create, initialise and add agents to the model, (2) define the environment where the agents live and (3) read the parameters needed for the simulation. The following code shows how we build the context for the segregation model.

Line 01 shows how we define a class that implements *ContextBuilder*. In the implementation, we need to overwrite the method *build* as shown in line 02. Lines 03–06 show an example of how to read parameters for the simulation. In this example, we read the number of agents, the world height and the world width. The values of these parameters are assigned to the variables *numAgents*, *h* and *w*, respectively. Line 07 adds a grid projection to the context. Hence, we are defining a grid environment for the agents in the context. The grid has wrap-around borders and when an agent is added to the grid, the agent will be put in a random cell that is empty (because the third parameter is false, one cell cannot accommodate more than one agent). The loop from line 08 to 11 adds a number of agents as specified in the simulation parameters. An agent is created from the agent type in line 09. The new agent is added to the context in line 10. Finally, the method *build* needs to return the model (i.e. the context) as shown in line 12. At this stage, we have a complete ABS model. Of course, we still need to specify how the agents and their environment will be displayed during runtime. Repast provides a wizard to define this, as explained in Repast's 'getting started' document (it comes with the software). This example shows the typical actions that we take inside the method *build*, i.e. reading simulation parameters and adding the agents and environment to the model. It should be noted that we can add and remove agents from their context dynamically during a simulation. Hence, the location of the code to add agents is not limited to this method. A context in Repast has other features that may be useful for a more complex model. For example, a context can have sub-contexts to form a hierarchy of contexts.

```
01. public class SchellingModel implements ContextBuilder<Object> {
02.   public Context<Object> build(Context<Object> context) {
03.     Parameters p = RunEnvironment.getInstance().getParameters();
04.     int numAgents = (Integer)p.getValue("initialNumAgents");
05.     int h = (Integer)p.getValue("worldHeight");
06.     int w = (Integer)p.getValue("worldWidth");
07.     GridFactoryFinder.createGridFactory(null).createGrid("Grid",
          context,
          new GridBuilderParameters<Object>(
             new WrapAroundBorders(), new RandomGridAdder<Object>(), false, w, h
          )
        );
08.     for(int i=0; i<numAgents; i++){
09.       Agent agent = new Agent("Agent-"+i);
```

```
10.       context.add(agent);
11.     }
12.     return context;
13.   }
14. }
```

2.5 How can ABS help in making better decisions?

Simulation-modelling paradigms such as DES and SD have been used to help people make better decisions. ABS can help people make better decisions in the same way as DES and SD. Macal and North (2010) provide a list of situations where ABS is useful. There is one situation in which ABS is particularly useful, i.e. when we need to explain how the interactions between independent decisions made by agents can lead to an observable pattern at the system level. For example, a policymaker may notice that there is some segregation in a city and try to explain why it happens. Based on her knowledge of the people living in the city, she provides a possible explanation. Her explanation is that if all the individuals in a city do not want to be a minority in their neighbourhood, the city will be segregated. In order to support this explanation, the above segregation model was built. When an experiment is done using the model, the model consistently generates segregation in the city. Hence, the model proves that her explanation is plausible. Of course, there might be other explanations. This is where empirical work is needed to test which explanation is better.

The above reasoning method is called *abduction*. We start with a result Q (e.g. there is segregation in my city) and we construct an explanation or rule $P \rightarrow Q$ (if all individuals in a city do not want to be in a minority in their neighbourhood, the city will be segregated). An ABS model is developed to implement the rule and executed to see if the rule ($P \rightarrow Q$) can generate the result (Q). If it is true, then we can conclude that the case (P) is one possible explanation. This reasoning is not as strong as induction and deduction. However, this method is common when there is limited information available to us.

3 CHALLENGES FOR ABS AS A DECISION SUPPORT TOOL

This tutorial shows that ABS can be used as a decision-support tool. Despite its benefits, we can see that there are challenges that may hinder the use of ABS as a decision-support tool. I conclude this tutorial by highlighting three such challenges, namely, a good conceptual model representation, suitable validation methods and the need for participatory modelling.

3.1 Agent-based model representation

In simulation modelling, we select a certain portion of the real world system to be simulated for specific objectives. The process of capturing the essential elements of the system is referred to as conceptual modelling and the resulting model is referred to as a simulation conceptual model (Pidd 2004). In a typical simulation-modelling process, the conceptual-modelling stage is followed by computer implementation, validation, experimentation and implementation on a real system (Pidd 2004). There is no consensus in the literature on the definition of a conceptual model in simulation. Robinson (2008) discusses and compares a number of different definitions. At some stage in a simulation project, the conceptual model needs to be communicated to project stakeholders. Hence, the role of conceptual-model representation is crucial because it facilitates effective communication between stakeholders. Nance (1994) refers to conceptual-model representation for this purpose as a communicative model. Stakeholders may have different expertise in their own field but not in another one. Hence, communicating computer code for a model may not be a good idea if some stakeholders have limited or no computer-programming knowledge. For example, someone who has limited programming knowledge may find it difficult to understand the code presented in Section 2. Hence, a good conceptual-model representation for ABS is needed to help the communication between different stakeholders in a simulation project. Effective communication is essential for the success of a simulation project (Robinson and Pidd 1998).

The three commonly used simulation-modelling paradigms in operational research and management applications are DES, ABS and SD. While the conceptual-model representations in DES and SD have been

dominated by process-flow and stock-and-flow diagrams, respectively, research into conceptual-model representation in ABS is relatively new. Onggo (2013) discusses and compares five conceptual model-representation methods for ABS (i.e. pseudocode/flowchart, Petri Nets, DEVS, UML and BPMN). Although each representation method has its strengths and weaknesses, he argues that BPMN is more friendly for stakeholders who may not be familiar with computer programming or software-engineering concepts. Examples of how BPMN have been used as a communicative model for ABS can be found in Onggo and Karpat (2011), Onggo (2012) and Onggo et al. (2013). However, there remain questions that have not been addressed as to how effectively to represent an ABS model using BPMN. For example, the ability of BPMN to represent the complex attributes of an agent (such as memory, beliefs and perceptions), environment (spatial and relational) and complex learning behaviour (which may alter the rules/ behaviour) is limited. Furthermore, work on the specification language needed to run an ABS model represented using BPMN is also limited. There is some early work by Onggo and Karpat (2011) in which an ABS model can be written using BPMN and a detailed specification can be added to the model using an agent-based simulation language (ABSL). The combination of BPMN and ABSL facilitates the simulation of an ABS model.

3.2 Validation

Stakeholders will need assurance that the ABS model that is used as a decision-support tool is valid before they make a decision based on the model. DES and ABS are typically used to represent stochastic dynamic systems and there is a need to track individuals' states in the simulation. Given this similarity, the validation techniques commonly used in DES, such as face validity, operational validity, white-box validation and black-box validation, are also suitable for ABS. However, model validation in ABS is especially demanding. First, there is a need to validate ABS models at various levels (agent level, system level and, possibly, some intermediate levels). A second challenge arises due to the fact that we often need to represent behaviour in ABS using rules or algorithms. Hence, we need to check if the rules used in our ABS model represent the rules used by real-world agents. The heterogeneity of the agents makes this issue even more challenging. Finally, an ABS model often requires high-fidelity data, which are not always available. Hence, validation using empirical data may not always be possible. Given these challenges, research that creates validation techniques and tools for ABS (e.g. Duong 2010, Niazzi et al. 2009, Collier and Ozik 2013, Gurcan et al. 2013, Onggo et al. 2014, Onggo and Karatas 2015) is needed. These tools can help to increase the confidence of stakeholders when using ABS for decision-making. More research is needed in this area.

3.3 Facilitated ABS Modelling

A simulation project that involves many stakeholders with differing views and objectives is challenging. A good model representation can help to improve the communication between stakeholders. However, in the case of differences in stakeholders' perspectives, the modelling process also needs to be designed to deal with the conflicting interests of multiple stakeholders. One possible approach is the group model-building that has been widely used in SD (Vennix 1999). The objective of group model-building is to build a socially-constructed description of an organization based on an understanding of multiple stakeholders (Andersen et al. 2007). There has been similar work in DES recently, e.g. facilitated modelling using DES (Tako and Kotiadis 2015) and SimLean (Robinson et al. 2012). As far as I know, a similar approach for ABS is still in the very early stages, e.g. in Newig et al. (2008). Facilitated modelling can help modellers to engage stakeholders during a simulation project and manage potentially conflicting interests from multiple stakeholders. Hence, research in this area will provide us with tools that can improve the involvement of stakeholders in the ABS model-building process and improve the credibility of the ABS model.

REFERENCES

Andersen D F, Vennix J A M, Richardson G P, and Rouwette E A J A (2007). Group model building: problem structuring, policy simulation and decision support. *Journal of the Operational Research Society*, **58 (5)**: 691–694.

Bonabeau E (2002). Agent-based modeling: Methods and techniques for simulating human systems. *Proceedings of the National Academy of Sciences of the United States of America* **99 (suppl. 3)**: 7280–7287.

Chan V W K, Son Y-J, and Macal C M (2010). Agent-based simulation tutorial – Simulation of emergent behavior and differences between agent-based simulation and discrete-event simulation. *Proceedings of the Winter Simulation Conference*, pp. 135–150.

Collier N, and Ozik J (2013). Test-driven agent-based simulation development. *Proceedings of the Winter Simulation Conference*, pp. 1551–1559.

Duong D (2010). Verification, validation, and accreditation (VV&A) of social simulations. *Proceedings of the 2010 Spring Simulation Interoperability Workshop*, 9 pages.

Gurcan O, Dikenelli O, and Bernon C (2013). A generic testing framework for agent-based simulation models. *Journal of Simulation* **7**: 183–201.

Kiesling E, Günther M, Stummer C, and Wakolbinger L M (2012). Agent-based simulation of innovation diffusion: A review. *Central European Journal of Operations Research* **20**: 183–230.

Macal C M, and North M J (2007). Agent-based modeling and simulation: Desktop ABMS. *Proceedings of the Winter Simulation Conference*, pp 1551–1559.

Macal C M, and North M J (2010). Tutorial on agent-based modelling and simulation. *Journal of Simulation* **4**: 151–162.

Nance R E (1994). The conical methodology and the evolution of simulation model development. *Annals of Operations Research* **53**: 1–45.

Newig J, Gaube V, Berkhoff K, Kaldrack K, Kastens B, Lutz J, Schlußmeier B, Adensam H and Haberl H (2008). The role of formalisation, participation and context in the success of public involvement mechanisms in resource management. *Systemic Practice and Action Research* **21**, 423–441.

Niazzi M A, Hussain A, and Kolberg M (2009). Verification & validation of agent-based simulations using the VOMAS (Virtual Overlay Multi-Agent System) approach. *Proceedings of the Second Multi-Agent Logics, Languages, and Organisations Federated Workshops*, pp 494–501.

North M J, Collier N T, Ozik J, Tatara E R, Macal C M, Bragen M, and Sydelko P (2013). Complex Adaptive Systems Modeling with Repast Simphony. *Complex Adaptive Systems Modeling* **1(1)**: 3.

Onggo B S S, and Karpat O (2011) Agent-based conceptual model representation using BPMN. *Proceedings of the Winter Simulation Conference*, pp. 671–682.

Onggo B S S (2012). BPMN pattern for agent-based simulation model representation. *Proceedings of the Winter Simulation Conference*, 10 pages.

Onggo B S S (2013). Agent-based simulation model representation using BPMN. In: Pau Fonseca I C (ed.) *Formal languages for computer simulation: Transdisciplinary models and applications*. IGI Global: Hershey, PA, USA.

Onggo B S S, Indriany C, and Gunal M (2014). Test-driven simulation modelling. *Proceedings of the International Workshop on Applied Modeling and Simulation*, pp. 43–48.

Onggo B S S, and Karatas M (2015). Agent-based model of maritime search operations: A validation using test-driven simulation modelling. *Proceedings of the Winter Simulation Conference*.

Peidro D, Mula J, Poler R, and Lario F-C (2009). Quantitative models for supply chain planning under uncertainty: A review. *International Journal Advanced Manufacturing Technology* **43**: 400–420.

Pidd M (2004). *Computer simulation in management science*, 5th edition. John Wiley and Sons: Chichester, England.

Robinson S, and Pidd M (1998). Provider and customer expectations of successful simulation projects. *Journal of the Operational Research Society* **49 (3)**: 200–209.

Robinson S (2008). Conceptual modelling for simulation part I: Definition and requirements. *Journal of the Operational Research Society* **59 (3)**: 278–290.

Robinson S, Radnor Z J, Burgess N, and Worthington C (2012). SimLean: Utilising simulation in the implementation of lean in healthcare. *European Journal of Operational Research* **219 (1)**: 188–197.

Schelling T (1978). *Micromotives and macrobehavior*. New York: Norton.

Sensfuß F, Ragwitz M, Genoese M, and Möst D (2007). Agent-based simulation of electricity markets: A literature review. Working paper sustainability and innovation, No. S5/2007, available from http://nbn-resolving.de/urn:nbn:de:0011-n-661574 {accessed 25/11/2015}

Tako A A, and Kotiadis K (2015). PartiSim: A multi-methodology framework to support facilitated simulation modelling in healthcare. *European Journal of Operational Research* **244 (2)**: 555–564.

Terna P (1998). Simulation tools for social scientists: Building agent based models with SWARM. *Journal of Artificial Societies and Social Simulation* **1 (2)** available from http://jasss.soc.surrey.ac.uk/1/2/4.html {accessed 25/11/2015}

Vennix J A (1999). Group model-building: Tackling messy problems. *System Dynamics Review* **15**: 379–401.

Zhou S P, Chen D, Chai W T, Juo L B, Low M Y H, and Tian F (2010). Crowd modeling and simulation technologies. *ACM Transactions on Modeling and Computer Simulation* 20 (4), Article 20, 35 pages.

AUTHOR BIOGRAPHY

BHAKTI STEPHAN ONGGO is a Lecturer (Assistant Professor) in the Department of Management Science at the Lancaster University Management School (LUMS), Lancaster, United Kingdom. He completed his PhD in Computer Science at the National University of Singapore and his MSc in Management Science is from Lancaster University. His research interests are in the areas of simulation methodology (conceptual modelling, discrete-event simulation, system dynamics and agent-based simulation) and simulation applications in business and management. His email address is s.onggo@lancaster.ac.uk.