European Quarterly of Political Attitudes and Mentalities EQPAM
Volume 5, No.4, October 2016
ISSN 2285 – 4916
ISSN-L 2285 - 4916

Towards Agent-Based Simulation of Emerging and Large-Scale Social Networks. Examples of the Migrant Crisis and MMORPGs

Bogdan Okreša Đurić, Igor Tomičić and **Markus Schatten**
Artificial Intelligence Laboratory
Faculty of Organization and Informatics
University of Zagreb
Croatia

Date of submission: October 19th, 2016 *Date of acceptance: October 23rd, 2016*

Abstract

Large-scale agent based simulation of social networks is described in the context of the migrant crisis in Syria and the EU as well as massively multi-player on-line role playing games (MMORPG). The **recipeWorld** system by Terna and Fontana is proposed as a possible solution to simulating large-scale social networks. The initial system has been re-implemented using the Smart Python multi-Agent Development Environment (SPADE) and **Pyinteractive** was used for visualization. We present initial models of simulation that we plan to develop further in future studies. Thus this paper is research in progress that will hopefully establish a novel agent-based modelling system in the context of the **ModelMMORPG** project.

Corresponding Author: Dr. Markus Schatten, Assistant Professor, Head of the Artificial Intelligence Laboratory
Affiliation: Faculty of Organization and Informatics, University of Zagreb
Address: Pavlinska 2, 42000 Varaždin, Croatia
e-mail: schatten.markus@gmail.com

European Quarterly of Political Attitudes and Mentalities - EQPAM, Volume 5, Issue No.4, October 2016, pp. 1-19.
ISSN 2285 – 4916
ISSN–L 2285 – 4916

EQPAM Volume 5 No.4 October 2016
ISSN 2285 – 4916
ISSN-L 2285 - 4916

1. Introducere

Modern computing systems, along with the available software that follows hardware developments, represent a basis and prospects for analyses that were not available even a decade ago. Simulations of human or artificial agent behavior is considered a rather complex task, partly since quality data is needed for functional and significant simulation results, and partly because a complex simulation requires an advanced computer system that is computationally capable of conducting the said simulation. Furthermore, increasingly complex and turbulent environments usually necessary for good simulations make it even harder to conduct simulations, which is one of the simulation-related features covered by using modern computing systems. Simulations that can be described using a complex and turbulent environment may successfully be conducted using decentralized systems, since those would lower the workload of any of the included specific hardware systems. Furthermore, simulations of usually human behavior benefit greatly from intelligent, or at least interactive, agents.

Such systems conform to the basic description of multi-agent systems (MAS). MAS are systems comprising individual agents that are ideally independent and intelligent, and various elements of their environment. In their basic form, agents are the individual units in the viewed system, an abstraction of human beings (or other systems as needed) in the real world. Intelligent autonomous agents are pieces of software that can perceive their environment using sensors, and act upon that same environment using actuators. Furthermore, intelligent agents can reason and act based on the data gathered by their sensors. Agents can interact with each other, e.g. in form of communication using message exchange protocols.

A concept similar to MAS is agent-based modelling (ABM). While MAS is concerned more with definitions of agents, and their individual modelling, ABM is a more system-wide approach, concerned with modeling behavior of the whole system, and observing the emerging behavior of the agents, and of the whole system. A special type of MAS is represented by the large-scale multi-agent systems (LSMAS) (Schatten, Ševa and Tomičić, 2016). Their main feature, which differentiates them from MAS, is the included quantity of agents. Higher numbers of included agents make LSMAS more complex, therefore better planning may be needed along with more resources for their execution. Even more so than MAS, LSMAS can be described as distributed, autonomous, and intelligent, thus being well-suited as an abstracted model for various large-scale application domains, like the Internet of Everything (IoE), smart cities, smart transportation infrastructure, smart living solutions, complex cyber-physical systems, and even various massively multi-player on-line games (MMOGs).

One of the environments for developing MAS is Smart Python multi-Agent Development Environment (SPADE). SPADE[1] is a MAS platform that allows users to create instances of specific classes representing agents and various kinds of behaviors, emphasizing behavior of agents and their communication using messages. Every SPADE agent's definition is based on an existing basic SPADE agent class that uses a function representing agent's initial behavior, and customized classes representing behaviors of the given agent.

An interesting result of MAS simulations are networks that emerge based on agents' interaction with their environment and with each other. The emerged network can be further analyzed using social network analysis (SNA) techniques that measure for example how important a certain element of the network is for the network as a whole, or for their immediate environment, or how important a certain agent is for information propagation through the given network. SNA elements will be used along with simple visualization techniques in this paper to dynamically represent the emerging network of the simulated systems.

[1] Further details available at https://pypi.python.org/pypi/SPADE

ISSN 2285 – 4916
ISSN-L 2285 - 4916

An interesting simulation model that uses MAS was developed by Fontana and Terna [2015], named **recipeWorld**. Herein we have used SPADE to re-implement **recipeWorld** in order to allow not only simulation, but also implementation of LSMAS that are connected to the real world. Our basic presumption is that LSMAS are too complex to be handled without constant influx of (big) real-time data and simulation using such data to be ahead of complex emerging situations (see Schatten, Ševa and Okreša-Đurić, 2015a, b; Voinea and Schatten, 2015) for an introduction to big data and web mining). Herein we will especially focus on the emerging social networks, as they are described by Fontana and Terna.

Apart from the re-implementation of **recipeWorld** using SPADE and **Pyinteractive** herein we will present two conceptual models of how real-time large-scale systems might be combined with profound agent-based simulation systems like **recipeWorld**. The first example will deal with MMOGs whilst the other shall deal with the contemporary migrant crisis in Syria and the EU. Both examples have a number of commonalities: (1) they are large-scale, e.g. there are a great number of agents that act upon the environment, (2) they are more or less real-time with huge amounts of big-data influx (server logs, chat data etc. for MMOGs; security cameras, social network applications, border personnel etc. for the migrant crisis), (3) they are time-critical, e.g. decisions have to be made in real-time, and (4) they constitute large-scale social networks that emerge due to interactions of agents which can have large impact on the overall behavior of the system.

The **recipeWorld** model will be described in some more detail in *Section 1.1. Section 1.2* will describe in more detail massively multi-player online role-playing games, a specific type of MMOGs, and *Section 1.3* will deliver basic details on immigration scenarios. Visualization technique used in this paper will be detailed in *Section 2.* Two specific proposed simulation scenarios that can be conducted using **recipeWorld** model are described in *Section 3:* one pertaining to the concept of MMORPGs (*Section 3.1),* and the other one concerned with immigration issues (*Section 3.2).*

1.1 The recipeWorld

As mentioned earlier, an interesting agent-based method was first described by Fontana and Terna in (Fontana and Terna, 2015) comprising two basic agent types: Factories and Orders. One of the main differentiating features of the mentioned authors' model is the inclusion of both ABM and network analysis, i.e. SNA, in the form of **recipeWorld**, which is an agent-based model that "*simulates the emergence of a network out of a decentralized autonomous interaction*". It is argued in (Fontana and Terna, 2015) that combination of methods of ABM and those of network analysis can "*increase enormously the potential of complexity-based policies*". Furthermore, it is stated that, when faced with dynamic analysis, network analysis is limited, e.g. in dynamics (focus of network analysis is on static networks, while ABM or MAS produce dynamic systems), behavior of nodes, and methods of traditional mathematical modelling of networks.

The **recipeWorld**, as "*an agent-based model that simulates the emergence of networks out of a decentralized autonomous interaction*" (Fontana and Terna, 2015), consists of three foundational elements:

- recipes - steps needed to be taken in order to achieve a certain end, vary in number;
- orders - objects that represent specific end to be pursued, and contain technical information, as well as basic data about the order instance;
- agents - problem-solving cores that can be active or inactive, and are able to perform some of the steps needed to complete a given recipe.

EQPAM Volume 5 No.4 October 2016
ISSN 2285 – 4916
ISSN-L 2285 - 4916

Recipes are basically immutable series of codes representing various order components that demand certain services in order to be fulfilled. For example, the following is a representation of a recipe: [3 2 5 1 4] that demands access to the services as follows: first, service 3 has to be performed, followed by service coded 2, and so on until the element that needs access to service number 4. Once all the elements of an order have been dealt with by their corresponding services offered by other elements of the system, the order is finished, i.e. the pursued end is met. **Figure 1** shows basic elements of the **recipeWorld** model: two orders (green) named by their ID numbers (7107 and 8714), with each having a single recipe of length two ([1, 2] and [2, 1] respectively), and two servicing agents (*Factory59498* and *Factory57780*) serving up to two services each.

What is novel and interesting about the approach of **recipeWorld** is that a social network is not initially defined, but emerges as a by-product from social processes that take place between the various agents of interest. Recipes are actually process descriptions that might be modelled using some well-known abstractions like process algebra.

Following the basic presumptions of **recipeWorld**, it is possible to adjust the model to many application domains. Two such applications will be illustrated in *Section 3* of this paper. Furthermore, **recipeWorld** sets an interesting environment for visual representation of temporal changes of the system. Coupled with the methods of SNA, the visualization can be informative in a way that delivers information to the intended reader much clearly than if they were presented with a series of numbers. Furthermore, it is often easier to read and interpret visually presented data, as opposed to alpha-numeric presentation.

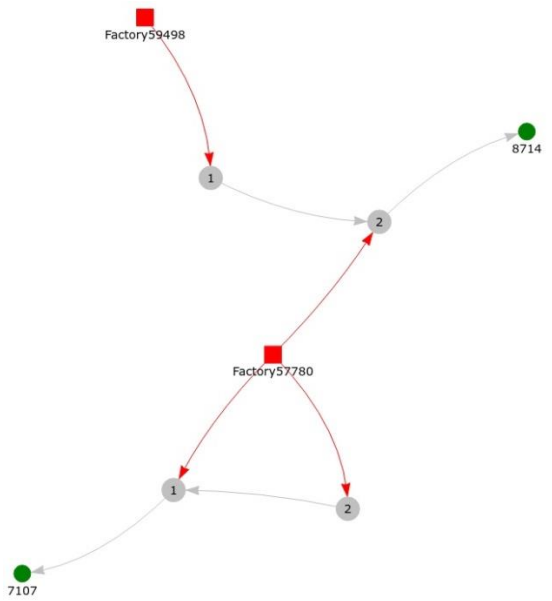

Figure 1.
Some recipeWorld elements: orders (green), recipe elements (gray), and service agents (red)

1.2 MMORPG

Computer games are a concept worthy of research in the online era, although rather stigmatized as being socially retardant and childish. It is possible to classify computer games in two broad classes: games played online (using the Internet), and those played offline. Some online games usually include great numbers of players that play simultaneously thereby engaging in various mutual interactions, and are thus

ISSN 2285 – 4916
ISSN-L 2285 - 4916

called massively multiplayer online games (MMOGs). The massiveness of a game is most notably observable in players' interaction which can be either real-time (e.g. private chat and instant messages), or based on time or interaction intervals (e.g. turn-based games, and in-game interaction). While there are numerous MMOGs, herein we will concentrate on massively multi-player online role-playing games (MMORPGs). It has become usual to consider the concepts of MMOG and MMORPG as synonyms throughout the Internet, whilst there are considerable differences.

MMOGs have had a significant boost in popularity in the last two decades, as their numbers about year 2015 could be measured in millions of monthly active users. Along with a numerous user-base, MMOGs are generating a significant financial influence as well [14]. Having observed the immense popularity of MMOGs, some other aspects arise, which are interesting research-wise: MMORPGs allow us to study two aspects of large-scale computing - social interaction of (usually large numbers of) players, as well as the design and implementation of large-scale distributed artificial intelligence (Schatten, Tomičić and Okreša-Đurić, 2015). The former is mostly in the domain of ABM (social science perspective), whilst the latter is studied by MAS (computer science perspective).

The MMORPGs have many different subgenres, but they all have similar basic mechanics: a protagonist placed in a world where they have to interact with NPCs (non-playing characters) and mobs (enemies, monsters, etc.) which have to be beaten or give them tasks (quests) that they have to solve to be able to buy better equipment, learn new skills (e.g. magic, fighting, etc.), or proceed to higher player levels (Schatten and Okreša-Đurić, 2016).

Two examples of MMOGs will be presented here that show the power of MMOGs and how interesting it may be to investigate in-game behavior of players and their actions. The first example is from an MMORPG called *World of Warcraft*[2] played by about four million monthly active players. The example describes how an (artificial) infection was spread through the virtual world, and the chaos caused by the said event (Lofgren and Fefferman, 2007). Furthermore, the described example may showcase how valuable MMORPGs may be in simulating and researching community behavior, with many of the numerous behavior patterns that can be met in the real world. One of the update patches presented in year 2005 included a special area of the virtual world meant for highly developed player avatars (player characters in the game) which advanced far in the game story. The area included an area boss mob of high level with one of his abilities being casting an infection upon the attacking players. The infection was intended to weaken an avatar, but it could spread. It was later noticed that there were two ways of spreading the infection: teleporting the avatar to a safe place (e.g. town in the virtual world), and by using pets (special characters designed to help avatars in some elements of the game), since pets could be summoned and unsummoned, but once infected unsummoning did not cure them. Upon teleporting to e.g. a town, the infected avatar spread the infection to surrounding avatars. The problem here is that the infection was lethal to low-level avatars. The deadly infection spread throughout the virtual world, and reportedly caused some major problems. Upon failing to introduce quarantine zones in the game, and try to end the infection in a "natural" way, the administrators and developers decided to restart the server, and the infection was gone.

The second example covers a completely different game, but of the same MMORPG genre. It shows how computer games can be a place where human organizations and ideas can be trained and tested. *EVE Online*[3] is situated in Space, and engages about 500000 monthly active players throughout the 13 years it exists, thus building a solid and complex galactic civilization in the virtual universe. The

[2] https://worldofwarcraft.com/
[3] https://www.eveonline.com/

EQPAM Volume 5 No.4 October 2016
ISSN 2285 – 4916
ISSN-L 2285 - 4916

players trade resources, fight, build, create, destroy, explore, and rule. The government within the game is made by the players, and is based on their interaction, their elections, schemes, and organizations. The long history of the game is thus rich with stories of political turmoil, great space wars, actions of enormous consequences and value, and construction and damage that could be calculated in large sums of real money.

1.3 Immigration Scenarios

Immigration is studied across various fields such as demography, political science, anthropology, economics, psychology and sociology (Berry, 2001), and its complex dynamic processes are also scientifically curious for computational modelling and simulations.

Analyzing immigration scenarios might prove useful in dealing with preferential settling factors and visualizing specific geographic location loads and migration dynamics, leading to a more holistic decision-making environment.

Modelling in such context could be based on any immigration scenario, and the European refugee crisis might provide a challenging setting for simulating immigration dynamics based on its vast numbers; according to (Banulescu-Bogdan and Fratzke, 2015), more than 487,000 people arrived on Europe's Mediterranean shores in the first nine months of 2015, the year in which the European refugee crisis began (BBC News, 2016).

There are estimates that predict about 1 million refugees yet to come, and European Commission proclaims this as the "*largest global humanitarian crisis*" of our time (Humanitarian Aid and Civil Protection, European Commission, Echo fact- sheet: Syria crisis, 2015).

The model proposed in *Section 3* is highly dependent on the motivational and decision making factors of the immigrants, and the scalability of the data-acquiring process in this context presents the greatest challenge in a successful and realistic modelling of the preferential settling simulations. Data aggregation and generalization might prove as useful tools when dealing with individual choices becomes an unfeasible process for model implementation.

1. Visualization Mechanics

It is arguably easier for humans to interpret information when presented with visualization, rather than when faced with raw data in alpha-numeric (Patterson, Blaha, Grinstein, Liggett, Kaveney, Sheldon, Havig, and Moore, 2014).

When focusing on behavioral or temporal in general, characteristics of agents of a simulated system, it is useful to observe changes and interaction in the system with the added temporal component, rather than analyzing static chosen moments in the observed period of time. In order to visualize changes within the simulated system, and interaction of the included agents, this paper uses a specific Python programming language library Pynteractive,"*aimed to create interactive visualizations provided by several HTML5 JavaScript libraries like vis.js, d3.js and many more."*[4]

The three modes offered by **Pynteractive** are dealing with graphs, charts, and maps. The charts option is right for visualizing the network that is expected to emerge from the observed interaction in the simulated system based on **recipeWorld**.

The commands used by the library are straightforward, and are shown in **Listing 1.**

[4] https://github.com/coelias/Pynteractive

European Quarterly of Political Attitudes and Mentalities EQPAM
Volume 5, No.4, October 2016
ISSN 2285 – 4916
ISSN-L 2285 - 4916

```
1  from pynteractive import *
2
3  N = Graph(directed=True)
4
5  N.view()
6
7  N.addNode('first')
8  N.addNode('second')
9
10 N.addEdge('first', 'second')
11
12 N.delNode('first')
```

Listing 1.
Basic commands of Pynteractive for graphs.

Inclusion of the **Pynteractive** library is conducted using line 1 of the listing. Line 3 initializes the graph, meaning that everything further referenced using the uppercase letter N will be considered a graph. Line 5 shows that graph in a web browser. The opened view is empty, since the graph is empty in the beginning (**Figure 2**).

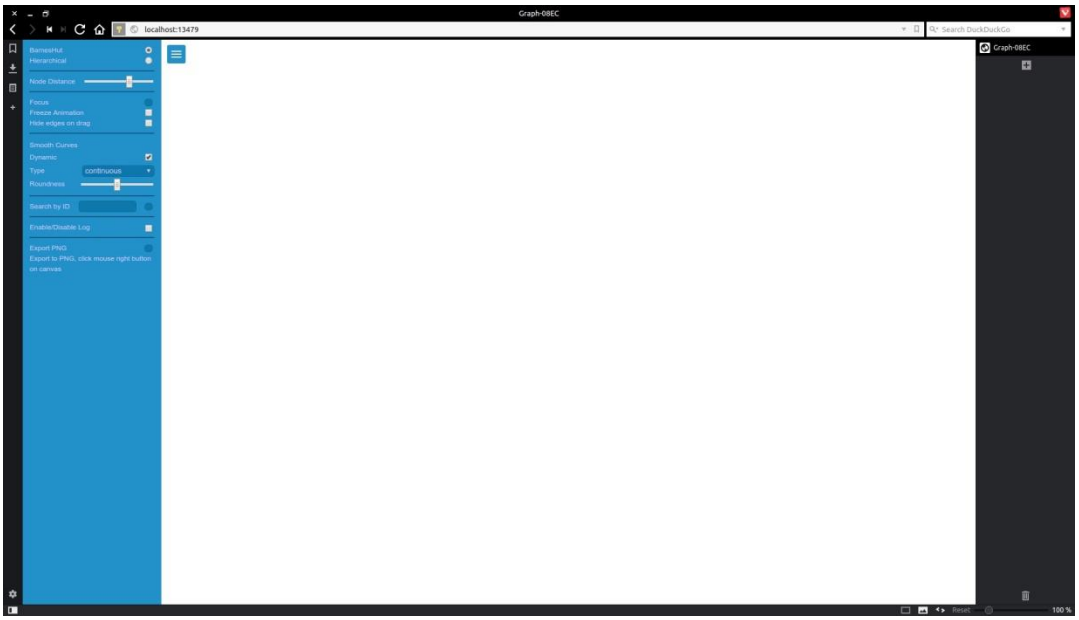

Figure 2.
Clean starting graph in Pynteractive

The next step is adding some nodes in the graph. Two nodes are added in lines 7 and 8, one in each (**Figure 3** and **4**, respectively).

Nodes are added using the *add.Node()* function which can be given the name of the created node, e.g. 'first'. After two nodes (vertexes) exist in the simple graph, an edge should be added between those nodes (**Figure 5).**

EQPAM Volume 5 No.4 October 2016
ISSN 2285 – 4916
ISSN-L 2285 - 4916

Function *addEdge()* shown in line 10 of the listing must have two nodes specified using their defined names, or identifying numbers.

Pynteractive allows for dynamic node and edge creation, modification and deletion. These real-time mechanisms make it possible to visualize temporal component of the simulated system. Therefore creation of nodes, their interaction with each other, and their moving throughout the system, can be visualized in real time.

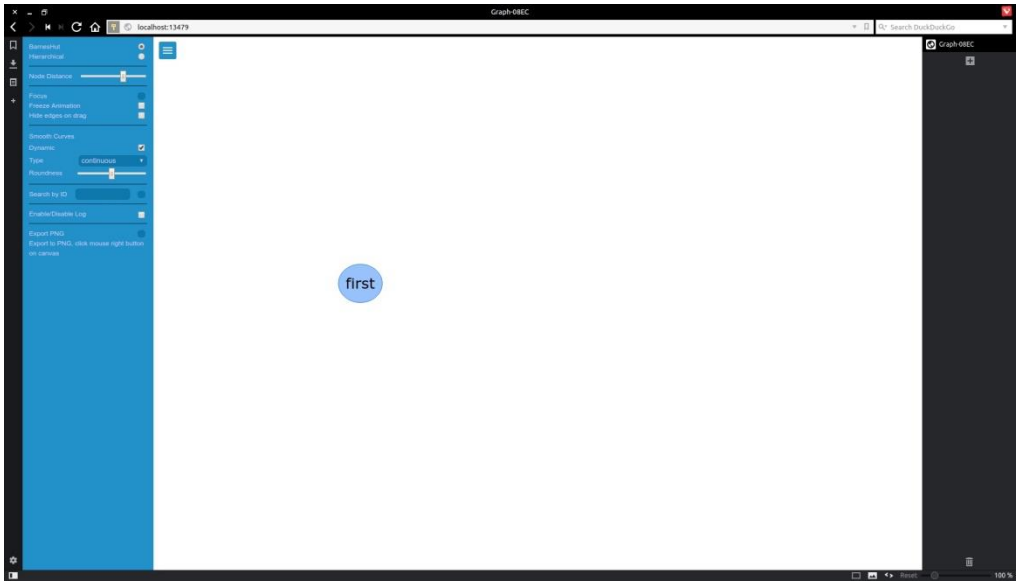

Figure 3.
New node 'first' added to the graph in Pynteractive

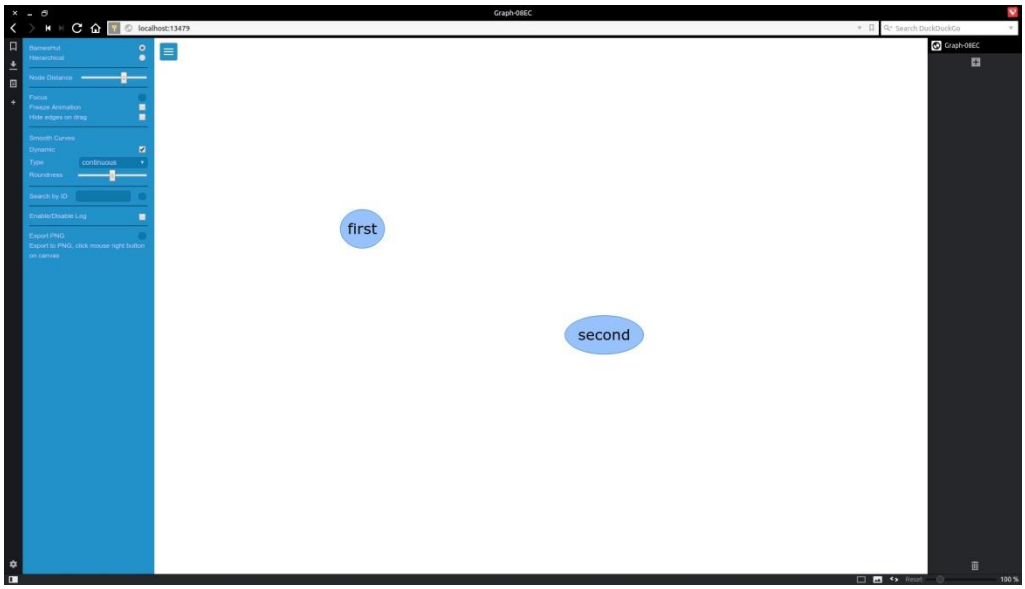

Figure 4.
Node 'second' added to the graph in Pynteractive

European Quarterly of Political Attitudes and Mentalities EQPAM
Volume 5, No.4, October 2016
ISSN 2285 – 4916
ISSN-L 2285 - 4916

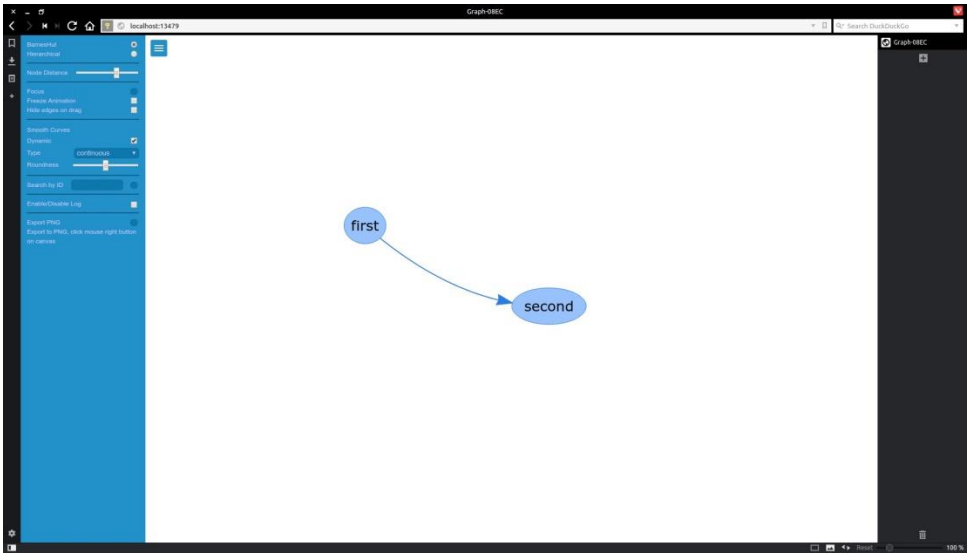

Figure 5.
Edge added from node 'first' to node 'second' in Pynteractive

Figure 6.
Deleted node 'first' in Pynteractive

The edge created in line 10 will be created from node named 'first' towards the node named 'second'. In the end, function *delNode* in line 13 will delete the node existing in the graph. The function will delete the node by the given name, e.g. 'first' (**Figure 6).** Since this node was a part of a certain edge the edge is deleted as well.

In addition to rendering changes made to the given graph in real time, **Pynteractive** user interface can be customized to an extent. The sidebar can be populated by additional buttons that perform custom functions, e.g. adding nodes on click, changing color codes of the graph, calculating various statistical and SNA values, etc. Furthermore, dynamical changes make it possible to calculate SNA values (such as various centrality measures

EQPAM Volume 5 No.4 October 2016
ISSN 2285 – 4916
ISSN-L 2285 - 4916

(Newman, 2010)), and apply them visually as they change. E.g. resizing a node based on the number of neighboring nodes it is connected to (nodes connected by one edge only) makes it easier to visually identify more important nodes of the given network.

2. Conceptual Examples

The cases included in this paper show how **recipeWorld** could be used coupled with visualization techniques to simulate two specific situations. The situations presented here, and the conducted simulations, will serve as a showcase of the idea, without the details of true data, for such a simulation may be subject to a future research.

Visualization of a case will be described using the following example, which is adapted from (Fontana and Terna, 2015), and is dealing with factories and orders. Factories and orders shall be modelled as individual intelligent agents using SPADE environment. Order elements, i.e. the recipe, are modelled as a special attribute of the order agent - each order agent is acquainted with their respective recipe. Demand for services, i.e. recipe parts of an order agent, and supply of services, i.e. services offered by factory agents, are met in a mechanism similar to a market. Order agents browse the marketplace of the system, where all the available services are offered, until they find the right one for their current recipe element. Upon finding the right service, an order agent starts communication towards the factory agent providing the demanded service. On the other side of the market, factory agents supply the market with their services, marketing them towards the order agents. Offered services may be added, removed, or changed through time. Once an offered service is demanded, communication is started between the demanding and the offering agent.

Factory agent instances are created first (**Figure 7).** Each factory agent instance is given a set of services they offer to the system. These offered services are registered in the system's market that can be browsed by other agents in the system. After registering their services, factory agents wait until they receive a message from an order agent requesting production of one of their parts.

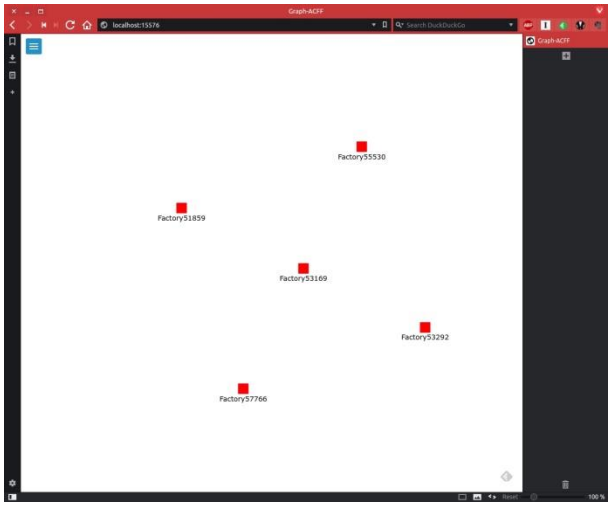

Figure 7.
Created factory agent nodes (red)

European Quarterly of Political Attitudes and Mentalities EQPAM
Volume 5, No.4, October 2016
ISSN 2285 – 4916
ISSN-L 2285 - 4916

Upon receiving a message from an order agent, each factory agent can react in two different ways, either commencing production, or denying service to the requesting order agent, on the grounds of producing a part for another order agent (blue node in **Figure 8).** Production process of a factory agent is represented by creating an edge between the factory agent and the corresponding part belonging to an order agent (**Figure 8),** all three represented as nodes in the emerging network. After the production is completed, the given factory is free to take another recipe part from another order agent, and the process is repeated.

Order agents appear on the scene after all the factory agents (**Figure 8).** Order agent instances are a bit more complex, having to look for available services and having a set of containing recipe parts.

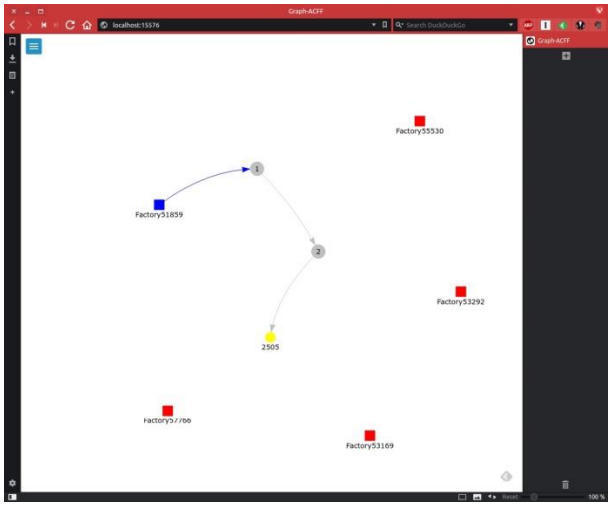

Figure 8.
Created order node (yellow) with 2-element recipe (gray), where one factory (blue) is already busy servicing an element

Each recipe part is added as a node to the growing graph, and is connected to the previously defined recipe part or the order agent node. This creation process ensures that every order agent node is connected to all of the relevant recipe part nodes in order which corresponds to their needed production order (**Figure 8).** Final state of a simple example is shown in **Figure 9.**

It was stated that agents communicate with each other. SPADE environment uses technologies that enable agents to communicate via a mechanism similar to popular online instant messaging. Message exchange is therefore the main means of communication in the specified system. Messages exchanged, along with their impact on the virtual lives of the agents of the given system, are shown in **Figure 10.**

According to their nature which is more complex than that of factory agents, order agents are modelled using finite state machine (FSM) agent behavior type, included as one of the possible agent behavior types in SPADE. FSM is applicable to modelling order agents since their behavior can be described as a number of distinct states that have particular conditions on how they change. The order agent, as described above, can be shown as a set of states illustrated in **Figure 11.**

Building from the conceptual elements and definitions above, SPADE pro- vides with elements that can be successfully combined into a working **recipeWorld** example. Factory agents can be implemented as shown in **Listing 2.** Factory agent is given two specific behaviors: *AnswerQuery* and *Produce*.

EQPAM Volume 5 No.4 October 2016
ISSN 2285 – 4916
ISSN-L 2285 - 4916

AnswerQuery behavior (line 2) contains some other code as well, but it is important that it sends a reply message (line 4) back to the order agent (line 5) stating that it will accept a production proposal (line 6).

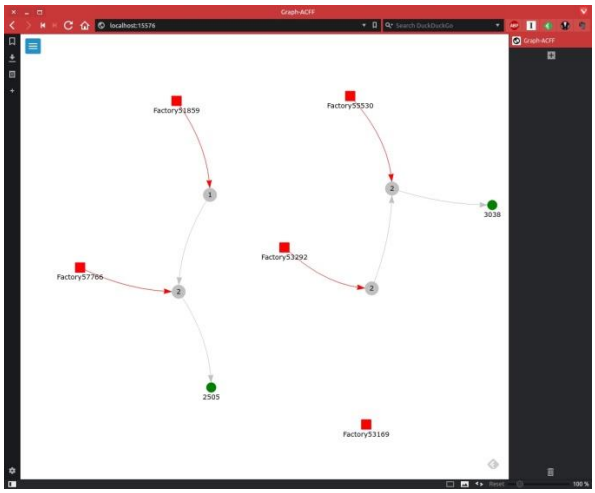

Figure 9.
Two serviced orders (green) with 2-element recipe each (gray)

The second behavior, Produce (line 8), produces the recipe part, i.e. services the recipe part of an order agent. The production process creates a new edge from the factory agent to the given recipe part node (line 9). Initial setup of the factory node is derived from the *_setup()* function, where the corresponding node is created in the given graph. Lastly, the mentioned behaviors are initiated in the given individual factory agent (lines 14-15).

```
1  class AgentFactory(spade.Agent.Agent):
2      class AnswerQuery(spade.Behaviour.EventBehaviour):
3          [...]
4          msg = spade.ACLMessage.ACLMessage()
5          msg.addReceiver(OrderAgent)
6          msg.setPerformative("accept-proposal")
7
8      class Produce(spade.Behaviour.EventBehaviour):
9          [...]
10         N.addEdge(me, recipePart)
11
12     def _setup(self):
13         [...]
14         N.addNode(factory)
15
16     [...]
17     self.addBehaviour(self.AnswerQuery())
18     self.addBehaviour(self.AnswerQuery())
```

Listing 2.
SPADE implementation summary for factory agent

European Quarterly of Political Attitudes and Mentalities EQPAM
Volume 5, No.4, October 2016
ISSN 2285 – 4916
ISSN-L 2285 - 4916

An order agent is of a structure that is more complex than that of a factory agent (**Listing 3**).

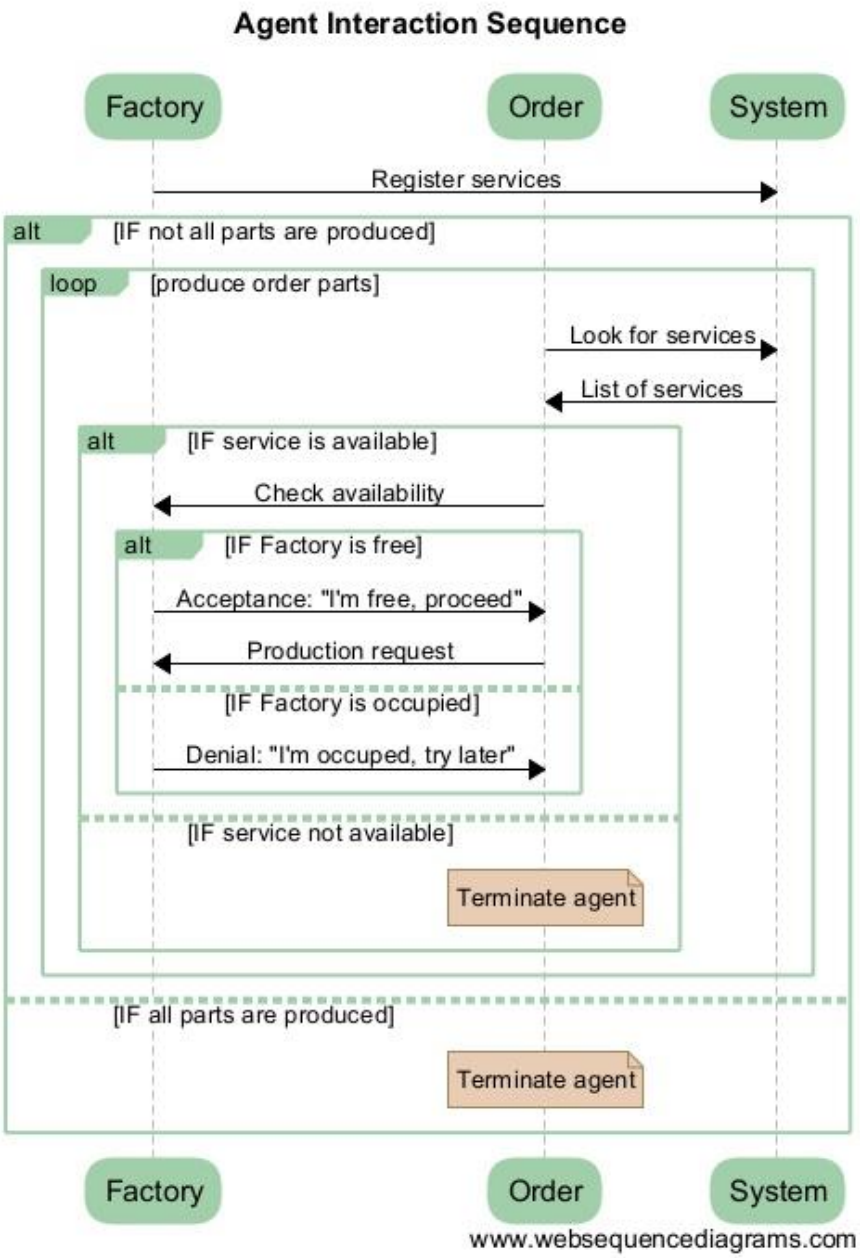

Figure 10.
Example sequence diagram of the recipeWorld agents

EQPAM Volume 5 No.4 October 2016
ISSN 2285 – 4916
ISSN-L 2285 - 4916

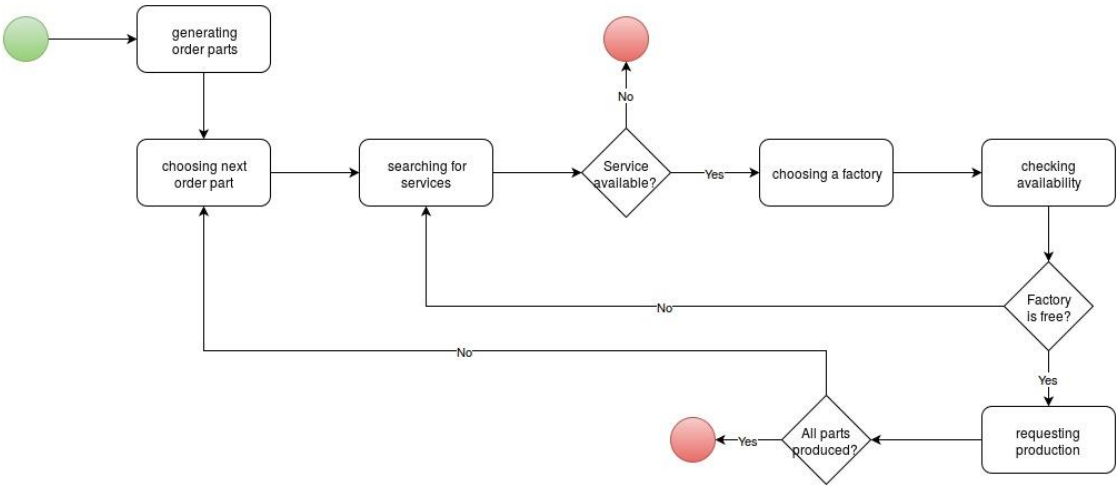

Figure 11.
Inner states of an order agent, and their flow from agent creation (green circle) to agent termination (red circle)

Initialization of an order agent is defined again by *_setup* function. During the initialization phase, an order agent gains its respective node in the graph (line 27), followed by visualization of its respective recipe parts. Each recipe part of the given order agent is visualized, and connected via an edge to the order agent (lines 28-30). Last element of the initialization phase is definition of the FSM conditions, detailed in *setFSM()* function, skipped in listing 3 for clarity. Order agent comprises five distinctive behaviors. The first behavior is used for browsing the system's market, and looking for the needed service (lines 4-5), namely *Service2* in listing 3. Upon determining which factory agents offer the needed service, behavior *CheckFactoryAvailability* is started, since the wanted factory agent may be taken, i.e. it may be producing another order's recipe part (lines 9-11). Since the reply has to be generated by the factory agent, order agent runs the *WaitForFactoryAnswer* behavior and calmly waits for the reply containing further instructions on what to do - whether to request start of production, or to look for another factory. If the chosen factory agent is not occupied, *StartProduction* behavior is launched and production of the specified recipe part is requested by the order agent (lines 18-20). Finally, when all the recipe parts of an order agent are produced, the last needed behavior (*FinishProduction*) is run, and the production process is finished, i.e. the goal of the given order agent is achieved.

ISSN 2285 – 4916
ISSN-L 2285 - 4916

```
1  class AgentOrder(spade.Agent.Agent):
2      class SearchForFactories(spade.Behaviour.OneShotBehaviour):
3          [...]
4          s = spade.DF.Service(name="Service2")
5          search = self.myAgent.searchService(s)
6
7      class CheckFactoryAvailability(spade.Behaviour.OneShotBehavic
8          [...]
9          msg = spade.ACLMessage.ACLMessage()
10         msg.addReceiver(FactoryAgent)
11         msg.setPerformative("propose")
12
13     class WaitForFactoryAnswer(spade.Behaviour.EventBehaviour):
14         [...]
15
16     class StartProduction(spade.Behaviour.OneShotBehaviour):
17         [...]
18         msg = spade.ACLMessage.ACLMessage()
19         msg.addReceiver(FactoryAgent)
20         msg.setPerformative("request")
21
22     class FinishProduction(spade.Behaviour.OneShotBehaviour):
23         [...]
24
25     def _setup(self):
26         [...]
27         N.addNode(order)
28         for y in range(0, numOfRecipePieces):
29             N.addNode(recipePiece)
30             edge = N.addEdge(recipePiece, self)
31
32     def setFSM(self):
33         [...]
```

Listing 3.
SPADE implementation summary for order agent

3.1 recipeWorld in MMORPGs

One is situated in a virtual world of an MMORPG. One of the driving forces in any game are tasks imposed upon the players, given by various NPCs throughout the game. Tasks serve as a way of driving the player in the direction of the main game scenario. Other, non-essential, tasks, are available to the player as well, and they usually provide the player with additional items, experience points (used to level up and advance the player's character), or additional options of the game. Tasks, therefore, contain several steps (objectives or goals) that need to be fulfilled for the task to be successfully finished. Such a structure reminds greatly of the **recipeWorld** concept of orders and recipes: an order is a task containing a goal that is pursued, and it is defined by a specific recipe specifying the task elements that need to be fulfilled for the pursued goal to be reached. For example, a specific task *Save thy Love* may be specified by the following steps:

EQPAM Volume 5 No.4 October 2016
ISSN 2285 – 4916
ISSN-L 2285 - 4916

[Look for Evermind Pick 10 Evermind Brew Healing Potion Heal thy Love].

As mentioned earlier, in *Section 1.1,* the third core element of the **recipeWorld** consists in the agents that can solve some of the elements of a recipe. Tasks usually demand the player to go to a specific place, kill a specific number of mobs, gather a specific number of items, talk to a specific NPC, find a specific item, etc. The mentioned events serve to fulfil a specific part of a task. Therefore, they can be considered as the mentioned agents that solve some of the elements of a recipe. Furthermore, should one consider a wider, more abstract, view of the mentioned agents, thus including geo-locations and mobs along with NPCs, it may be possible to simulate the task solving problem using the **recipeWorld**.

To come back to the example of the infection which spread among the virtual world and left the the administrators and developers of the game no other option as to restart the game server, it might be an interesting scenario to employ the approach presented herein. For example, since all data from a game are available to the administrator (albeit in the form of various game server logs) it might be possible to connect these data to a real-time simulation platform and experiment with possible solutions to the problem. As has been shown by (Pastor-Satorras and Vespignani, 2001) the configuration of a social network highly influences the speed of infection spreading. Thus, since the social network is an emerging feature of the social interaction among players, what needs to be done is "reconfigure" the social network to stop the infection. One well known method is to quarantine hubs (e.g. nodes in the network in which lots of interactions are taking place like airports, central bus stations etc.), but there are of course other solutions. Thus, the administrators would need to rewrite the quests given out by NPCs, as stated earlier, to avoid larger gatherings of players, and thus stop the infection from spreading. In the language of **recipeWorld**, the recipes (e.g. processes) of artificial players need to be changed in a way which will change the emergent social network, and thus stop the unwanted effect of infection.

3.2 recipeWorld and Immigration Scenarios

When modelling immigration scenarios, cities can be implemented as factory agents, providing defined services to orders. Multiple services from a single factory agent in this context are possible, and could be considered for a more realistic setting. Immigrants could be implemented as orders, having their motivational and decision-making factors implemented as the elements of the order, and thus pursuing their goals.

Possible elements in the preferential settling model are proposed as follows, both short-termed and long-termed:

- A possibility of entry
- Labour opportunities
- Social policies
- Organized admission shelters
- A possibility of long-termed settling and citizenship
- A freedom of movement
- A positive attitude of government towards the refugees and their culture
- A positive mindset of residents towards the refugees and their culture
- Cultural and/or religious similarities

European Quarterly of Political Attitudes and Mentalities EQPAM
Volume 5, No.4, October 2016
ISSN 2285 – 4916
ISSN-L 2285 - 4916

- Logistics
- Strong economy
- Adequate school system

These factors could be used to model immigrants' preferential settling choices, and are listed here for the purposes of simulation functionality testing only; they are not listed either systematically or holistically in the context of migration processes.

For example, an immigrant may be specified by the following elements:

[Cross border X Temp stay at admission shelter Move freely further Long-termed settling Find a job].

An individual, or a group of individuals, could be modelled in this manner, according to priorities and decision / motivational factors driving the immigration process.

By using this model, we propose that the following immigration issues could be tackled:

1. Fairness in the context of the quota system in distributing refugees among various countries. A fair distribution could be achieved by using the observable geographic migration load.
2. Decision making and inter-country negotiation processes.
3. Border patrol planning.
4. Relocation procedures planning; etc.

Visual presentation of the migration dynamics with focus on the individual geographical location loads could prove to be a relevant asset in tackling immigration issues. A modeler could implement real-time events that would disrupt the default migration dynamics; for example, a destination location with high migration load could become inaccessible due to sudden changes in country's policies, leading to real-time relocation processes based on the preferential settling parameters.

3. Conclusion

In this research in progress paper we have presented some of the activities per- formed by the Model MMORPG project that aims on establishing novel ABM techniques for both implementation and simulation of LSMAS. Herein we have presented the **recipeWorld** developed by Fontana and Terna as a possible solution for simulation of large-scale social networks. Their initial platform has been re-implemented using SPADE (to allow for not only simulation, but also implementation of LSMAS) and **Pyinteractive** (to allow for interactive visualization).

We have provided two conceptual examples of modelling based on MMORPGs and the current migrant crisis in the EU. We plan to extend and develop these examples further to see if they can act as an example of using ABM and MAS approaches together. Our main objective is to allow not only simulation of current (live) events, but also decision making support for potential decision makers confronted with large-scale systems steering. Such decision makers could for ex- ample be a MMORPG administrator or developer who wants to study what will happen if he introduces a new feature into the game, by simulating

EQPAM Volume 5 No.4 October 2016
ISSN 2285 – 4916
ISSN-L 2285 - 4916

it based on the current behavior of players. Another example might be a government officer who wants to introduce policies in order to better control migrant related situations based on live (big) data.

In the future, we plan to develop a fully functional modelling tool that will allow us to connect with various real-time services and (big) data streams that can be directly included into the simulation of a LSMAS.

Acknowledgements

This work has been supported in full by the Croatian Science Foundation under the project number 8537.

References

*** BBC News. Migrant crisis: Migration to Europe explained in seven charts. 2016.

*** Humanitarian Aid and Civil Protection, European Commission. Echo fact- sheet: Syria crisis. 2015.

Banulescu-Bogdan, N., and Fratzke, S. (2015) Europe's migration crisis in context: Why now and what next, *Migration Policy Institute's* website, September, 24, 2015.

Berry, J.W. (2001) A Psychology of Immigration, *Journal of social issues*, 57(3):615–631, 2001.

Fontana, M., and Terna, P. (2015) From Agent-based models to network analysis (and return): the policy-making perspective, *Working Paper Serie* 07, 2015.

Lofgren, E.T., and Fefferman, N.H. (2007) The untapped potential of virtual game worlds to shed light on real world epidemics. *Lancet Infectious Diseases*, 7(September):625–629, 2007.

Newman, M. (2010) Networks: An Introduction. Oxford University Press, New York, USA, 2010.

Pastor-Satorras, R., and A. Vespignani, A. (2001) Epidemic spreading in scale-free networks. *Physical review letters*, 86(14):3200, 2001.

Patterson, R.E., Blaha, L.M., Grinstein, G.G., Liggett, K.K., Kaveney, D.E., Sheldon, K.C., Havig, P.R., and Moore, J.A. (2014) A human cognition framework for information visualization, *Computers & Graphics*, 42:42–58, August 2014.

Schatten, M., and Okreša-Đurić, B. (2016) Social Networks in "The Mana World" - An Analysis of Social Ties in an Open Source MMORPG, *International Journal of Multimedia and Ubiquitous Engineering*, 11(3):257–272, 2016.

Schatten, M., Ševa, J., and Okreša-Đurić, B. (2015a) Big data analytics and the social web-a tutorial for the social scientist, *European Quarterly of Political Attitudes and Mentalities* (EQPAM), 4(3):30–81, 2015.

Schatten, M., Ševa, J., and Okreša-Đurić, B. (2015b) An introduction to social semantic web mining & big data analytics for political attitudes and mentalities research. *European Quarterly of Political Attitudes and Mentalities EQPAM,* 4(1):40–62, 2015.

Schatten, M., Ševa, J., and Tomičić, I. (2016) A roadmap for scalable agent organizations in the internet of everything. *Journal of Systems and Software*, 115:31–41, 2016.

Schatten, M., Tomičić, I., and Okreša-Đurić, B. (2015) Multi-agent modeling methods for massively Multi-Player On-Line Role-Playing Games, in: P. Biljanović (Ed.), 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), pp. 1256–1261, Opatija, HR, May 2015, IEEE.

Voinea, C.F., and Schatten, M. (2015) Recovering the Past. Eastern European Web-Mining Platforms for Reconstructing Political Attitudes, *European Quarterly of Political Attitudes and Mentalities*, 4:22–39, 2015.

ISSN 2285 – 4916
ISSN-L 2285 - 4916