ModelMMORPG - D4.7. Report on experimental data analysis

Schatten, Markus; Okreša Đurić, Bogdan; Tomičić, Igor

Large-Scale Multi-Agent Modeling of Massively Multi-Player On-line Role-Playing Games

D4.7. Report on experimental data analysis

This project was funded by the Croatian Science Foundation

Principal investigator:

Markus Schatten

lab

2015.

PUBLISHED BY ARTIFICIAL INTELLIGENCE LABORATORY,
FACULTY OF ORGANIZATION AND INFORMATICS, UNIVERSITY OF ZAGREB

HTTP://AI.FOI.HR/MODELMMORPG

Some of the results presented in this deliverable have been previously published in [44, 45].

Technical Report No. AIL2017001 – First release, November 2017

Document compiled by: Markus Schatten, Bogdan Okreša Đurić and Igor Tomičić with inputs from other project team members

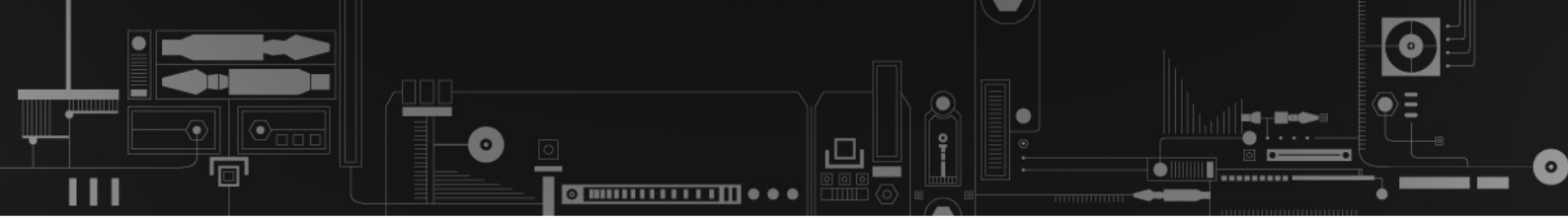
This work has been supported in full by the Croatian Science Foundation under the project number 8537.

hrzz
Hrvatska zaklada za znanost

foi
SVEUČILIŠTE U ZAGREBU
FAKULTET
ORGANIZACIJE I
INFORMATIKE
VARAŽDIN

lab

Contents



hrzz
Hrvatska zaklada za znanost

foi
SVEUČILIŠTE U ZAGREBU
FAKULTET
ORGANIZACIJE I
INFORMATIKE
VARAŽDIN

lab

1. Project Description

1.1 Abstract

Massively multi-player on-line role playing games (MMORPGs) give us the opportunity to study two important aspects of computing: (1) large-scale virtual social interaction of people (players) and (2) the design, development and coordination of large-scale distributed artificial inteligence (AI). A common denominator for both aspects are the methods used to study them: social interaction can be descibed and simulated using agent-based models (ABM social science perspective) whilst distributed AI is commonly modelled in terms of multi-agent systems (MAS computer science perspective).

The important question to ask in both perspectives is how do agents organize in order to perform their tasks and reach their objectives? Project ModelMMORPG (Large-Scale Multi-Agent Modelling of Massively On-Line Role-Playing Games) will employ a combined empirical and theoretical approach towards finding the answer to this question.

From the empirical side, we shall study the human behaviour on a number of venues across various gaming servers in order to find most suitable structures, cultures, processes, strategies and dynamics employed by most successful player communities. From the theoretical side, we shall test a multitude of organizational architectures from organization theory in various MMORPG settings, and compare them with methods found in empirical research.

Our research is therefore aimed towards enriching the organizational design methods for the development of MMORPG to foster the development of self-organizing and adaptable networks of large-scale multi-agent systems.

With this in mind, our main goals are:

1. To identify and formalize adequate organizational design methods for developing LSMAS in MMORPGs.
2. To couple them with real-life and future scenarios from industry.
3. To provide open and accessible tools, which will allow for design, development, implementation, control, simulation and maintenance of LSMAS in MMORPG

1.2 Introduction

Role-playing video or computer games (commonly referred to as only role-playing games or RPGs) are a game genre in which the player controls the actions of some protagonist (or potentially several party members) in a world which is well defined [57]. A massively multi-player on-line game (MMOG) is a (computer) game that supports a great number of players playing on-line simultaneously causing or even fostering interaction among them [55]. Massively multi-player on-line role playing games are thus a mixture of these two genres allowing players to control the action of their protagonist (avatar) by interacting with a potentially large user-base on-line [56].

The global market for MMO games is growing rapidly with 2011 $\approx$ 8.5 billion €, 2012 $\approx$ 10.2 Bn€, 2013 $\approx$ 11.7 Bn€ and 2014 $\approx$ 15.0 Bn€ [19, 30]. While the economic importance of MMORPGs is obvious, another aspect is of equal importance: it allows us to investigate two aspects of large-scale computing - (1) social interaction of (large numbers of) players through a computing platform as well as (2) the design and implementation large-scale distributed artificial intelligence (in form of non-player characters – NPCs, mobs – various monsters to be fought, as well as AI players – bots). Both aspects can and should be studied using agent-based methods, the former by ABM (a social science perspective) and the latter by MAS (a computer science perspective), whilst the important question to ask in both perspectives is: how do agents organize in order to perform their tasks and reach their objectives?

The ModelMMORPG (Large-Scale Multi-Agent Modeling of Massively On-Line Role-Playing Games) project will employ a combined empirical and theoretical approach towards finding the answer to this question. From the empirical side, we shall study the human behavior on a number of venues across various gaming servers in order to find most suitable organizational structures, cultures, processes, strategies and dynamics employed by most successful player communities. From the theoretical side, we shall test a multitude of organizational architectures from organization theory in various MMORPG settings, and compare them with methods found in empirical research. The research is therefore aimed towards enriching the organizational design methods for the development of MMORPGs and to understand the undelying principles of self-organizing and adaptable networks of large-scale multi-agent systems.

MMORPGs have a number of different subgenres, but a usual setting is that a protagonist is placed into a world in which he interacts with various NPCs and mobs which give out tasks (quests) that it has to solve to be able to buy better equipment, learn new skills like magic and similar, or proceed to higher levels. In ModelMMORPG we have chosen The Mana World (TMW)[1] MMORPG to conduct our research. The reasons for selection were: (a) it is open source (GPL licensed) allowing us to modify code and add additional functionality, (b) it has a supportive community, (c) it supports a number of interaction techniques which can be studied (e.g. trade among players, IRC based chat, organizing teams called parties, social network functions e.g. friends, enemies, parties etc.), (d) it is a (more or less) finished game featuring lots of quests that can be analyzed.

In order to answer our outlined questions, we firstly designed a special quest in which players ought to organize their activities in order to solve it. The quest is designed during a 3-day brainstorming session. Later on, during a data collection phase we will allow players to play the game in order to collect their behavioral data during a period of one month. After data collection, the data has been analyzed using social network analysis (SNA) and natural language processing (NLP) techniques in order to identify patterns of organizational behavior among successful players. In addition to these patterns, various organizational forms from organization theory literature have been been formalized in form of an ontology and consequently a meta-model for a graphical modelling tool. This tool has been extended with end-point plug-ins that allow us to not only

[1] See `http://themanaworld.org` for details.

develop models of agent organizations, but also to generate code templates to implement such agent systems. Herein three such organizations have been modelled and from these models three simulations were implemented in order to test the expressivity of the modelling tool as well as analyze data gathered during simulation. In this deliverable the results of these simulation experiments are presented.

1.3 Team Members

Markus Schatten (Principal investigator)
Head of Artificial Intelligence Laboratory
markus.schatten@foi.hr
+385 42 390891

Joaquim Belo Lopes Filipe
Departament of Systems and Informatics, Escola Superior de Tecnologia de Setúbal
joaquim.filipe@estsetubal.ips.pt
+351 265 790 040

Nikola Ivković
Department of Computing and Technology
nikola.ivkovic@foi.hr
+385 42 390872

Mladen Konecki
Department of Theoretical and Applied Foundations of Information Sciences
mladen.konecki@foi.hr
+385 42 390873

Mario Konecki
Department of Theoretical and Applied Foundations of Information Sciences
mario.konecki@foi.hr
+385 42 390834

Robert Kudelić
Department of Theoretical and Applied Foundations of Information Sciences
robert.kudelic@foi.hr
+385 42 390852

Ivan Magdalenić
Department of Computing and Technology
ivan.magdalenic@foi.hr
+385 42 390872

Marko Maliković
Faculty of Humanities and Social Sciences Rijeka
marko@ffri.hr
+385-(0)51-265-765

Bogdan Okreša Đurić
Artificial Intelligence Laboratory
dokresa@foi.hr
+385 42 390891

Jurica Ševa
Department of Theoretical and Applied Foundations of Information Sciences
jseva@foi.hr
+385 42 390873

Pietro Terna
Università di Torino
pietro.terna@unito.it
+39 011 6706067

Igor Tomičić
Department of Computing and Technology
igor.tomicic@foi.hr
+385 42 390869

2. Introduction

At the begining of the ModelMMORPG project we wanted to develop a framework for design, development, implementation, control, simulation and maintenance of large-scale multiagent systems (LSMASs) that is supported by open and accessible software tools. Massively Multi-player Online Role-Playing Games (MMORPGs) were identified as a natural playground for studying such complex systems, but other domains including but not limited to the Industry 4.0, cooperative intelligent transport systems (C-ITSs), smart grids, smart cities, virtual & augmented reality (VAR), electronic sports (eSports), socio-economic systems etc. These application domains (ADs) intentionally overlap to show some of the basic building blocks of modern large scale applications - organization, cooperation, competition, robustness and (collective) intelligence [41].

Three such domains have been selected herein to test the expressivity of the framework and tools which have been developed during the ModelMMORPG project: (1) a large-scale economic simulation (based on the recipeWorld by Fontana and Terna [17]), (2) an open-source MMORPG called "The Mana World" as well as (3) a Smart City / Smart Grid environment (based on a smart self-sustainabilty framework developed in [49]).

These three domains were modelled using the developed LSMASs modelling tool (developed in [13, 14] and then further implemented using particular end-point plug-ins one of which was presented in [45] and [44]. The first simulation experiment is oriented towards agent-based modelling of socio-economic phenomena. The third is a simulation of a realistic technical system which could (with minor adjustments) be directly implemented into a real-world scenario. The second is a mixture of both: on one hand it allows us to study a social phenomena (on-line behaviour of human players) whilst on the other it allows us to implement distributed artificial intelligence systems for various tasks like automated game testing.

The first simulation experiment tries to formalize a scenario in which an economic network emerges between companies who share orders with each other through so called recipes (production templates). In order to reach further from the initial example given by [17] the experiment also includes mergers and acquisitions between companies based on two strategies: (1) monopolists - try to establish a monolopy in a specific product, and (2) diversifiers - try to diversify their portfolio by having as much as possible different products.

The second simulation experiment is bound to the MMORPG "The Mana World" for which belief desire intention (BDI) agents with a knowledge-base and automated planning system have been implemented to allow for automated game testing. These agents have the ability to play the game as human players would, and are able to solve a number of implemented quests. To test for their organizational capabilities we have implemented three types of agents: (1) leaders - establish a party and invite other players, (2) extremist followers - join the first party they are invited to and never leave it, (3) opportunists - joint the first party they are invited to and afterwards leave it if there is an invitation to a party with better performance.

The third simulation experiment deals with the idea of a smart resource sharing grid in a smart city environment. This experiment is an expansion of the experiments shown in [51, 52]. The scenario models a smart city in which devices (agents) of various kinds (resource producers, resource consumers, resource storages) share resources in order to facilitate self-sustainability. Due to the possible scale of a smart city environment, the agents are organized in a hierarchical structure that minimizes unnecessary broadcast communication.

In the following chapters these three scenarios will be analyzed in more detail.

3. 1st Large-Scale Experiment - recipeWorld

3.1 Introduction

The work presented in this chapter was motivated by contents of the Complex Networks course at Vilfredo Pareto Doctorate in Economics, which includes dealing with agents, their interaction and analysis of a generated network (i.e. graph) based on agent interaction within the observed system. The analysis of such a structure can give us interesting insight into how a system develops and thrives, how resources within the system are transferred, which agents interact, etc.

Organization is often defined formally, in a way which should be respected since it is a part of organizational culture, yet formal ways are often avoided and substituted by more effective actions. When interaction patterns are prescribed, deviations from the defined path can be observed, e.g. when people create shortcuts through green areas of parks because the prescribed walking lane is not effective. The idea of such "natural" creation of norms is rather well-researched, yet it is interesting to observe interaction of agents and analyse the emerging network, based solely on agent activity.

3.2 Network of Agent Interaction

3.2.1 Network

A network in the most general terms is "*any system that admits an abstract mathematical representation as a graph whose nodes (vertices) identify the elements of the system and in which the set of connecting links (edges) represent the presence of a relation or interaction among those elements*" [5]. Edges can have a specific direction, creating directed graphs or networks, or they can have specific weight values assigned, thus creating weighted graphs or networks. Networks and network analysis are performed by using techniques mostly coming from graph theory, and as such have long been used in areas including discrete mathematics, communication research, sociology, etc. [5].

Networks are analysed and described using network measures which comprise a vast array of node- or edge-based measures, including degree centrality and other centrality measures, pure degree measures, clustering, degree distribution, and many others. Based on statistics and graph

theory, these measures help identify properties of the given network.

3.2.2 Agents

An agent is an individual in a multi-agent system (MAS). Usually, in modern research and in context of agent-based modelling (ABM), an agent is a piece of software that can perceive its environment using sensors, and act upon that same environment using actuators, as presented in Fig. 3.1. Agents can interact with each other, e.g. in form of communication using message exchange protocols.

An example of agent interaction, where both agent-based modelling and network analysis are included, is given in [17] in form of the *recipeWorld*, which is an agent-based model that "*simulates the emergence of a network out of a decentralized autonomous interaction*". It is argued in [17] that a 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), behaviour of nodes, and methods of traditional mathematical modelling of networks.

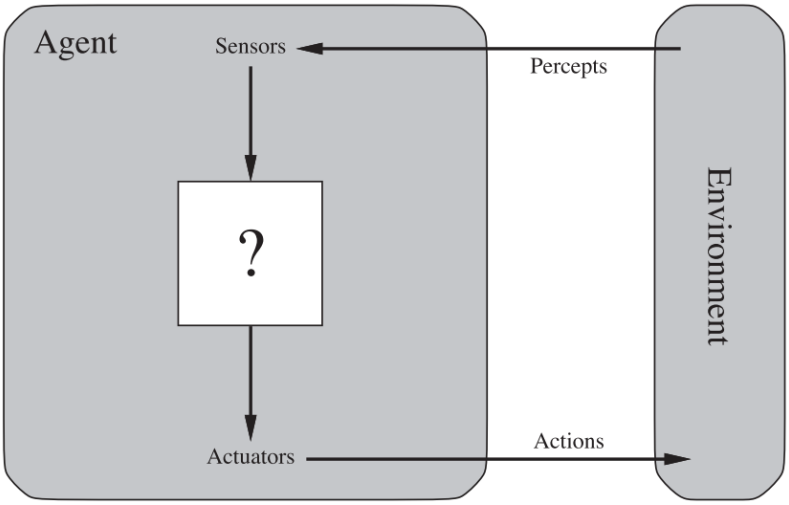


Figure 3.1: Simplified illustration of an agent and their relation with their environment [34]

Therefore, ABM and network analysis provide us with a good toolset for the creation of large networks generated by interaction of individual agents pursuing their own end. Such a combination can, as stated in [17], overcome e.g. static limitations of networks, since ABM/MAS is inherently dynamic, problem of modelling nodes and their likeness to their real world counterparts, since agents can have behaviour and attributes modelled as detailed as needed, and an ABM/MAS is constrained in number of agents only by available computational power.

An example system presented in this chapter will follow the basis laid out in [17], creating a dynamic system of interacting agents, which generates a network that can be further analysed. Afterwards we will extend this example by introducing organizational roles of companies in the network, as argued further.

3.2.3 Inital Example

A simple example useful for showing the general idea of how the metamodel can be used will be presented here, for the sake of clarifying the basics of the metamodel.

This example is based on the recipeWorld idea [17] comprising factories (service agents), orders (service-seeking agents) and recipe parts (services). For more in-depth description please refer to the original paper.

Figure 3.2 shows the basic model of the recipeWorld developed using the developed metamodel (see [13] for details) comprising the core elements of recipeWorld. An individual agent situated in the system fashioned after the recipeWorld idea is represented using an organisational unit element of the metamodel (stick figure in Fig. 3.2), in its individual state. Such an individual agent has access to individual knowledge artefact (black-yellow in Fig. 3.2) comprising individual knowledge of or about the given agent. Only two roles (blue hats in Fig. 3.2) are present - one for factory agents (service providers) and one for order agents (service seekers). Since the services are bound to individual agents, they are specified as a part of the agent's individual knowledge. Both roles have access to an organisational knowledge artefact (black-blue in Fig. 3.2) which defines the language used in the system, basic concepts needed for successful communication and their properties. Each of the two modelled roles provides the agent playing that particular role with a set of actions grouped into processes (green squares in Fig. 3.2). Specific objective (circular in Fig. 3.2) can be reached using an appropriate process, and, by extension, by playing the appropriate role. Objectives are modelled layered, i.e. the rightmost objective consists of all the objectives on the left of it that are connected to it.

The following example system is based upon the work of Fontana and Terna presented in [17], yet different tools were used. The example system described in this chapter was developed using the programming language Python, with two Python libraries as core of the developed system: SPADE[1] [21], and Ubigraph. SPADE is a MAS platform that allows users to create instances of specific classes representing agents and various kinds of behaviours, emphasizing behaviour of agents and their communication using messages. Ubigraph is a graphing tool based on *xmlrpclib* library for Python, which allows for dynamic 3D network visualization native to Python. Fused together, Ubigraph and SPADE make MAS visualization possible in real time, as opposed to widespread tools which generate visualization of agent interaction using snapshots in discreet time.

Every SPADE agent's definition is based on an existent basic SPADE agent class that uses a function representing the agent's initial behaviour, and customized classes representing behaviours of the given agent, as shown in code listing 3.1.

Listing 3.1: Basic elements of agent class `AgentOrder`, and one behaviour class

```
class AgentOrder(spade.Agent.Agent):
    class SearchForFactories(spade.Behaviour.OneShotBehaviour):
        [...]
    def _setup(self):
        [...]
```

The example system consists of two types of agents:

- `AgentOrder` representing individual orders;
- `AgentFactory` representing individual factories.

Three different node types of the visualized network represent three important entities of the system, with edges representing interaction or relation (red edges represent production relation, and white edges represent belonging), visible in Fig. 3.3:

- Order agent is represented as a cube with three states identified by colours:
  - yellow order is created and progressing well;
  - green order has finished its production;
  - blue order contains a part which requires an unavailable factory service, and is therefore terminated.
- Each order contains several parts represented as spheres.
- Every factory agent is represented as a big icosahedron.

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

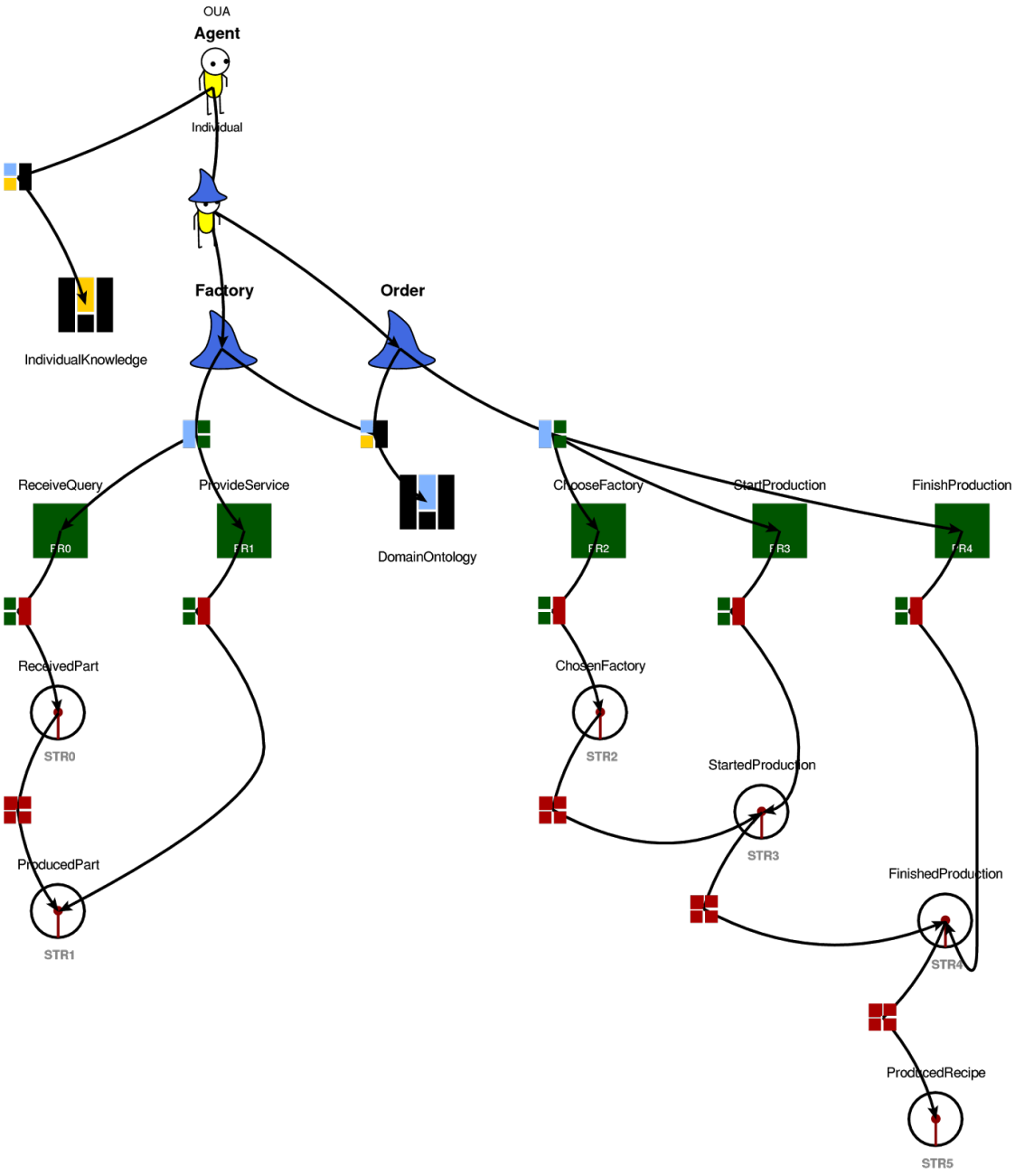


Figure 3.2: LSMAS organisational metamodel showcase using the recipeWorld example

Factory agent instances are created first. Each factory agent instance is given a set of services it offers to the system. These offered services are registered in the central system registry that can be searched by other agents in the system. Factory service combinations are first saved in an array, and dedicated to a specific factory upon agent creation.

After registering their services, factory agents are dormant until they receive a message from an order agent requesting production of one of their parts. 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. In the developed example system each factory can produce up to one part at any given time. Order and factory agent's message exchange is represented visually in Fig. 3.4.

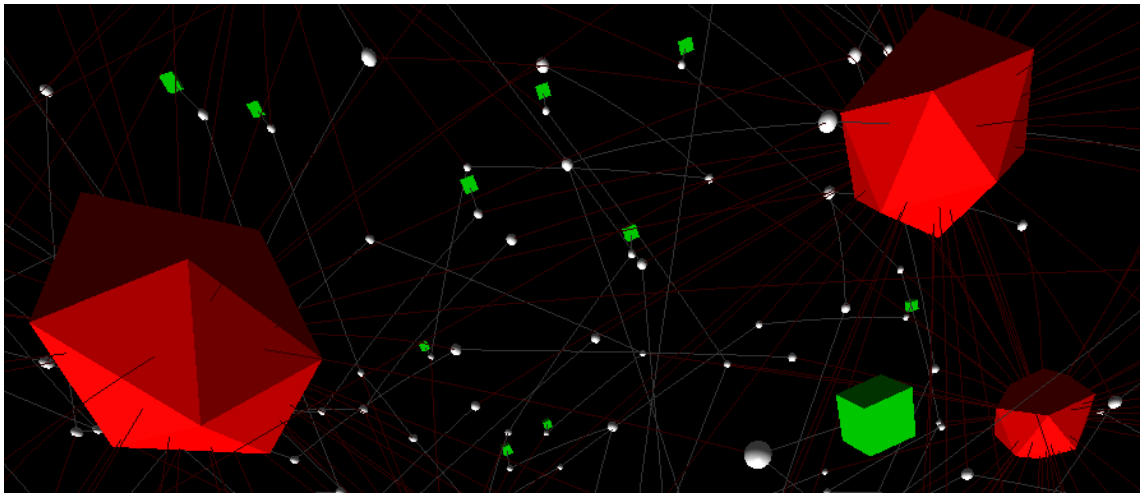


Figure 3.3: Three different node types of the network corresponding to entities of the system

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, all three represented as nodes in the emerging network. After the production is completed (lasting a random chosen period of time), the given factory is free to take another order's part, and the process is repeated.

Order agent instances are a bit more complex, having to look for available services and defining a set of containing recipe parts. The set of recipe parts is defined upon agent creation, based on the available services and defined number of recipe parts each order is supposed to contain. Each recipe part is added as a node to the growing graph immediately after being chosen, 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, as shown in Fig. 3.5.

Upon creation and definition of recipe parts, the order agent searches for the available services registered by factory agents. Since the order agent is in part defined as a finite state machine, after discovering whether the wanted service is offered, the agent continues on to another behaviour of theirs, based on the established service availability.

If the wanted service is registered as available in the system, order agent randomly chooses one of the factories providing the wanted service, and sends them a message proposing creation of the needed recipe part. Upon receiving an answer of positive (the factory is free) or negative (the factory is occupied) content, the order agent will continue with one of the two appropriate behaviours. In case of a negative answer from the factory agent, the order agent will choose another factory and repeat the negotiation process. Otherwise, if the wanted factory agent is free, the order agent requests start of production. Once no more recipe pieces are left, the order agent will finish production, change style of their respective node and terminate. This message exchange process is shown in Fig. 3.4, and the possible behaviour changes of order agent are depicted on Fig. 3.6.

The resulting network is visually indicative of the interaction present in the system. The density of factory agent nodes indicates their production activity, and persistent edges between recipe parts and their respective order agent nodes continue to illustrate belonging (which recipe part belongs to which order agent).

The developed example system makes it possible to observe changes in the system dynamically, as they are happening, capturing interaction and communication dynamics in real time. The generated network explorable in 3D is exported as a list of edges when the simulation is finished. This final output file contains all the data needed for further analysis of the generated network,

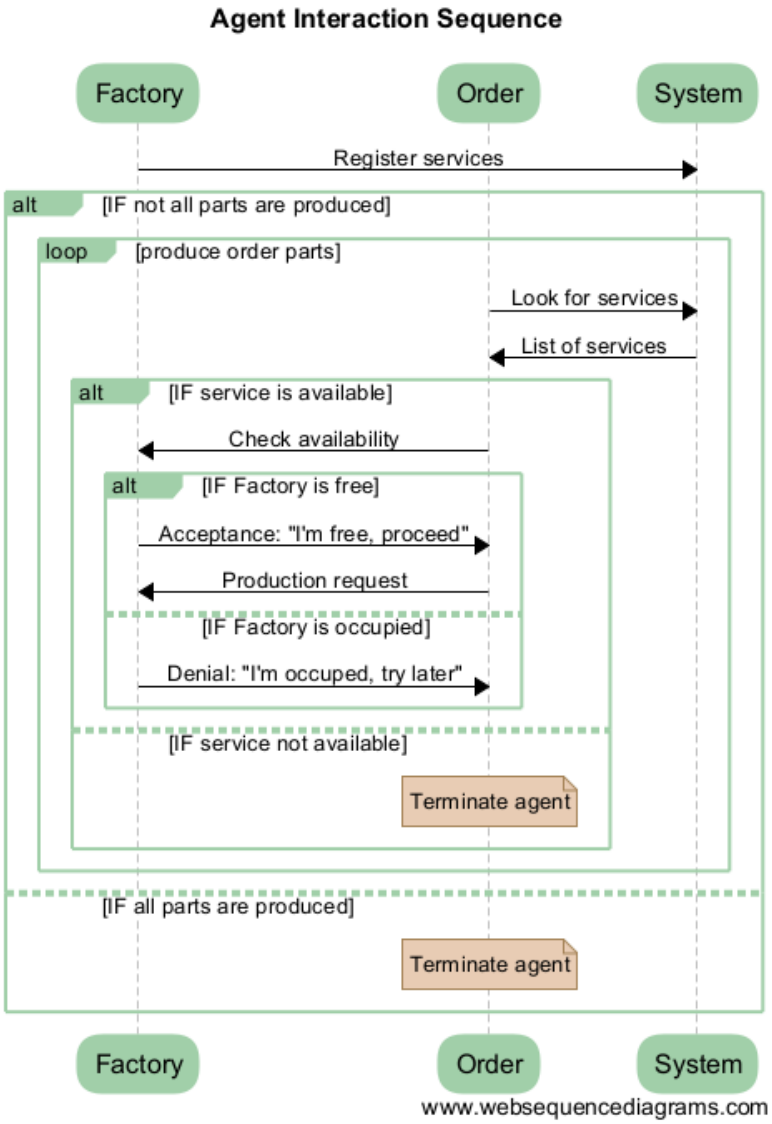


Figure 3.4: Interaction sequence diagram of the example system's agents

using any of the earlier mentioned network measures, or some other methods.

The developed example system, in its current state is a very basic one, with simple features only, but with potential for expansion in ways of defining further attributes of both types of agents, as well as recipe parts. Furthermore, easier manipulation of some basic elements crucial to execution of the simulation (e.g. number of factory agents, number of order agents, number of available services, etc.) would enhance ease of use of the visualization and its practicality.

A video[2] is available to showcase how the example system behaves when populated by 5 factory (limited to providing only 1 service each) and 32 order agents (with 4 recipe parts each), with 4 particular services available. Source code of the example system is available at `https://goo.gl/txoF1h`, without the necessary SPADE and Ubigraph platforms.

[2]The referenced video was made by Bogdan Okreša Đurić the author of this chapter, available for watching at `https://goo.gl/FQqcEZ`

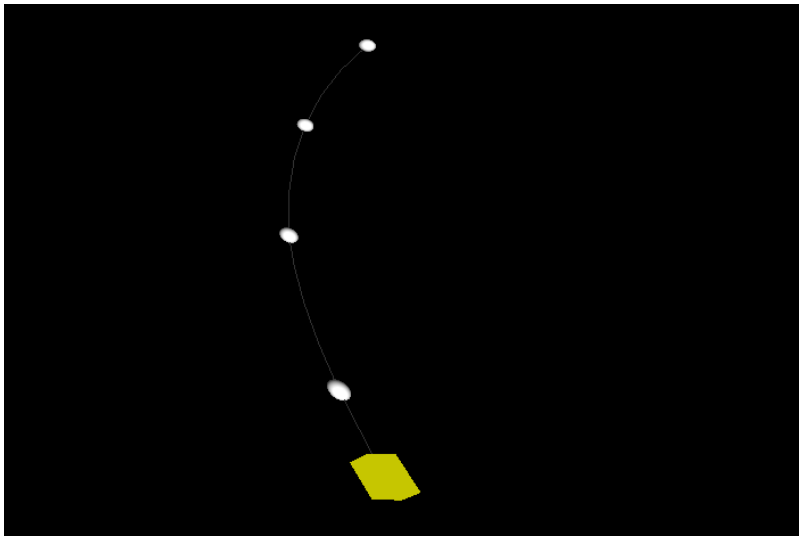


Figure 3.5: Visual representation of order agent node, and the accompanying recipe part nodes

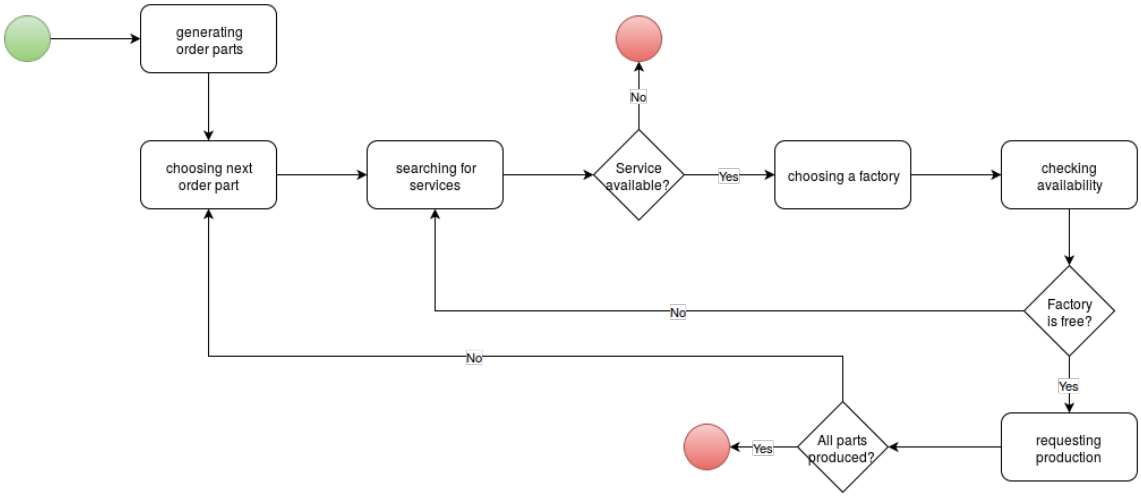


Figure 3.6: Inner states of an order agent, and their flow from agent creation (green circle) to agent termination (red circle)

3.3 Advanced Example

The more complex take on the example described above, is concerned with simple organizational concepts that are built upon the existing example scenario. With the goal of simulating data about the behaviour of factories based on their principal types: Monopolist or Diversifier.

There is one basic change in this example though, concerning the visualization technique - the used visualization package is pynteractive[3] instead of Ubigraph since it became obsolete.

The necessary element to be introduced is a sort of a resource in the system - money. All agent start with a specific amount of money at the beginning of the simulation, which varies between the class of agents (factories start with almost nothing, and they earn money, while orders start with some money, and spend it on using services at specific factories).

Factories have a few important new features added, such as the ability to perform a takeover - buying another factory agent and taking all their money earned in the future, effectively becoming

[3]For further info see: `https://github.com/coelias/Pynteractive`

their owners. The stated allows for creation of groups of factories with defined leaders. Furthermore, clusters can be formed this way that e.g. provide all the possible services available in the simulation, or, conversely, that provide all the instances of one specific service in the simulation, thus creating a monopoly position.

The order agents have remained mostly unchanged, with minor changes conforming to the new simulation version.

The main difference between the stated factory types (Monopolist and Diversifier) is in the manner they treat the takeover process. Monopolist factory agents are looking for other factory agents that provide the same services as they do (i.e. at least one service that is the same as the services provided by the given factory agent). Therefore, they aim on removing any factory agents that would be a threat to them. Contrariwise, the Diversifier factory agents look for other factory agents that provide services that the given factory agent cannot provide, thus widening their palette of offered services, making themselves available to the wider range of oncoming order agents (i.e. the newly formed cluster can provide all the services, but the factory agent that is the owner receives all the necessary money). On the subject of money - it is being transferred from the owned factory agents to their owners, even in a multi-layered ownership, e.g. when the factory agent A already owned factory agent B, and factory agent C bought A, the money from B is transferred all the way to C.

The frequency of a factory agent checking whether there is a factory agent to be taken over is defined within the programming code, and is, at this moment, based on the production volume of a factory agent - the more the factory agent produces, the more chance they have for performing a takeover.

Another novelty within the simulation example is a newly introduced agent class: AgentCharter. The said agent is instantiated only once, and its main purpose is to gather data from all the active factory agents, and write them down in an external file. The addition of this kind of an agent is welcome in the context of creating data in a viable format easily accessible for further analysis. This agent is the observer of the simulation.

3.3.1 Results and Discussion

The simulation was run several times with varying ratio of Monopolist and Diversifier agents. One set of simulations was run with 25% of factory agents being Monopolists and 75% of factory agents being Diversifiers; one set of simulations was run with 50% of factory agents being Monopolists and 50% of factory agents being Diversifiers; and one set of simulations was run with 75% of factory agents being Monopolists and 25% of factory agents being Diversifiers. The mentioned sets are further referenced as 25-75, 50-50, and 75-25 respectively.

The results of each of the simulation types is shown in figures 3.7 through 3.12.

The basic settings of all the simulation runs, for all the simulation types, are the same: 100 factory agents, each providing two random types of services from the pool of 20 of them, serving 1000 order agents. Factory agents start with 10 money, while order agents start with the amount of money between 250 and 500 units. Neither prices of individual services, nor their necessary time for production, have not changed between various runs of simulations.

A general situation visible from the line charts of these simulation types is a sharp drop of money value of a factory agent. This phenomenon is typical for a takeover moment, meaning that the given factory agent became an owner of another factory agent. Notice that, in most of the cases where this is noticeable, the steepness of the curve following the takeover event is greater than before the given event.

The results shown in Fig. 3.7 show top 10 factory agents through simulation time, by their accumulated money, in the 25-75 simulation. Top 10 factory agent are ranked as such based on their result in the final moment of the simulation, which is visualised in Fig. 3.8. Figure 3.7 therefore

shows 10 most profitable factory agents in the 25-75 simulation type. It is worthy to note that out of the top 10 factory agents, most of them are Diversifiers, with only three Monopolists. Considering the starting ratio of 25-75, the ending ratio of the top 10 factory agents is 33-67, meaning that, in the constraints of the simulation, the Monopolist factory agents grew stronger in time. It is easier to observe this effect in Fig. 3.8, where it is visible that Monopolist factory agents have 26% of the total money shared by the top 10 factory agents. Therefore, in the context of money, the starting ratio of the simulation has not changed drastically.

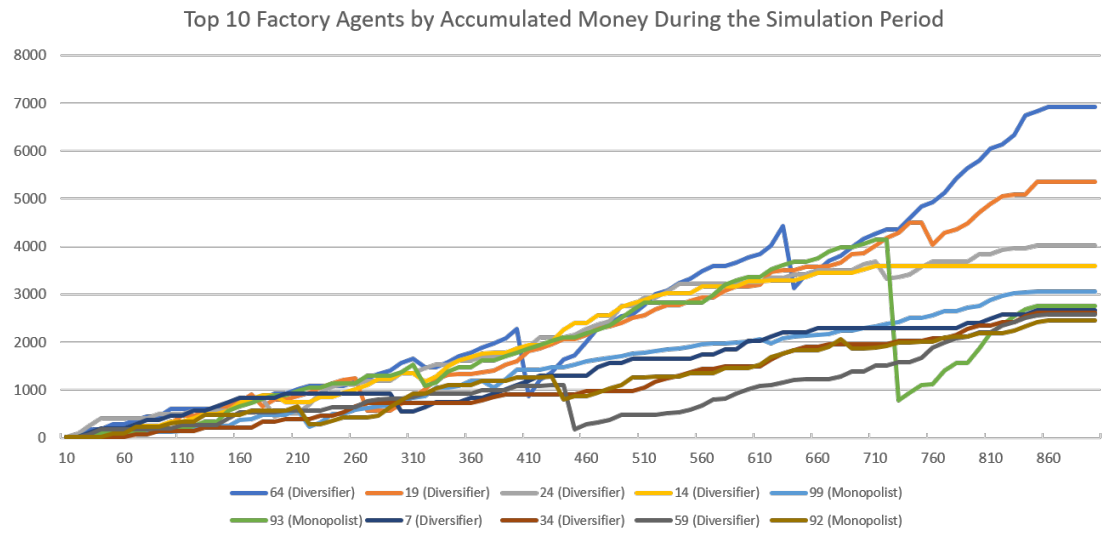


Figure 3.7: Top 10 factory agents by accumulated money during the simulation period (25-75 simulation)

The results shown in Fig. 3.9 show top 10 factory agents through simulation time, by their accumulated money, in the 50-50 simulation. Top 10 factory agent are ranked as such based on their result in the final moment of the simulation, which is visualised in Fig. 3.10. Figure 3.9 therefore shows 10 most profitable factory agents in the 50-50 simulation type. It is worthy to note that out of the top 10 factory agents, most of them are Diversifiers, with only four Monopolists. Considering the starting ratio of 50-50, the ending ratio of the top 10 factory agents is 40-60, meaning that, in the constraints of the simulation, the Monopolist factory agents grew weaker in time. It is easier to observe this effect in Fig. 3.10, where it is visible that Monopolist factory agents have 43% of the total money shared by the top 10 factory agents. Therefore, in the context of money, the starting ratio of the simulation has changed slightly, in favor of Diversifier factory agents.

The results shown in Fig. 3.11 show top 10 factory agents through simulation time, by their accumulated money, in the 75-25 simulation. Top 10 factory agent are ranked as such based on their result in the final moment of the simulation, which is visualised in Fig. 3.12. Figure 3.11 therefore shows 10 most profitable factory agents in the 75-25 simulation type. It is worthy to note that out of the top 10 factory agents, most of them are Monopolists, with only two Diversifiers. Considering the starting ratio of 75-25, the ending ratio of the top 10 factory agents is 80-20, meaning that, in the constraints of the simulation, the Monopolist factory agents grew stronger in time. It is easier to observe this effect in Fig. 3.12, where it is visible that Diversifier factory agents have 27% of the total money shared by the top 10 factory agents. Therefore, in the context of money, and under the constraints of this simulation, the starting ratio of the simulation has somewhat changed, in slight favor of Diversifier factory agents.

Analyses of the amount of money possessed by each of the factory agents at the end of the described simulations is valid here since all the factories start with the same amount of money at

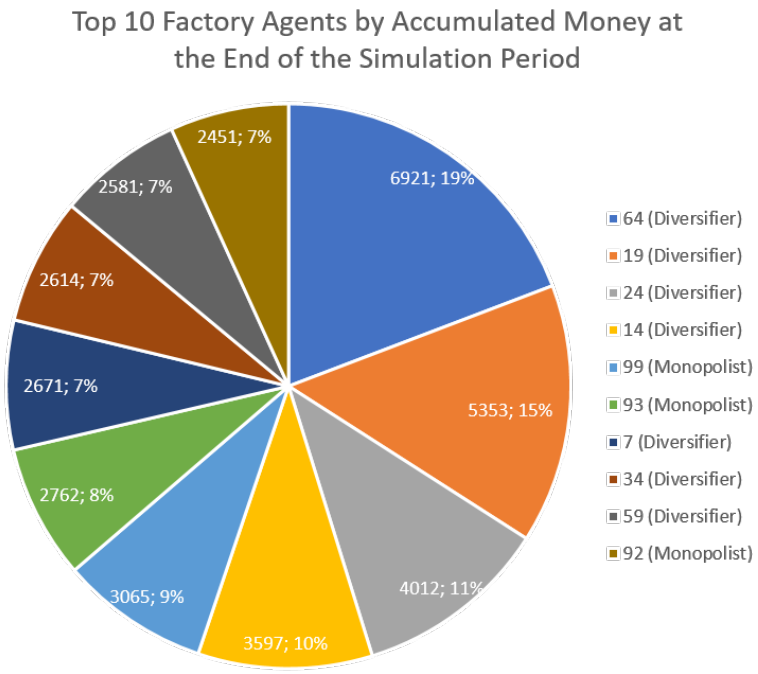


Figure 3.8: Percentage of top 10 factory agents by accumulated money during the simulation period (25-75 simulation)

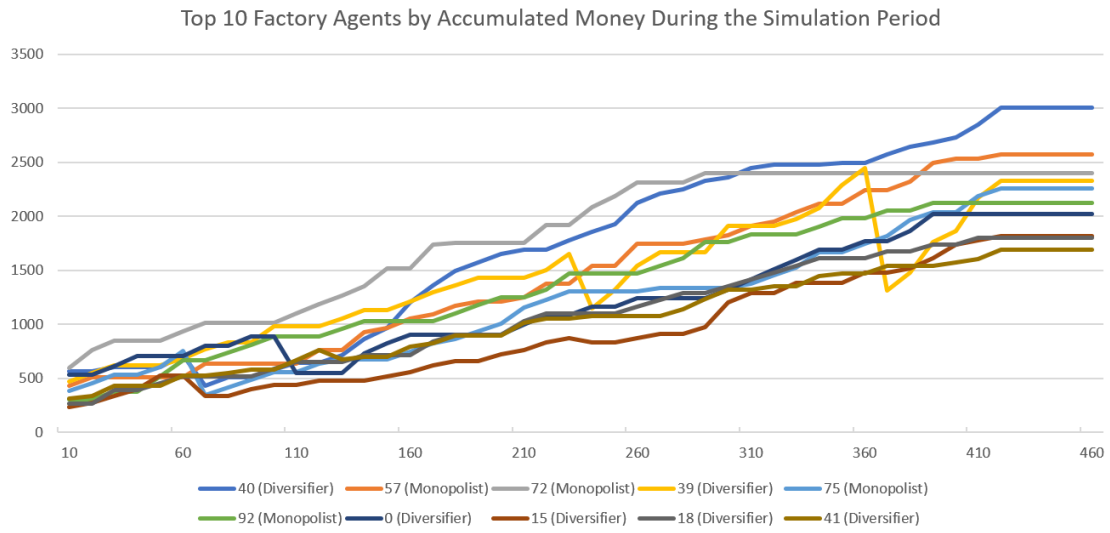


Figure 3.9: Top 10 factory agents by accumulated money during the simulation period (50-50 simulation)

the beginning of each simulation. Therefore the initial factory type ratio is applicable to the amount of money these factory agents have.

Based on the data at the end of the run simulations, it is noticeable that there is a slight advantage of Diversifier factory agents in the context of the amount of money they have, as opposed to the Monopolist factory agents. Conversely, the situation with the ratio of factory agent types remains mostly unchanged, even though slight differences are noticeable. There are several possible ways of

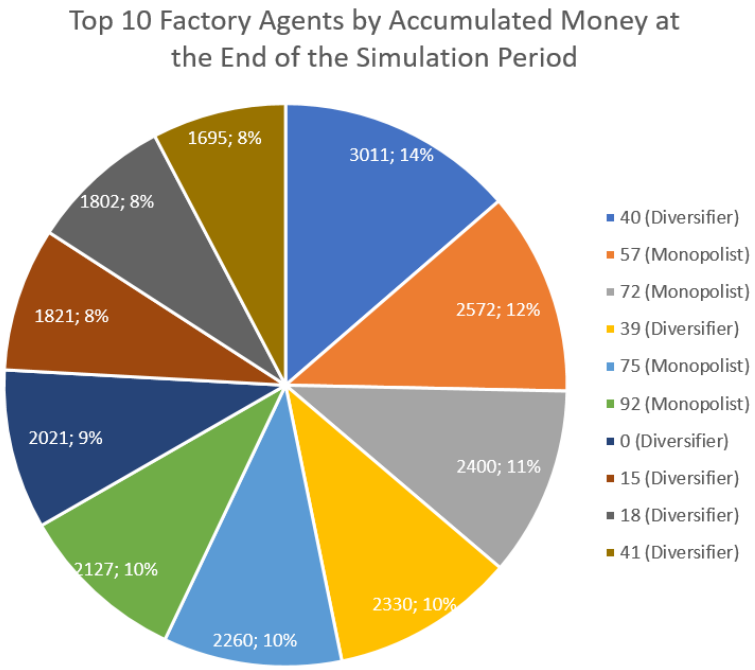


Figure 3.10: Percentage of top 10 factory agents by accumulated money during the simulation period (50-50 simulation)

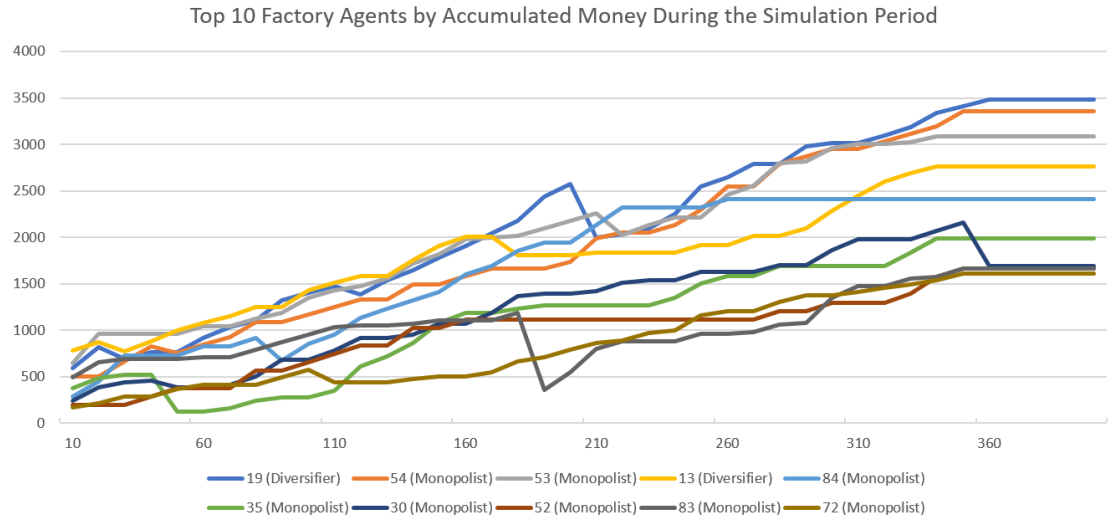


Figure 3.11: Top 10 factory agents by accumulated money during the simulation period (75-25 simulation)

explaining this observation. It is possible that the number of orders should be higher, thus providing more opportunities for factory agents to consider the possibility of taking over other factory agents. Furthermore, lowering the threshold for factory agents to consider the said possibility may be another way of increasing the chances of a takeover event taking place. Tinkering with the latter can create an even more unnatural environment though. The greatest constraint for the simulation with an increased number of order agents are hardware constraints, since the SPADE simulation in its current state is quite a resource-hungry process bordering unexecutability on an average personal

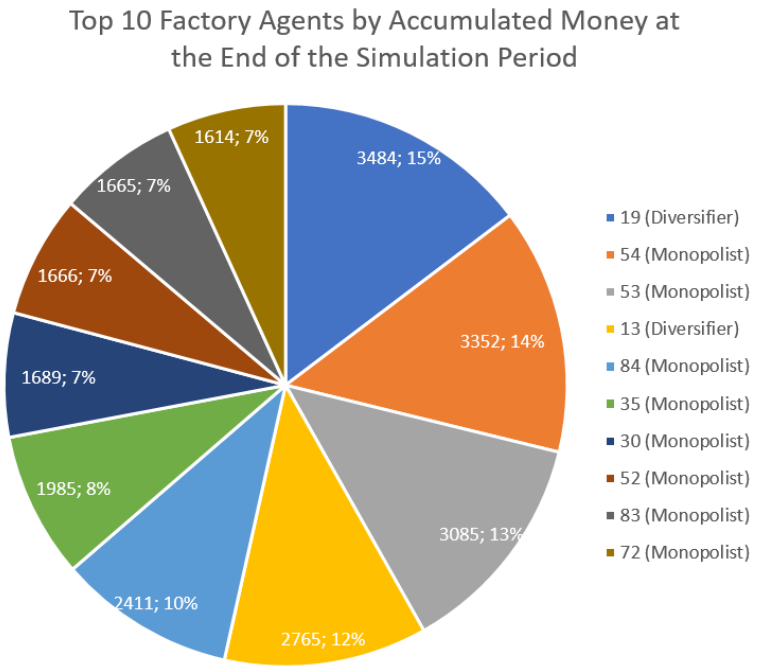


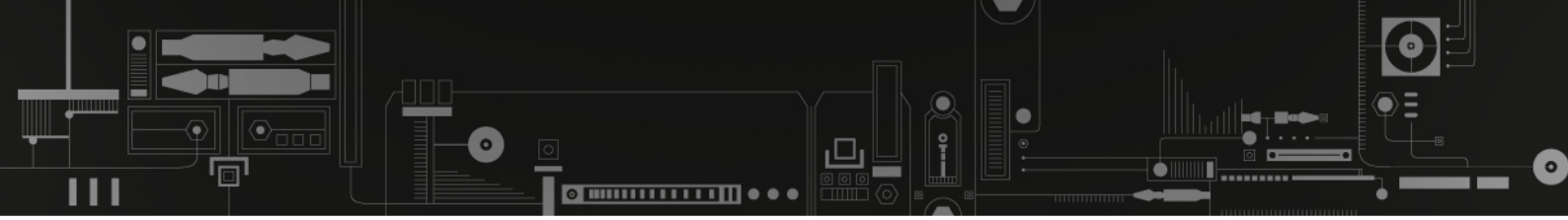
Figure 3.12: Percentage of top 10 factory agents by accumulated money during the simulation period

computer. Either way, it is presumed that longer simulation time would further the slight changes observable at this stage.

3.4 Conclusion

The example MAS/ABM described in this chapter is building upon the work of Fontana and Terna [17]. The main difference is in using another MAS platform, i.e. SPADE, and visualization tool capable of generating visual representation of the emerging agent interaction network in real time. The generated network can be further analysed using any of the usual network analysis tools (e.g. NetworkX for Python), since the results are saved in edge list format.

Even though the example system is a work in progress, it clearly shows how a production model may be developed using interacting agents. Furthermore, interacting agents may be used in many other application domains, e.g. traffic simulations and other situations that can be modelled using simulated communication, or even negotiation simulation.



hrzz
Hrvatska zaklada za znanost

foi
SVEUČILIŠTE U ZAGREBU
FAKULTET
ORGANIZACIJE I
INFORMATIKE
VARAŽDIN

lab

4. 2nd Large-Scale Experiment - The Mana World

4.1 Introduction

Automated testing of computer games is a complex task, especially in the context of multi-player games since, apart from testing the game logic, the interaction between players has to be tested as well. In massively multi-player on-line (MMO) games where thousands of players may play simultaneously, the task of automated testing becomes even more challenging. Role playing games (RPGs) on the other hand, require testing procedures that are able to solve complex tasks and puzzles (quests) that might not only include numerous actions like collecting items, fighting monsters, talking to non-playing characters (NPCs) etc. but also organizing into guilds or parties to solve harder tasks collectively. All of these challenges are combined in massively multi-player on-line role playing games (MMORPGs). In this chapter an agent-based and model-driven approach to testing an open-source MMORPG called "The Mana World" (TMW)[1].

The presented approach is agent-based in terms of artificial players or bots being modelled as agents within a TMW environment. The agents are implemented using SPADE (Smart Python Agent Development Environment) [21] with an implemented belief-desire-intention (BDI) architecture using an SWI Prolog knowledge base for reasoning. The approach is model-driven since a graphical modelling language is used to model the tests to be implemented and used for experiments. The graphical modelling tool was implemented using the AToM^3 meta-modelling toolkit [10].

MMORPGs offer a natural application domain for MASs, and especially large-scale MASs (LSMASs). MMORPGs are on-line games in which large numbers of players interact both collaborating and competing with each other by solving various quests or tasks. Not only (human) players interact, but also artificial entities including non-playing characters (NPCs), various mobs (the enemies or monsters to be fought), and some times even bots (artificial players), albeit in most cases illegally.

During the ModelMMORPG project we have set our objective to study MMORPGs from an agent-based perspective in order to develop methods and tools that will enable us to solve interesting problems bound to these games. Some of these problems include (1) automated gameplay testing – e.g. development of bots/agents that will play the game and deliver possible bottlenecks, (2) load

[1] See `https://www.themanaworld.org/` for details.

testing – MMORPG servers have to be robust under potentially very heavy loads of simultaneous players, (3) bot detection – on most MMORPGs forbid the use of automated player bots often used by players to acquire resources and use various automated and semi-automated Turing tests to recognize bots, (4) design of human-artificial agent interfaces – in such games players interact with numerous artificial entities like the mentioned NPCs and mobs, (5) analysis of social human on-line behaviour – players interact through various chatrooms and VoIP technologies, fight each other, organize into parties or guilds, create their social network etc., and all these data is often stored and available for analysis, (6) simulation of human on-line behaviour – using the previous mentioned data, (agent based) models of behaviour can be established.

The workplan of the ModelMMORPG project was to firstly study human behaviour in MMORPGs by developing a special quest for an open-source MMORPG called The Mana World (TMW) that would especially foster organizational behaviour between players. Results of this study that featured social network analysis and natural language processing methods, were presented in [40]. Additionally, and in-depth literature review was conducted to identify most valuable organizational design patterns that can be used in developing LSMAS [35]. Afterwards, a number of MMORPG modelling methods were identified [42] and a meta-model for LSMAS development was developed [13]. Finally, on the foundations of this meta-model, we are building a graphical modelling tool and model-driven LSMAS application generator that will allow us to generate LSMAS templates based on their organizational model. We will test this modelling tool and generator on MMORPGs on a number of scenarios. In this paper we shall report on the current state of development of this tool.

4.2 Related Work

MMORPGs continually prove to be a modern concept of applied information technologies providing users with a wide arrange of services, ranging from entertainment, relaxation, or action, over education, reflex, and reaction building, to research, experiment field, and a source of huge amounts of data. The impact of MMOGs (Massively Multiplayer Online Games) on modern-day market and life is undeniable [42], both in commercial and player context.

Automated playing agents are usually frowned upon by the gaming community which is reflected in the literature mostly dealing with techniques of preventing bots from playing MMO games (like [20, 26, 58]). Therefore the attempts at creating a virtual simulated world comprising artificial intelligent agents, such as presented within this research, is a rather unique task. There were only few attempts to create automated testing environments for large-scale settings like in [24] Jung et al. which in order to ensure game stability developed the VENUS simulator and an efficient method for simulating large numbers of virtual clients in on-line games. Also [6, 7] propose the VENUS II system allowing for blackbox and scenario-based testing by generating game grammar based user behaviour packages to facilitate various game-related scenarios.

4.3 The Mana World

TMW is a free 2D open source MMORPG licensed under GPL, with the visuals reminding of classic RPG games of the 1990s like Zelda or Final Fantasy.

Players of TMW are able to connect to one of several officially available game servers or install their own local server with optional customizations. From within the game, players are able to interact with each other by using a number available mechanisms such as chat, fights, trade, or creating social network communities (including friends, enemies, etc.) and organizing into parties of players. The game story itself is emerging from the quests and the numerous interactions, both of which are of crucial importance to this work – quests encourage forming parties, which are establishing their own inner interactions, mechanics, responsibilities, delegations, community feel, and other aspects of such social groups aiming to reach their common goal.

Figure 4.1: The Mana World – an open source MMORPG

We have chosen to use TMW as our main development and testing playground since it is open source and has a very helpful community.

Being an open source, TMW project includes client, server and on-line world development and dissemination, and calls for volunteer developers without restrictions to work on any part of the game: server, content, graphics, music, etc. The communication protocols and message formats that form the basis for client-server interaction are described in more detail in further sections.

4.4 System Architecture

There have been numerous attempts in establishing model driven game development frameworks (see [32, 47] for a good overview) as well as model driven MAS development (see for example [18, 31, 35]). Besides of [27] there have been very few studies dealing with agent-based game development, and even less with model driven agent-based game development.

One of the objectives of the ModelMMOPRG project was to establish an agent-based framework for the development of LSMASs with special regard to MMORPGs as an initial application domain. With this in mind, we have designed a complex development system which is general in terms of allowing to develop LSMASs regardless of applicative domain and have specialized it for MMORPGs through a plug-in system which allows us to connect various additional domain specific tools and libraries, depending on the problem at hand.

Figure 4.2 gives an outline of the system architecture which consists of three components, some having a number of sub-components: (1) an ontology defining the most important concepts LSMASs' organizations (see [15, 38] for more details); (2) the modelling tool comprising a metamodel (derived from the previously mentioned ontology [13]), a graphical user interface and an application template generator; and (3) a MMORPG plug-in which allows generating models for the use with TMW, consisting of a high-level interface, a low-level interface, a knowledge base and a planning system. Individual agents that are modeled as a part of the system using the modeling tool are detailed and instantiated in the latter part of the described architecture.

These components are described in more detail in the following sections.

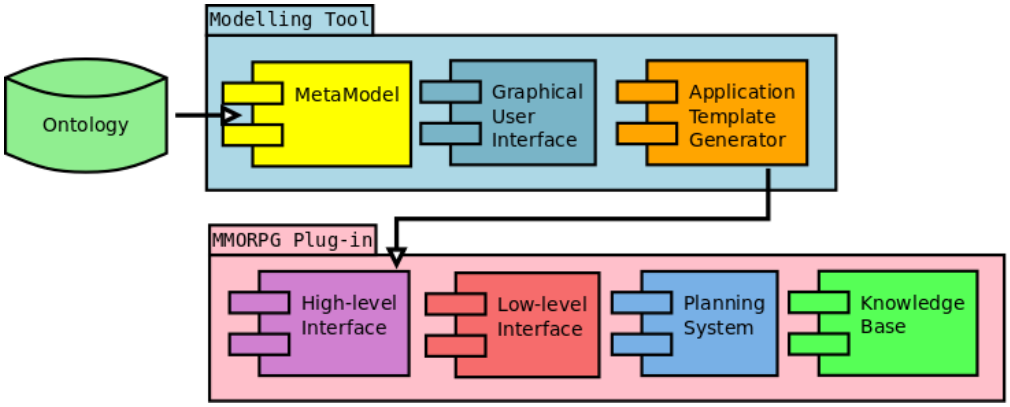


Figure 4.2: System Architecture

4.4.1 Modeling Tool

The modeling tool is implemented using AToM^3 metamodeling environment for several reasons, main of which are: AToM^3 is an open source tool freely available for download and use, meaning that possibly needed modifications can be introduced rather easily to the already available environment; AToM^3 is inherently designed for modeling and metamodeling processes, therefore is highly suitable for the set goals of this research; programming language used to develop AToM^3 is Python, a widely used language that is suitable for this research because it allows for easy integration with other parts of the described research, and is highly customizable and extensible using many libraries available.

The basis upon which this modeling tool was built is an organizational metamodel for LSMAS [13]. The current version of the metamodel[2] features several concepts that can be used for modeling a MAS: Organizational unit, Role, Objective, Individual knowledge artifact, Organizational knowledge artifact, Process, and various properties between the stated concepts. Additionally, to take into consideration organizational dynamics and agent interaction graph grammars and temporal logic are partially supported by the modeling tool.

The first step in developing the mentioned metamodel and subsequently the modeling tool for this research was to select concepts to be modeled.

The concept set was not introduced randomly, but based on the already available OOVASIS ontology [38] comprising organizational concepts useful for LSMAS, and a short review [1] of some of the prominent MAS and LSMAS models that are described and published thus far.

Furthermore, selection procedure for the concepts to be included in the meta-model was based on observing and testing various MMORPGs and the elements they feature.

Established core concepts are those that represent individuals, i.e. individual player-agents, and those that represent roles, i.e. grouped sets of organizational, behavioural, and normative constraints.

Several other concepts of high importance were identified when populating the set of concepts to be included in the metamodel as follows (some of them are visible in Fig. 4.3):

- Quest - MMORPG players are often motivated to advance through the game by quests, objectives consisting of smaller tasks that have to be achieved in order to successfully complete the given quest, since they are rewarded for quest completion either by receiving some important items, experience points (accumulation of which is vital to player's character progression), or another kind of a reward;
- Party - since the element of socialization is of great importance in MMORPGs, players

[2]The modeling tool is available at `https://github.com/Balannen/LSMASOMM`

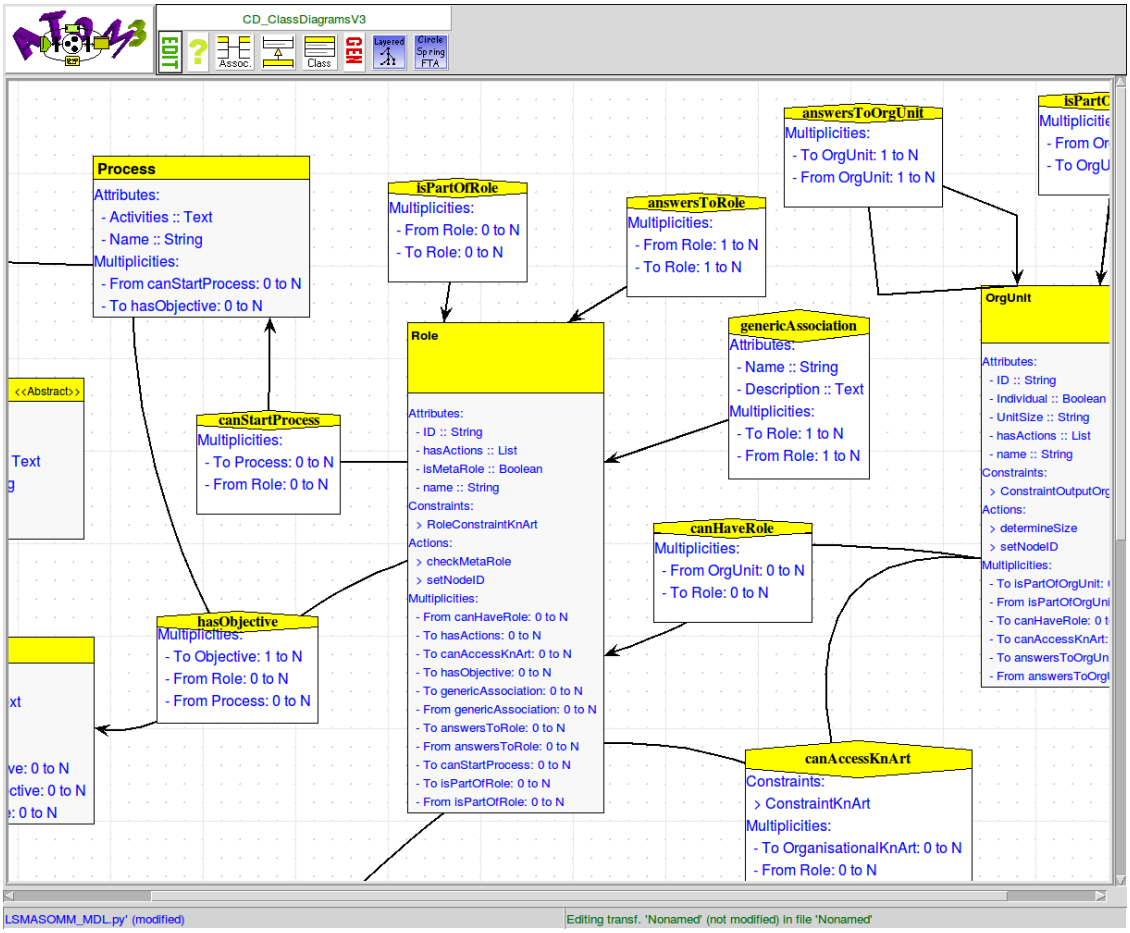


Figure 4.3: Some of the concepts of the metamodel being modeled in AToM^3 interface

often form groups (parties, guilds etc.) that help them utilize the power of cooperation when completing various quests, facing enemies, advancing through the game in general, or accessing parts of the game that would be inaccessible otherwise;

- Actions - since the system often used in MMORPGs grants specialization enhancements to players using the concept of roles, roles are used in the metamodel being developed as the way of defining various available actions that can be conducted by players playing various roles, and which are grouped into processes that can successfully complete a specific task, therefore proving to be vital for a specific quest the particular task is a part of.

It is worth noting that one of the objectives of the metamodel is to utilize a certain level of abstraction. Therefore no concepts are included that would allow a user to model details of a player agent, or many details of a role the players could play, or a specific way of defining a quest, etc. What the metamodel is designed for is a high-level description of a system of agents playing a particular game, providing users with features that allow them to generate a programing code outline for the modeled system.

One of the more practical features of the modeling tool is programing code-generating of the modeled system, described in the following section.

4.4.2 Application Template Generator

While the metamodel offers its users an environment for describing and modeling a system of agents in an MMORPG, the application template generator brings the modeled system closer to life.

The application template generating feature of the final modeling tool will allow the user to generate basic elements needed for implementing the modeled system. This feature can already serve for receiving essential programing code structure for Smart Python Agent Development Environment (SPADE) [21] agents. SPADE is the environment of choice since it is developed in and uses Python which makes it easy to integrate it with various additional modules (the modeling tool included) that help its integration with TMW, and because it allows for easy definitions of agents in a multiagent environment that use advanced deductive knowledge bases.

The generator uses enhanced native features of AToM^3 for storing and printing nodes of a model. Implementation is based on the Zope Object Database (ZODB), an object-oriented database for transparently and persistently storing Python objects. The use of ZODB is motivated by further development of the metamodel, in a multi-model modeling direction that would allow a user to model a system using several different models representing various layers or perspectives of the same system, therefore using the same elements.

Essential information was therefore extracted from the elements of a model, and stored in ZODB in custom format. Saved data is then converted using the current version of the application template generator into essential programing code for a system comprising SPADE agents.

4.4.3 High-Level Interface

The high-level interface[3] implements a belief-desire-intention (BDI) agent model using SPADE. In it's current state, it defines an artificial player with a number of agent behaviours in which the agent observes it's environment (observation is provided by the low-level interface and stored in the agent's knowledge base described below), contemplates about current objectives it wants to achieve (actual quests provided by various NPCs from the game), plans it's future actions based on knowledge about the world (using a STRIPS [16] based planner described below) and goes on to take actions to implement the current plan (using basic actions provided by the low-level interface).

The high-level interface provides the intersection between the modelling tool and application template generator above and the low-level interface, knowledge base and planning system below. Thus, it had to be developed last and has to integrate almost all other components. While the connection to the lower layers (low level interface, knowledge base, planner) is more or less well established (the low level interface is a Python module easily imported into the high level interface; the knowledge base and the planner are SWI Prolog modules easily interfaced through SPADE's knowledge base features), the connection to the upper layers (modelling tool and application template generator) need to be developed further in future research.

4.4.4 Knowledge Base & Planning System

The knowledge base and the planning systems have been partially described in [28]. The knowledge base consists of a TMW ontology that defines the basic entities and actions available in the game (NPCc, items, mobs, maps etc.) as well as detailed descriptions on possible quests. The quest descriptions were derived from the TMW wiki and coded into Prolog, while the map descriptions were parsed from the actual XML map descriptions from TMW code. Herein we have chosen to use SWI Prolog to implement the knowledge bases and planner, mainly due to the fact that SWI is the most complete and available Prolog system. Still, other options like XSB, ECLiPSe or Flora-2 are possibilities. The planning system is a STRIPS-based planner which consists of a set of rules which model the various quests provided by NPCs in the game. For example, the following listing defines the "Trade skill" quest in which a player can learn to trade items with other players by giving a trader in the city of Tulimshar 5 gold coins.

[3]The MMORPG plug-in including the high-level interface are available at `https://github.com/tomicic/ModelMMORPG`

Listing 4.1: Example planning rule

```
do_quest(A,trade_skill) :-
% Preconditions:
    location(trader_in_tulimshar,XTT,YTT),
    level(A,La), La>=1,
    \+ done_quest(A,done_trade_skill),
    money(A,Ma), Ma>=5,
% Deletions:
    retract(money(A,Ma)),
% Additions:
    walk_to_location(A,XTT,YTT),
    assert(plan(talk(A,trader_in_tulimshar,'Do you have anything for me?'))
        ↪ ),
    NewMa is Ma-5, assert(money(A,NewMa)),
    assert(ability(A,trade_ability)),
    assert(done_quest(A,done_trade_skill)).
```

4.4.5 Low-Level Interface

The main architecture of TMW is build around three servers and a client. The servers carry out the following functions:

- Managing accounts and connections to character server (login server - the first server the client connects to);
- Managing characters, connecting them to the map server, handling saving and loading of character data (character server);
- Managing game content (such as monsters, items, maps, etc.) and the interaction of game content with players (map server).

Communication between TMW server and client is implemented by custom application layer messages on the top of TCP transport layer protocol. Both the client and the server are implementing finite state machines that are changing their states in response to messages which are being sent and received between them. The messages are mostly defined with fixed length binary fields, but some messages contain variable length fields. Unfortunately, only several message types are fully documented, and the others merely pointed to the locations of C++ source code where they should have been implemented.

We have reverse engineered the structure of the messages by analyzing the source code and the real-world network traffic by using the network protocol analyzer Wireshark. To make the network traffic analyses easier we have developed a basic packet dissector for Wireshark using the Lua programming language and semiautomated tools in spreadsheet processing applications. Since TCP deals with streams of bytes it is up to the programmer to discover borders between different Mana messages. More than one Mana message can arrive in the same TCP segment, but also one message can be separated into multiple TCP segments. Based on the value at the beginning of the message, the type and the structure are inferred. Especially hard to decode are the messages that do not obey byte boundaries, e.g. 12-bit integers.

The main challenge in the Low-Level Interface development was to identify, locate, interpret and rewrite un-documented packets and functions written in C++ to the Python environment. The main idea is to "simulate" the actions of the human player within the game by using the Python script based on TMW client implementation code. Starting with the most basic functions such as character creation, server login request, server connection, character choice, etc., the interface progressed to imitate in-game functions such as moving the character on the map, attacking other

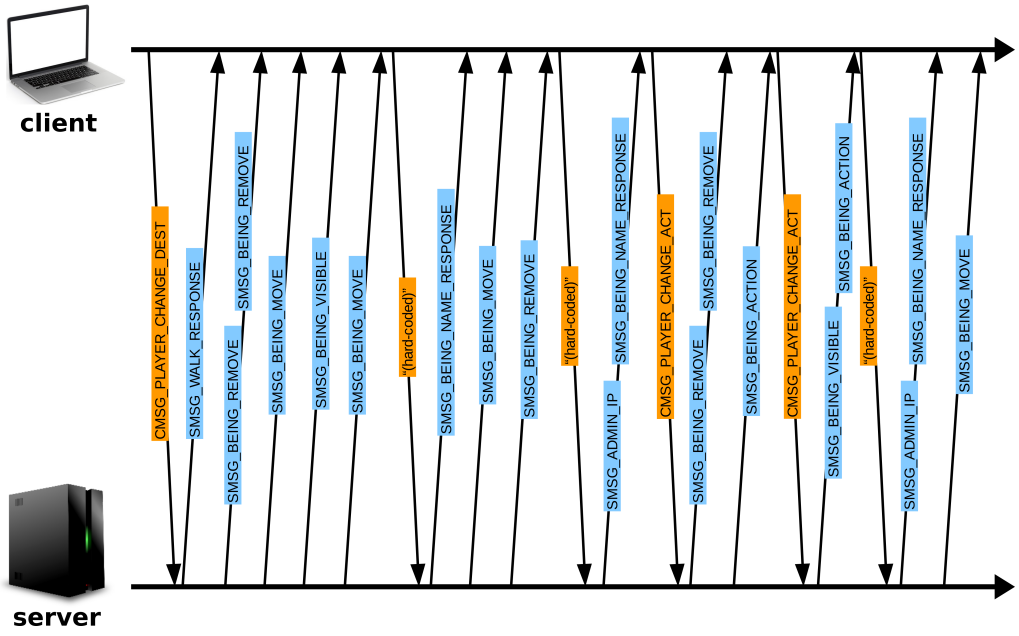


Figure 4.4: An example of message flow between a client and the server

Offset	Length	Contents
0	2	Packet ID
2	2	Packet Length (47 + (32 * ServerCount) + 0)
4	4	Session ID (Part 1)
8	4	Account ID
12	4	Session ID (Part 2)
16	4	*Unused* - Always set to 0x00000000
20	26	*Unused* - Currently set to account.lastlogin for some reason.
46	1	Character Sex
		WorldInfo Sub-packet (Repeated for each world)
X+0	4	Address
X+4	2	Port
X+6	20	Server Name
X+26	4	Online Users (Count)
X+30	2	*Unused*

Figure 4.5: An example of a simple, well documented packet [48]

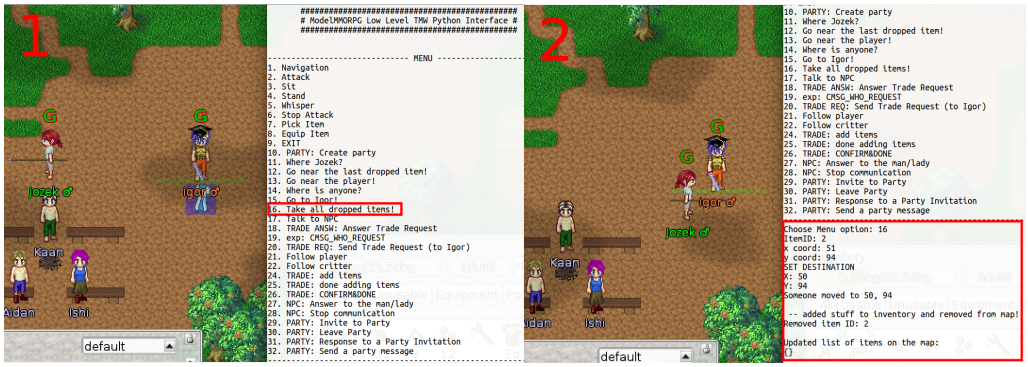


Figure 4.6: Python client: collecting items on the map

creatures, picking up and equipping items, chatting with other players, creating and joining/leaving parties, following other players, trading, etc.

Figure 4.6 shows a sample session of the implemented interface in which the current player picks up an item.

The logic of the Python client is detailed within four key Python classes, namely:

- Character - manages all relevant data of a TMW character;
- Connection - manages a connection to the three TMW servers and provides a low level interface to control a character (moving, attacking, creating parties, chatting, taking items, etc.);
- Packet - manages data about a given packet; interprets packets received from the server;
- PacketBuffer - deals with incoming packets from a given socket server.

Packets are implemented as Python dictionaries, which store hexadecimal codes of packets as keys, and packet length and packet name as values in tuples.

```
PACKETS = {
0x0061 : (50, 'CMSG_CHAR_PASSWORD_CHANGE' ),
0x0062 : (3, 'SMSG_CHAR_PASSWORD_RESPONSE' ),
0x0063 : (-1, 'SMSG_UPDATE_HOST' ),
0x0064 : (55, '(hard-coded) 1' ),
// etc ...
}
```

Core player manipulations are implemented in *Connection* Class methods, such as the following:

Listing 4.2: Example interface method

```
def takeAllDroppedItems(self):
  for key, value in Packet.droppedItemDict.items():
    itemID = key
    debug( "ItemID: %d" %itemID )
    x = value[1]
    debug( "x coord: %d" %x )
    y = value[2]
    debug( "y coord: %d" % y )
    c.setDestination(x-1, y, 2)
    time.sleep(2)
    c.itemPickUp(itemID)
    time.sleep(0.5)
```

The listing illustrates an example of a Python client method *takeAllDroppedItems* which allows an agent to pick up all items that were recently dropped possibly by some monster that has just been defeated. The method makes use of two other methods: *setDestination* which allows moving the character in a 2D space and *itemPickUp* that allows picking up items which are within reach.

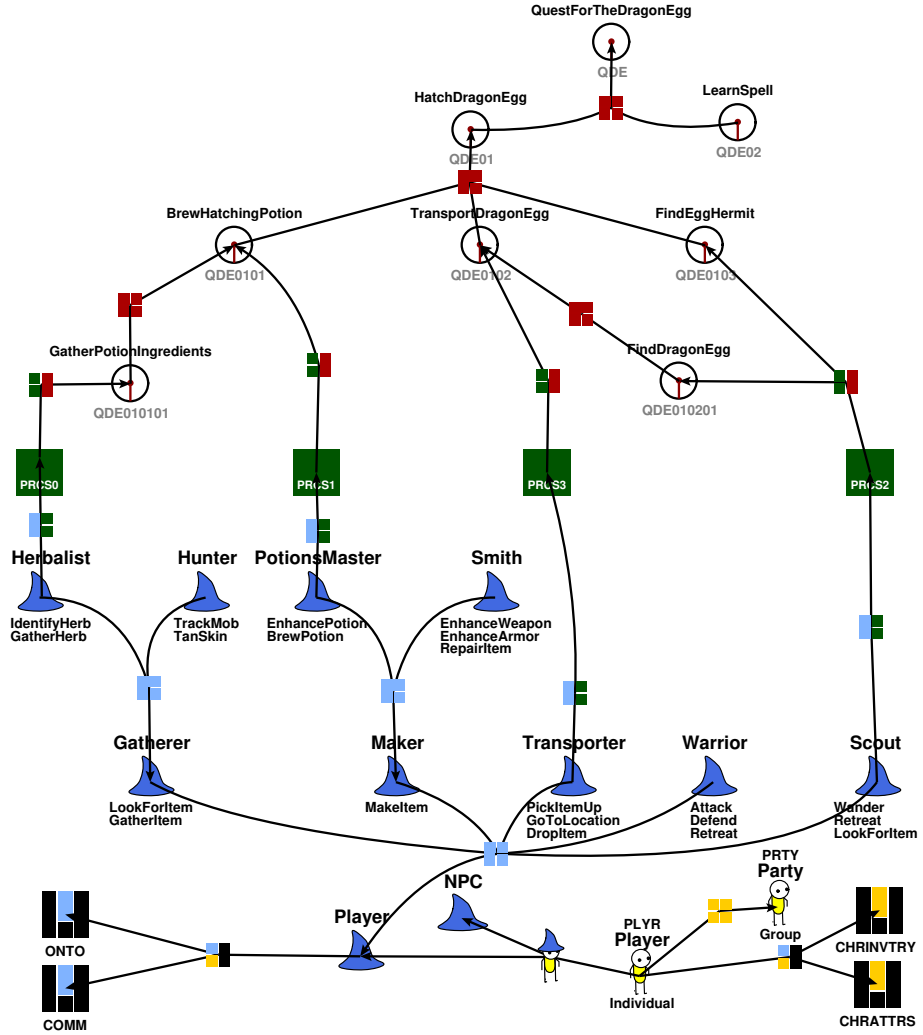


Figure 4.7: Model of the Quest for the Dragon Egg example in TMW domain

4.5 Example Models

4.5.1 The Quest for the Dragon Egg

The model developed within this example is based on the quest developed in early stages of the ModelMMORPG project: *The Quest for the Dragon Egg*. The quest was developed for TMW, and features several NPCs, various items, and demands that players use different kinds of actions and utilize socialization features of the game.

The quest requires the player to seek the *Dragon Egg* item in one of three random possible locations in *The Mana World*, retrieve the egg, hatch it using the brewed *Hatching Potion* with the helping hand of the *Hermit* NPC, and visit the *Arch Wizard* NPC to finally receive the ultimate prize of the quest – a spell to invoke a powerful dragon to their side.

Finishing the quest is not as straightforward as it seems from this high level description though,

as many additional constraints are introduced. The quest is valid for 24 h of real-world time only, i.e. the egg is rendered useless 24 h after being created in the world. The egg is guarded by many violent monsters. Once the *Dragon Egg* item is picked up by a player, the only way to transport the item to the *Hermit* NPC is by the constant cooperation of three players. Furthermore, the *Hatching Potion* item is difficult to create, since the ingredients are not easily found. Once the egg is hatched, the given player must visit the *Arch Wizard* NPC in order to finally complete the quest.

Some of the above stated sub-tasks (e.g. potion hatching, egg finding, egg taking) implicitly require various roles in order to be successfully finished. The described quest, and roles necessary for its completion, along with the individual player class, are modeled using the current version of the described modeling tool as shown in Fig. 4.7.

Individual player-agents are instantiated from the definition of an individual player shown as stickmen in Fig. 4.7. Players can form a group of players, called a party in MMORPG domain, and each player has access to some individual knowledge, shown to the right of the modeled player. Every player agent plays one of the two high level roles: Player or NPC. When playing the Player role, the individual player can play any of the available roles, shown as blue hats in Fig. 4.7. Each role can access some organizational knowledge, shown in the lower-left corner of Fig. 4.7. Additionally, each role has some defined actions, shown to the right of each of the roles in Fig. 4.7 (e.g. role Warrior has actions Attack, Defend, and Retreat), that can be combined into processes, shown as green squares in Fig. 4.7, that can be used to successfully complete some tasks (shown circular in Fig. 4.7). A quest, visible as the topmost element in Fig. 4.7, is divided into simpler sub-tasks that make the original quest reachable by the agents of the system.

The generated code of the modeled system is shown in the following listings. Listing 4.3 shows generated code for player agent class using the *tabula rasa* approach, i.e. every player starts only with a behavior that enables them to change their role and thus gain actions and personal features.

Listing 4.3: Generated code for SPADE Player agent

```
import spade
from RoleBehaviours import *

class OrgUnit45Player(spade.Agent.Agent):
    class ChangeRole(spade.Behaviour.OneShotBehaviour):
        def _process(self):
            print 'Player: behaving ChangeRole'

    def _setup(self):
        self.addBehaviour(self.ChangeRole(), None)
```

Behaviors that can be used by playing a particular role are described in Lst. 4.4. Only essential elements of a behavior (i.e. an action named using SPADE naming scheme) are presently generated by the generating feature of the metamodel. Finally, agents are instantiated using SPADE as shown in Lst. 4.5.

Listing 4.4: Excerpt from generated code for SPADE behaviors

```
class PickItemUp(spade.Behaviour.OneShotBehaviour):
    def _process(self):
        print 'Transporter: behaving PickItemUp'
...
class BrewPotion(spade.Behaviour.OneShotBehaviour):
    def _process(self):
        print 'PotionsMaster: behaving BrewPotion'
```

```
...
class Attack(spade.Behaviour.OneShotBehaviour):
    def _process(self):
        print 'Warrior: behaving Attack'
```

Listing 4.5: Generated code for SPADE agent instantiation

```
import spade
from RoleBehaviours import *
from OrgUnit45Player import *
if __name__ == "__main__":
    agent1 = OrgUnit45Player("OrgUnit45Player1@127.0.0.1","s")
    agent1.start()
```

These generated templates can now be extended using application specific programming. For example the behaviour PickItemUp from Lst. 4.4 could be extended with high-level and low-level interface code in a way similar to the following:

Listing 4.6: Generated code extended with high- and low-level interface code

```
class PickItemUp(spade.Behaviour.OneShotBehaviour):
    def _process(self):
        print 'Transporter: behaving PickItemUp'
        for r in self.myAgent.ask( 'position(X,Y),itemNear(X,Y,Item)'):
            self.myAgent.itemPickup( r[ 'Item' ] )
        sleep( 1 )
```

As one can see in Lst. 4.6, the behaviour firstly queries the current agent's knowledge base for near items. Then it uses the agent's itemPickup method to pick up one item after another.

Use of the metamodel is best explained using an example. Simple example from TMW domain, generally applicable to MMORPGs, modeled using the metamodel will include players, several roles, and two quests. One quest consists of several tasks: the player has to travel to a specific place in the in-game world, and kill 5 monsters there. In order to successfully finish the other quest, the player has to go to go to the Mithril Mine, fight the guardian monster, mine some of the ore located there, and take it back to the quest-giver. Using the set of all the actions available in TMW game, several roles are created that will provide the players with necessary actions. Each of the roles contains several action sets (processes) that will help the player playing the specific role achieve a particular task of a quest. The described situation is shown using the metamodel in Fig. 4.8. Individual agents (yellow) can group into parties (yellow as well). Every player has access to some individual knowledge about their basic statistical values (e.g. skill levels, character traits, etc.), and can play any of the possible roles (only one at any given point in time). Three roles (blue, i.e. Smith, Warrior, and Scout) offer players distinct sets of actions. These actions are grouped into processes (green) that allow the players playing given roles to complete some of the tasks (red) that are a part of one of the available quests (red as well). Roles have access to some basic knowledge about the modeled system, in form of an ontology.

Using the generator feature of the modeling tool, six files are generated: `TheSystem.py` containing the core of the system instantiating SPADE agents (Lst. 4.7), one file (excerpt in Lst. 4.8) per player element in the model containing essential agent code (individual agents are generated as plain, i.e. the only behavior they possess at this stage of the metamodel is intended for changing roles), `RoleBehaviours.py` which lists all the actions available to the modeled roles of the system (Lst. 4.9), one agent file, `DirectorAgent.py`, for an unmodeled agent that answers agents' queries about which roles are most suitable for which tasks (Lst. 4.10, and another agent

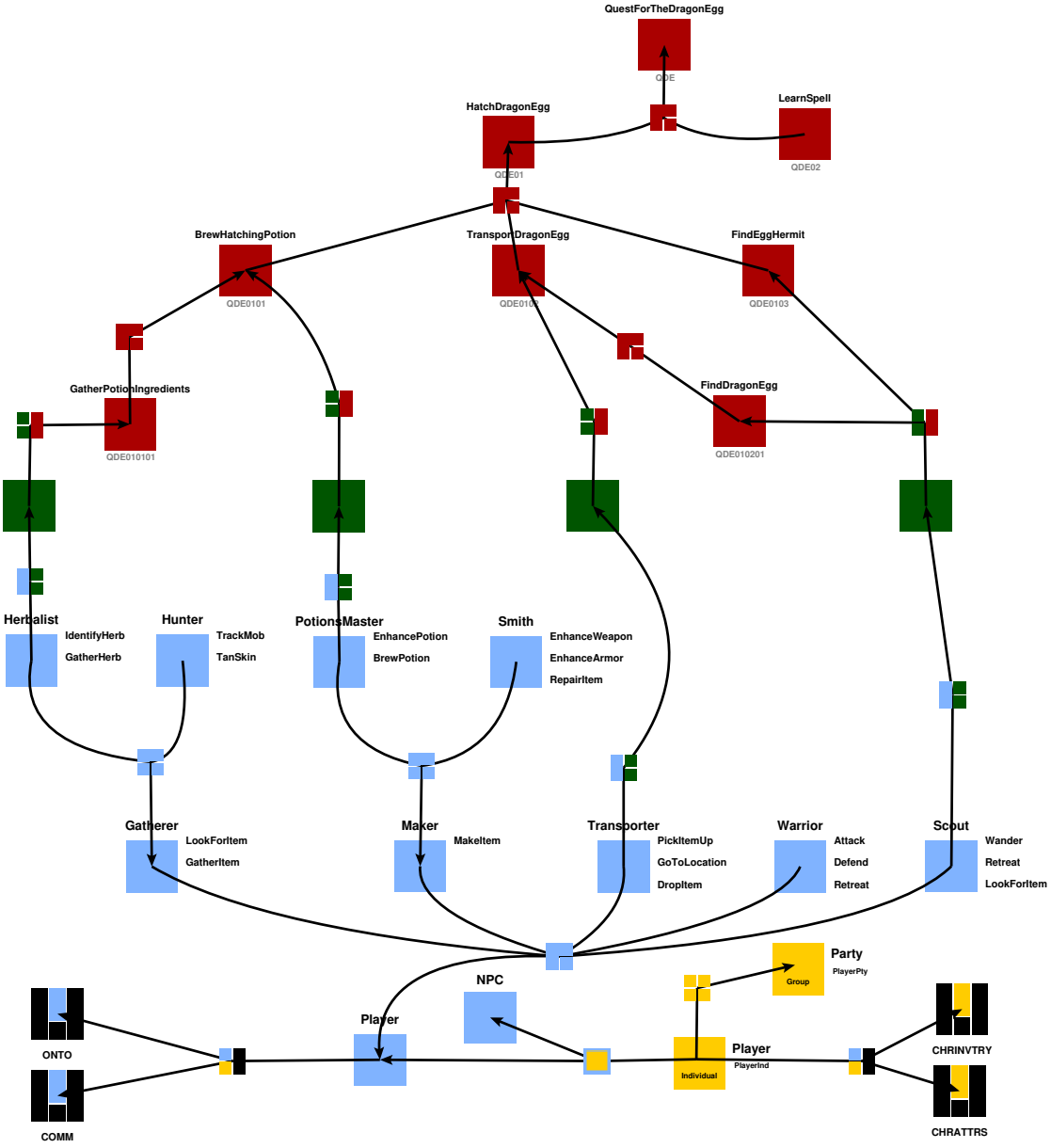


Figure 4.8: Model of an example in TMW domain

file, `TeacherAgent.py`, for another unmodeled agent which is expected to teach agents available actions (SPADE behaviors) and which role they can be performed with (Lst. 4.11). The unmodeled agents are set to use internal SPADE knowledge base and reasoning to help other agents. The main motivation for introduction of the unmodeled agents is a decentralized knowledge base utilizing agents. It is important to note here that an organizational unit element in the metamodel (yellow in Fig. 4.8 represents something that can be considered a class of individual agents, i.e. not instantiated individuals, but the most general form of an agent that will gain its specific features through role-playing.

Listing 4.7: Generated SPADE code showing initialization of individual agents for the modeled system

```
if __name__ == "__main__":
    agent0 = DirectorAgent("DirectorAgent0@127.0.0.1","s")
    agent0.start()
    agent1 = TeacherAgent("TeacherAgent1@127.0.0.1","s")
    agent1.start()
    agent2 = OrgUnit44Player("OrgUnit44Player2@127.0.0.1","s")
    agent2.start()
```

Listing 4.8: Generated code for SPADE Player agent

```
class OrgUnit44Player(spade.Agent.Agent):
    class ChangeRole(spade.Behaviour.OneShotBehaviour):
        def _process(self):
            print 'Player: behaving ChangeRole'
    def _setup(self):
        self.addBehaviour(self.ChangeRole(), None)
```

Listing 4.9: Excerpt of generated code for SPADE behaviors in the library of Role actions

```
class ChangeRole(spade.Behaviour.OneShotBehaviour):
    def _process(self):
        print 'I would like to change...'

class Mine(spade.Behaviour.OneShotBehaviour):
    def _process(self):
        print 'Smith: behaving Mine'
```

Listing 4.10: Excerpt of generated code for SPADE Director agent

```
class DirectorAgent(spade.Agent.Agent):
    class FindSuitableRole(spade.Behaviour.OneShotBehaviour):
        def _process(self):
            print 'DirectorAgent: behaving FindSuitableRole'
    [...]
    def _setup(self):
        self.addBehaviour(self.FindSuitableRole(), None)
        [...]
        self.configureKB('SWI', None, 'swipl')
        self.addBelieve('canReachGoal(Scout,GoToSteamyGlade)')
        self.addBelieve('canReachGoal(Scout,GoToMithrilMine)')
```

```
        self.addBelieve('canReachGoal(Warrior,KillRedGnomes)')
```

Listing 4.11: Excerpt of generated code for SPADE Teacher agent

```
class TeacherAgent(spade.Agent.Agent):
    class FindSuitableBehaviours(spade.Behaviour.OneShotBehaviour):
        def _process(self):
            print 'TeacherAgent: behaving FindSuitableBehaviours'
    [...]
    def _setup(self):
        self.addBehaviour(self.FindSuitableBehaviours(), None)
        [...]
        self.configureKB('SWI', None, 'swipl')
        self.addBelieve('hasAction(Scout,GoToLocation)')
        self.addBelieve('hasAction(Scout,PickupItem)')
        self.addBelieve('hasAction(Scout,DropItem)')
```

4.5.2 Sorfina's Tutorial Quest

This example is focused on automated solving of an initial tutorial quest of TMW in order to show some of the most significant elements of the framework. The process of testing a part of the game using the framework consists of a number of steps. Firstly, the test is modelled using a graphical modelling language as elaborated before. Besides of modelling structure, processes, roles, objectives, and other organizational features, the language also allows modeling organizational dynamics through temporal logic and graph grammars as the example shows on Fig. 4.9.

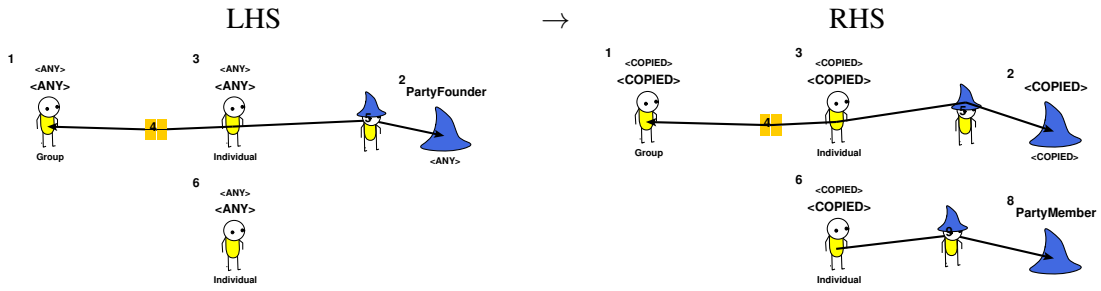


Figure 4.9: Example graph grammar rule related to joining a party inside TMW

The modeling tool can be used for generating an application template to be connected with a special game plug-in developed for the purpose of this framework. The plug-in consists of the low-level interface (dealing with the actual network level protocol establishing a connection to the game servers), and the high-level interface (a SPADE agent template connecting the low-level interface to a knowledge base [KB] and planning system). The high-level interface is basically a BDI agent having various behaviours including updating of its KB (beliefs) based on actual percepts provided by the low-level interface, updating of objectives and quests to be solved (desires), as well as obtaining plans on how to solve a given quest (intentions). For example, the method that selects the next objective (quest) to be solved is shown in Lst. 4.12.

Listing 4.12: Example method – selecting the next objective

```
def selectObjective( self, objectives ):
 ''' Select most relevant objective (quest) to be solved next '''
```

```
query = "sort_quests( '%s' ), quest_no( NPC, '%s', Name, No )." % ( self.
    ↪ avatar_name, self.avatar_name )
quests = self.askBelieve( query )
if quests:
 next = sorted( quests, key=lambda x: x[ 'No' ] )[ 0 ][ 'Name' ]
 self.say( 'My next objective is quest: ' + next )
 return next
```

Additionally, the agent has the ability to act on the TMW environment (again through the low-level interface) and check if its actions were successful. The last step, after connecting the application template and the plug-in is to start the tests and analyze them.

The model of the initial TMW quest, given by an NPC called Sorfina is presented in Fig. 4.10. The player has to move slightly within the confines of the first gaming area, have a conversation with two NPCs, equip some items, and leave the area. Using the modeling tool, this quest can be broken into several tasks.

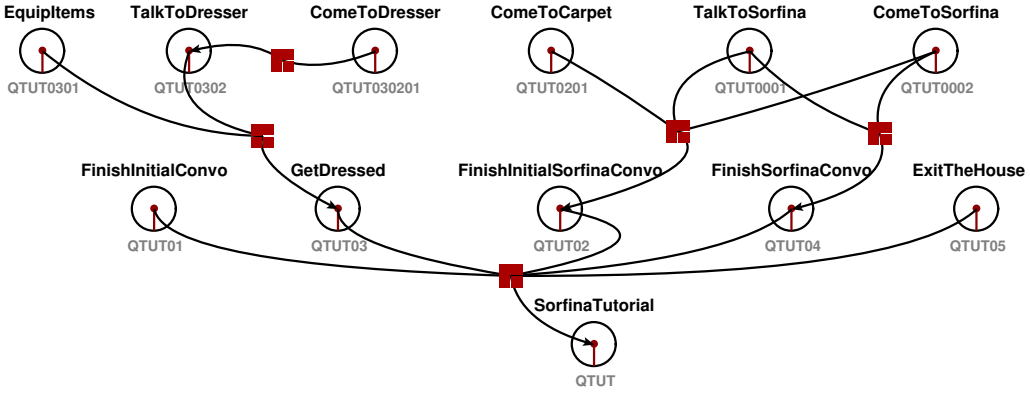


Figure 4.10: Tutorial quest breakdown into tasks

These tasks are reflected as actions inside plans for solving the given quest or part of the game. Actions have their preconditions (for example, some other quests have to be solved in order to attain the current one) and postconditions (for example and action `goToLocation( Map, X, Y )` has the postcondition `location( Map, X, Y )`).

For the sake of this example we have implemented a plan for Sorfina's tutorial quest.[4] The algorithm of the planner is quite simple in this case: firstly the planner sorts all given quests by priority of solving. In case the agent hasn't got any current quest, a default random walk quest is assigned to it, in order to find an NPC or other means to acquire a quest. Then, the quest with the highest priority for which all preconditions are met, is selected for solving. The planner starts the quest and derives one action after another to be enacted by the agent. The agent then fulfils the action and checks if the action's postconditions have been achieved by updating its KB. If an action fails, the planner returns the quest into the list of waiting quests and starts over. If all actions have been achieved successfully (e.g. all envisioned postconditions have been met) the quest is marked as achieved and the planner starts over.

We have tested the implemented quest on a dedicated TMW server with multiple parallel agents trying to solve Sorfina's quest. Fig. 4.11 shows an example session in which 8 agents are solving the quest. While the times for solving the quest have differed (mostly due to random walks of players) all agents were able to solve the quest.

[4]The source code of the testing framework and modelling tool are available on GitHub at `https://github.com/tomicic/ModelMMORPG` and `https://github.com/Balannen/LSMASOMM` respectively.

Figure 4.11: A screenshot of multiple player agents trying to solve Sorfina's quest

4.6 Simulation Experiment

In order to extend the implemented agents with organizational capabilities we have additionally implemented organizational roles which represent behaviours which, when enacted, introduce new behaviours to an agent. To test these capabilities an agent based simulation experiment, which is socio-economic in nature, has been conducted. The main organizational behaviour players in The Mana World (TMW) can enact are party related behaviours including but not limited to creating parties, inviting other players to parties, joining parties etc. We have implemented three organizational roles related to parties:

Leaders : one of their main objectives is to create a party and invite as many players are possible to join in.

Extremist followers : join the first party they are invited to and never leave it.

Opportunists : join the first party they are invited to and then choose randomly to join other parties with a slight favor for parties with more players in it.

The experiment has been set up as follows: n agents will be created in regular intervals Δt. They will play for a predefined amount of time t after which the emerged social network will be analyzed. The social network is a bipartite network in which agent/player nodes are connected to party nodes and vice versa. In our case we have chosen that $n = 60$, $\Delta t = 30$ seconds and $t = 20$ minutes which is roughly enough time to solve the first few quests on the initial Candor map of TMW.

We have implemented three variants of this experiment depending on the distribution of organizational roles the simulated agents enact and the order in which they are introduced into TMW:

1. 30-0-30 (30 extremist followers, 0 opportunist, 30 leaders)
2. 0-30-30 (0 extremist followers, 30 opportunist, 30 leaders)
3. 20-20-20 (20 extremist followers, 20 opportunist, 20 leaders)

Variants with only one type of agents woudn't make any sense since in none of them would be any social interaction.

During all experiments detailed logging data has been collected with special regard to party related agent actions (e.g. creation of party, invitation to join a party, joining a party, leaving a party etc.). Additionally all agents' knowledge bases (e.g. their "state of mind") at the end of each simulation have been collected as well.

Before conducting the experiments we have posed the following research questions:

1. How do network statistics change over time and how agent role distribution influences these

changes?

2. Does time influence node statistics, e.g. do agents created earlier have more connections and better influence?
3. Does success rate (e.g. solving of quests) influence node statistics, or simply put do more successful agents have better influence?

In order to make it easier to start the experiments a script was implemented that starts each of the above given scenarios on our development server. The script starts a given number of agents that start as new players in TMW each enacting a role given by the script. The enaction of roles is then responsible for the behaviour of each agent in regard to organizational capabilities.

4.7 Results & Analysis

For each simulation scenario we have analyzed the social network that has emerged between the simulated players. In particular, since we are dealing with a bipartite network of agents and parties, the node degree of each party (and consequently its leader) has been analyzed in more detail. Figure 4.12 shows the dynamics of node degrees for each party during the first (30-0-30) simulation.

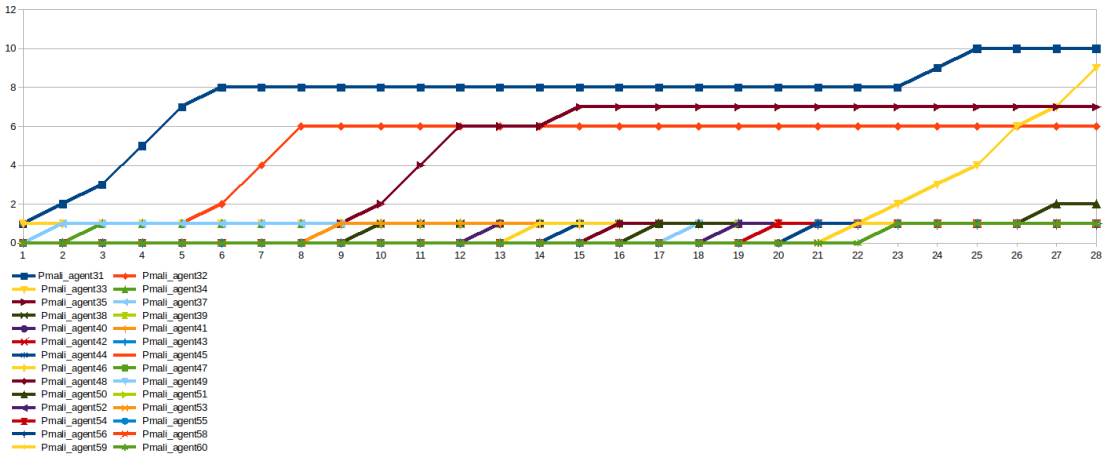


Figure 4.12: Node degrees for the first simulation

As one can see from the figure in most cases leader agents that were introduced earlier had a better chance of acquiring followers, and since there are no opportunists in this scenario, their node degree was constantly growing. There is only one exception of a leader being able to acquire a greater number of followers even though being started later during the simulation. On more detailed inspection this exception has occurred due to physical constraints of the game. Agents were implemented in a way that they can only invite other agents if they are in line of sight and not more than a certain distance away. This agents was the first to leave a house and get to another part of the map where he was able to invite all nearby followers, thus gaining degree in a later stage of game advancement.

To answer the question if advancement through the game correlates to the degree of a (leader) agent, we have calculated the correlation coefficient between the degree of an agent and the number of solved quests. In this case $r = 0.20395$ indicates no particular correlation. Additionally, we also calculated r for node degree in connection with the passed time to see if there is any relation. In that case $r = -0.57773$ indicates a moderate negative relationship, meaning that leader agents started earlier had a better chance of acquiring followers.

Figure 4.13 shows the dynamics of parties' node degrees during the second simulation (0-30-30) in which 50% of the nodes were opportunists and 50% leaders. Due to the behaviour of opportunists

a certain amount of randomness is introduced into the graph.

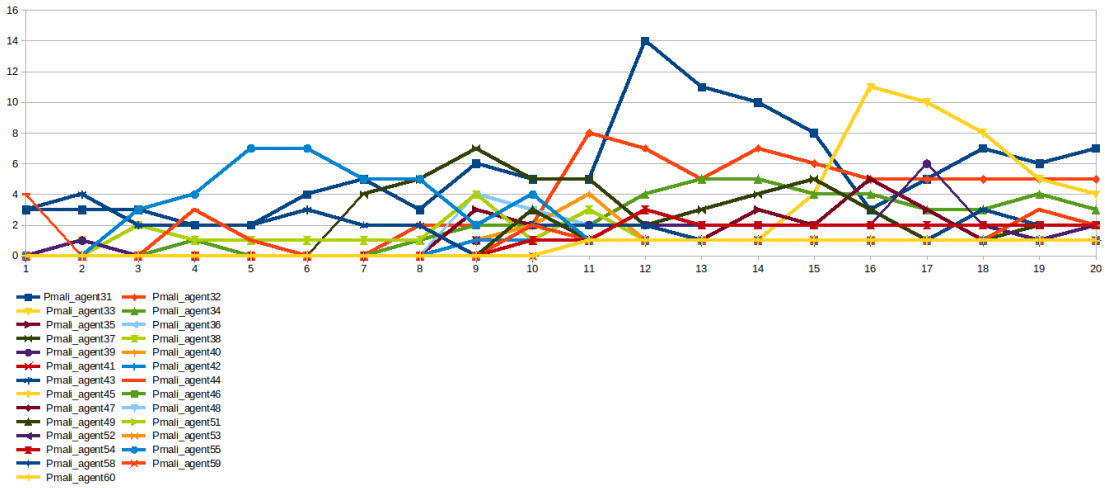


Figure 4.13: Node degrees for the second simulation

There seems to be a moderate positive correlation between progression through the map (number of solved quests) and the actual degree of a (leader) node ($r = 0.63950$). Additionally, there is a moderate negative correlation between time and node degree ($r = -0.60867$).

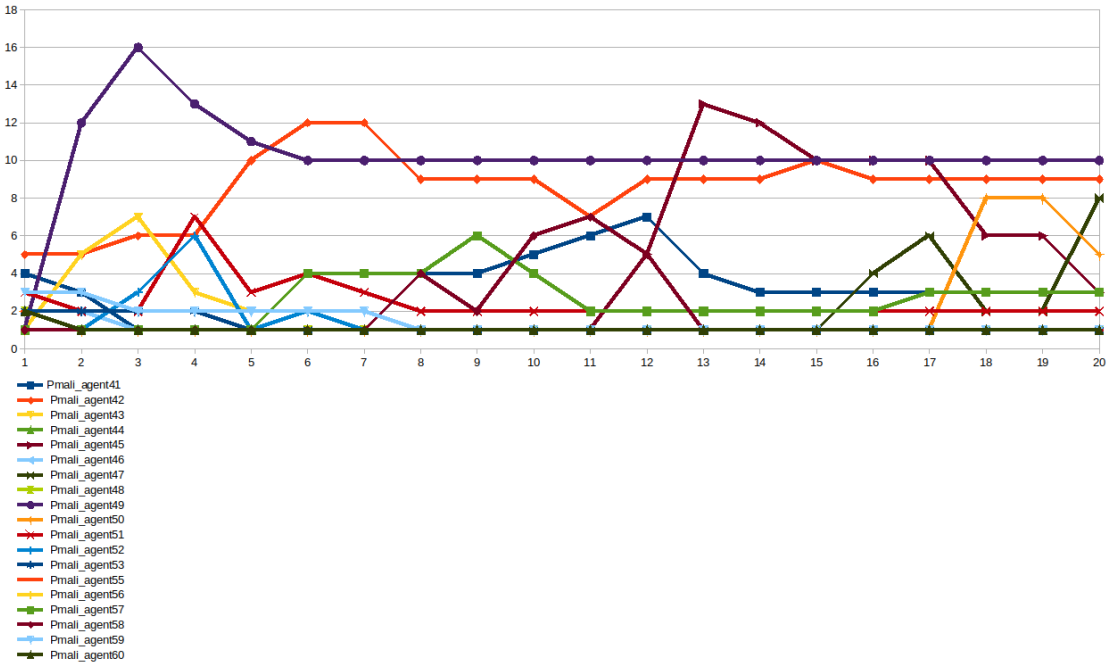


Figure 4.14: Node degrees for the third simulation

Figure 4.14 shows the dynamics of node degree for the third and last simulation scenario in which we had $\frac{1}{3}$ of extremist followers, $\frac{1}{3}$ of opportunists, and $\frac{1}{3}$ of leaders. As one can see there is a smaller amount of randomness as in the previous case since the number of opportunists is lower. Interestingly, except for the two obvious cases, most leader agents started earlier had bad performance at the end of the simulation in regard to node degree. This seems to be due to the case that these two extreme cases actually were able to attract most of the extremist followers early in

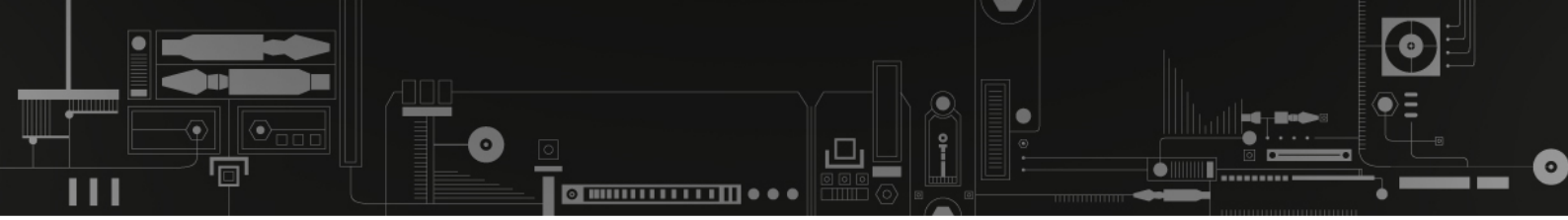
the game, thus also attracting opportunist.

This is confirmed by the correlation coefficient between time and node degree $r = -0.36261$ which shows only a weak downhill linear relationship. Similarly the correlation between progression through the game and node degree is also weak negative ($r = -0.37427$) meaning that in this case actual progression negatively influences a (leader) agents ability to attract followers.

4.8 Conclusions

In this chapter we have reported on the current state of development of a model driven and agent based MMORPG development platform. This platform is part of a broader scope LSMAS development framework which has been specialized with a plug-in system that allows us to model MMORPG related scenarios.

We have shown how the platform and plug-in can be used to model specific scenarios and have even conducted three simulation experiments that included organizational roles of agents that were enacted by the respective agents that acquired them. The simulations have shown interesting social aspects of an MMORPG which indicates that the platform can be used for both model-driven development of (technical) systems and agent based simulation modeling.



hrzz
Hrvatska zaklada za znanost

foi
SVEUČILIŠTE U ZAGREBU
FAKULTET
ORGANIZACIJE I
INFORMATIKE
VARAŽDIN

lab

5. 3rd Large-Scale Experiment - Smart Cities

5.1 Introduction

According to [22], smart cities ought to be "*the urban center of the future, made safe, secure environmentally green, and efficient because all structures - whether for power, water, transportation, etc. are designed, constructed, and maintained making use of advanced, integrated materials, sensors, electronics, and networks which are interfaced with computerized systems comprised of databases, tracking, and decision-making algorithms*". Thus, smart cities represent large-scale, complex, socio-technical and socio-cybernetic systems which are inherently distributed in terms of interacting components and parallel in terms of interlacing processes. Smart cities are one of the important application domains of the Internet of Things (IoT), in which smart devices of various kinds cooperate to provide better services for citizens [3]. An adequate formalism for modelling such systems is found in agent-based modelling, both in the context of engineering smart technologies, and simulating social behaviour.

Apart from being "smart", for cities to be self-sustainable, they should be able to produce, manage, and consume their resources in an intelligent manner. Herein we will introduce the Smart Self-Sustainable Human Settlements (SSSHS) Framework (which we have already presented in [52]) that has demonstrated its efficiency in prolonging the period of self-sustainability in small scale resource management experiments. SSSHS uses an agent-based approach in which resource producing, storing, and consuming units are represented as intelligent software agents that are able to negotiate resource use, depending on current needs of their users (e.g. inhabitants of a settlement).

In order to make this framework suitable for large-scale settings, herein we will introduce additional techniques bound to organizational modeling of large-scale multi-agent systems (MASs) similar to the foundations which we have outlined in [52]. Large-scale multi-agent systems (LSMASs) need to be organized in a robust and scalable manner that will allow software agents in such systems to function and adapt to possibly complex environments they have to face. To establish such a complex organization, an adequate normative system with an organizational structure consisting of organizational roles has to be put in place which agents can enact depending on current environmental and system-specific circumstances. Herein we will introduce a dynamic

hierarchical structure based on a recursive definition of organizational units [36, 37], which represent the building blocks of (agent) organizations.

The rest of this chapter is organized as follows: firstly in section 5.2 we provide an overview of relevant literature. Then in section 5.3 we present the organizational meta-model that is used for modeling a smart city's resource management system in form of an LSMAS. Afterwards, in section 5.4, the SSSHS framework is introduced in more detail. In section 5.5 we provide a detailed description of modelling techniques for smart cities as a self-sustainable LSMAS. In section 5.6 an example scenario is provided and analyzed. In section 5.7 we discuss our findings and in section 5.8 we draw our conclusions and provide guidelines for future research.

5.2 Literature Overview

Organisational models for MASs have been developed based on features of human organisations from their initial phase, yet only recently have researchers become interested in exploring the meaning of organisational features in LSMAS. An interesting overview of the existent models for organisational modelling of MAS in general, although some of those are applicable to LSMAS as well, was given in [9]. The models described in the mentioned paper are evaluated against a set of modelling dimensions for agent organisations. We have presented an improvement of those dimensions, making them more susceptible to the modern ideas which encourage development of LSMAS, presented in the form of a set of perspectives of organisational modelling, in [37]. The organisational metamodel used in this chapter is being built based on these perspectives [13, 14].

Agent-based modeling approaches proved to be adequate for modeling of the IoT domain [4, 39, 50, 52], especially when considering the potential interoperability issues that may derive from heterogeneous set of devices, communication protocols, networks, data formats, etc. IoT systems are, by definition, distributed and intelligent, which is equivalent to most definitions of MAS. In the domain of smart cities, which can be viewed as large-scale IoT systems, the use of agent-based modeling and implementation techniques becomes an even greater necessity. Due to the mere scale of such systems, organizational design of agent organizations seems to be an adequate toolset to approach such systems.

Several recently published papers describe research on smart buildings and smart grid in cooperation with agents and IoT concepts, some of which employ the idea of swarm [23], some deal with user intention models [59], some work on coordination of agents of energy management systems [46], and some are considering energy systems using ontology-based systems of agents [2]. Also, Roscia et al. discuss a model of the Intelligent Distributed Autonomous Smart City (IDASC) by using multi-agent systems and Internet of things [33], and elaborate on an infrastructure and architecture of a multi-agent system for a smart city model, concluding with a conceptual, cross-domain model of a smart city. Clearly, the intention of this chapter differs from already published work in details such as dealing with self-sustainability emerging from individual agents' interaction and their mutual cooperation, and communication which is translated to higher aggregation levels, i.e. groups of groups of individual agents are considered as well.

The IoT initiative has often been referenced in the context of sustainability, energy efficiency and environmental issues, especially in the context of the environmental IoT (EIoT) domain. For example, Vlacheas et al. are considering the key issues that might prevent the Internet of Things from supporting the sustainable development of smart cities: the heterogeneity among connected objects, and unreliability of associated services. The authors propose the cognitive management framework - a framework that allows selection of individual devices (agents) for the specific task to be performed in the given spatio-temporal context in an effort to address those issues [53]. The work presented herein deals with setting up the adequate roles (inside an imposed normative organizational structure), which can then be enacted by specific agents. From this perspective, the work of [53] is complementary, but with a different intention.

An agent-based simulator that attempts to create the dynamic behavior is presented in [25]. The simulator deals with heterogeneous devices that have the producing and/or consuming roles within the simulation, and are able to act in an autonomous manner and collaborate with each other. The main focus of the simulation carried out by the authors was to reproduce a residential scenario of energy consumption, with appliances having only two possible states (ON and OFF). The appliances considered herein, use our proposed SSSHS approach and have the possibility of operating time manipulations together with several modes of consuming operations (delayed, advanced, default, savings, off), which offers more possibilities in terms of managing self-sustainability.

Another approach to observing dynamics of resource exchange (namely renewable electric energy) was described in [29]. The authors introduce the idea of an exchange of electricity and a novel digital currency they called NRGcoin. Their work focuses on the negotiation and exchange process of renewable electricity as the only resource being exchanged, as opposed to mechanics of individual units working with various resources as presented in this chapter. The idea of introducing a special currency for energy exchange, along with specific behaviour of included housing unit agents is noteworthy though, since it presents one of the possible ways of negotiation interaction of various housing units of a system. Furthermore, their model helps in regulating consumption and production, and managing electricity storage.

5.3 Large-Scale Multi-Agent Systems - an Organisational Metamodel

The idea of organization is not a new concept in the world of MAS, and seem to be crucial to LSMAS. The first thoughts about the organisational modelling of a MAS were derived from the domain of human organisations and the set of concepts used to describe organisations of humans. Since humans can be abstracted as agents, as far as their position and significance in an organisation is concerned, and their interaction can be formalised, it is apt to speak of copying features of human organisations to MAS. Furthermore, it is argued [8, 12] that LSMAS benefits from some organisational features and constraints, since having a defined code of conduct can help utilise individual agents' features, strengths and group commitment, along with overcoming their temporal limitations and skill constraints.

There are two dominant perspectives on organisation as a concept in MAS, and subsequently LSMAS, that are mutually not excludable [1, 11]. Organisation can be observed as a concept imposed on the given system (organisation centred MAS), thus being devised by an agent outside of the goal system, and then cast upon it, imposing the authority and the order. This approach constraints the agents of the system using an already devised set of rules and other organisational features. Such an approach can comprise agents that can perceive the given organisation. Furthermore, such agents can influence the organisation to change, if such a change is needed based on the state of the environment and the whole system. Therefore, the agents can change the organisational features of the system, if the proposed change can make it easier to achieve a group goal. On the other hand, emerging organisation is about a bottom-up approach to the idea of organisation - organisational constraints are results of individual agents' interactions. Instead of a given organisation, organisational features of such a system grow based on agents' perception of the environment and their reactions. The emerging organisation is therefore more open to change, and the constraints it introduces on the system are more natural, being derived from the system itself. It is argued that LSMAS benefit the most from the combination of both, slightly favouring the cast-upon approach. Although the emerging organisation approach seems more natural for an LSMAS primarily because of the decentralised feature of a MAS, the imposed-upon organisation allows the agents to act faster [54]. On the other hand, emerging MAS organizations can be more robust and adaptive, since their behaviour depends exclusively on their mutual interactions with the environment they are situated in.

A number of MAS models have tackled the problem of organisation when dealing with artificial

autonomous agents, but there are only a few of them specialising for LSMAS [9]. Therefore, we have proposed a novel LSMAS organisational metamodel featuring human organisational concepts in recent research [13, 14, 37], one that would introduce additional organisational concepts into modelling LSMAS, in particular organisational dynamics, and is currently in an early stage of development. The current results can be used to model a system though, as will be shown in this section using the metamodel as a work in progress.

The metamodel is aiming to introduce an enhanced experience for users modeling LSMAS with organisational features. In its final version, the metamodel is envisioned to support modelling of organisational features including organisational structure, normative elements and organisational dynamics. The idea of the metamodel stems from the research and guidelines we set in [37, 43], where a set of organisation modelling perspectives is presented that is deemed as useful for the modern research of LSMAS. The metamodel is being developed with general use in mind, with intentions of allowing developers of various LSMAS domains to use it effectively. Domains such as the Internet of Things and smart cities can be abstracted using the idea of LSMAS, and are therefore good for being modeled using the metamodel described herein.

The metamodel uses a recursive approach similar to that of holons and holarchy [1], for example the basic element of the metamodel is an organisational unit which represents either an individual agent or a group of organisational units [37]. A similar approach is applied to various other elements of the metamodel (for example roles and objectives).

One of the features of the finished metamodel will allow the user to generate programming code for the modeled system. The generated code will provide the developer with essential code for creating the modeled system based on the metamodel.

Following are the elements included in the current version of the metamodel, aimed at describing an LSMAS:

- Organisational Unit: An organisational unit is the concept that models the actors of the system - individual agents and groups of agents. An organisational unit plays roles, conducts actions and achieves goals following the norms of the system.
- Role: The role element serves as a way of representing norms of the modeled system. Every organisational unit can play a certain set of roles which dictate actions and behaviours available to the given organisational unit and constraints that have to be respected by the organisational unit.
- Individual Knowledge Artefact: Individual knowledge artefacts are the way of modeling knowledge available to individual agents. This knowledge is not universally accessible, i.e. each individual can access their knowledge artefact, unless stated otherwise.
- Organisational Knowledge Artefact: Since roles are elements of the system rather than elements of the individual agents, roles have access to artefacts containing knowledge concerning the system, along with various norms and other organisational features of the modeled system.
- Objective: The objective element is used for modelling various kinds of goals of the system. Objectives are defined recursively as well, therefore an objective can consist of several subobjectives until the basic level is reached.
- Process: Each role provides the organisation unit playing the given role with a set of actions. Various combinations of these actions are designated as processes that serve a purpose of achieving a specific goal. There is a strong connection between objectives, processes and roles, since processes make various roles applicable for specific objectives.

There is a number of modelled elements for connecting the stated class-like elements of the metamodel. These are primarily used to designate various types of relations between the elements included in the metamodel. Various relations are possible, some of which are shown are:

- Being a part of: since one of the more interesting aspects of organisation for this metamodel is

formation of groups of organisational units, a very important connection represents grouping of various elements.
- Available roles for enacting: each organisational unit can, at a given point in time, play a certain role. When it is enacting a role, the organisation unit is enhanced with applicable actions, knowledge and constraints.
- Accessible knowledge artefacts: the general idea is that organisational units can access individual knowledge artefacts, and roles have access to organisational knowledge artefacts, since the first describe individual agent (or group of agents), and the latter are concerned with the system.
- Serving a process: each role has a set of actions at their disposal. A group of actions from one or more roles that can be conducted in order to achieve a particular objective is called a process.
- Relevance for an objective: it is important to specify for which objective each process is useful, since such a description makes it possible for an agent to easily find a role suitable for the obstacle or an objective the agent is faced with.

5.4 The Smart Self-Sustainable Human Settlements (SSSHS) Framework

The basic building blocks of the SSSHS framework's metamodel (not to be confused with the organizational metamodel described above) are depicted in Fig. 5.1. The smart self-sustainable system is composed of individual dwelling units, mutually interconnected in a network infrastructure that allows both resource and data/message exchange between units. Individual dwelling units can initiate communication with other dwelling units upon detecting events that point either to resource depletion within their own subsystems, or to resource production overflow.

Individual dwelling units are composed of several agent types, each of them playing a specific role. Such roles include producer role, dealing with the production of resources according to the input data distribution; consumer role, dealing with the consumption of resources according to consumer's inner specifics; storage roles, dealing with storing resources, communication, triggering the self-sustainability mechanisms, etc. Each storage agent deals with only one resource type, and is connected to other storage agents of the same resource type in other dwelling units within the self-sustainable system.

The framework facilitates modeling and simulation (through a software platform we have developed in Python) of human settlements in several key phases:

1. Defining temporal granularity of the simulation (a second, a minute, an hour, a day, 4 days, 2 weeks, a month, etc.), i.e. the smallest time unit that enables changes in the simulation namespace.
2. Defining the total duration of the simulation.
3. Implementing scenario context by instantiating model's entities (producer, consumer, storage agents).
4. Defining the inner states of the included agents (setting the relevant parameters of agents according to available data and context, thus tuning their behaviours).
5. Implementing a multi-agent system by defining mutual relationships of agents, thus creating a network of agents that can communicate, negotiate, make offers, etc.
6. Setting the relevant input data from the environment that would affect the included agents and their behaviour during the simulation run.
7. Initiating the simulation run, analyzing the results produced by the framework, tweaking and optimizing the model according those results.

The modeling process from the modeler perspective is depicted in Fig. 5.2.

The framework proved to be able to prolong the self-sustainability of the settlement by using its inner resource management mechanisms [49, 50, 51, 52].

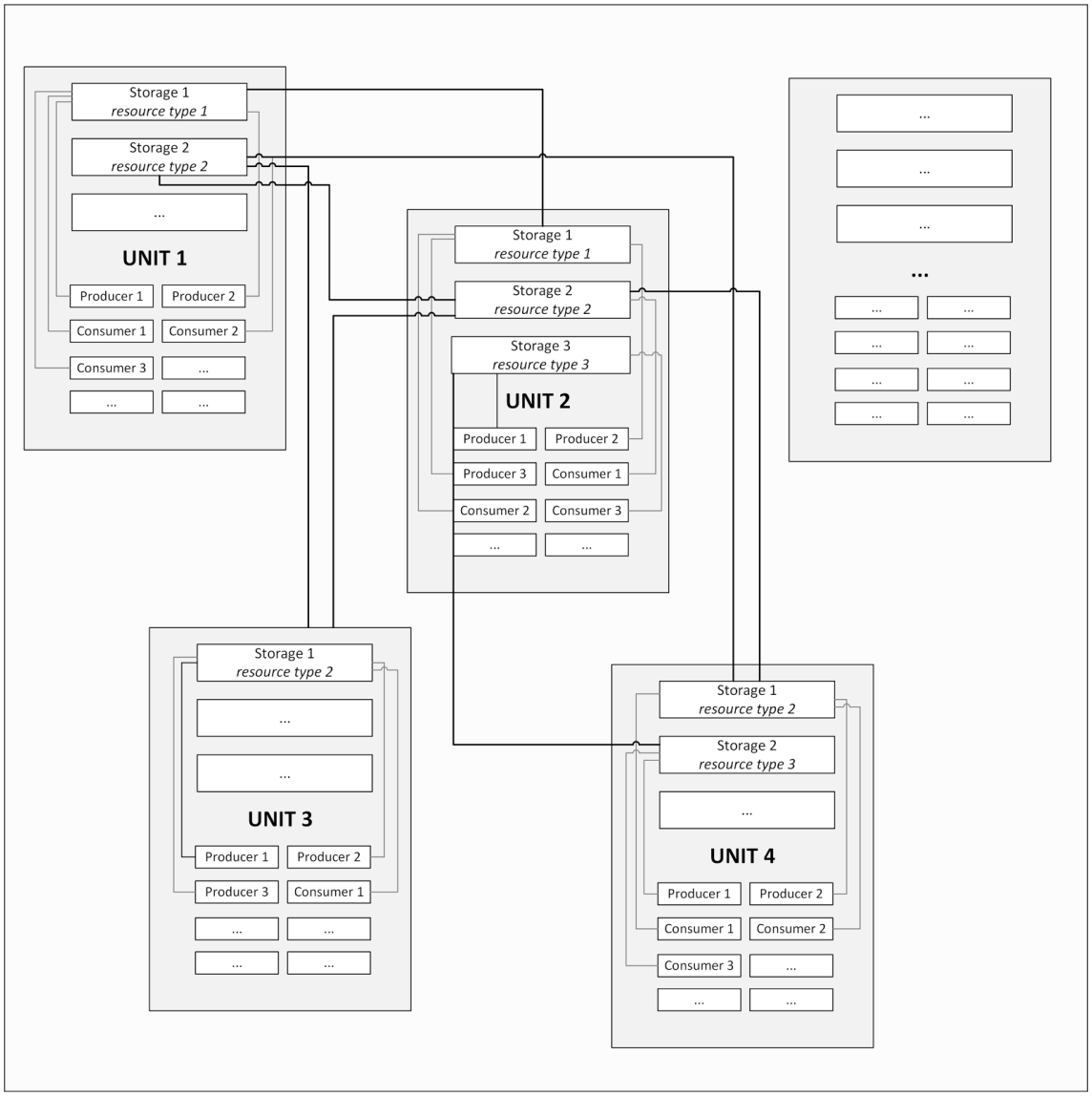


Figure 5.1: Basic building blocks of the SSSHS metamodel, as shown in our previous work [51]

Dwelling units can manage one or more resource types, thus enabling the framework to simulate one or more resources simultaneously in the same simulation run, without explicitly limiting the upper number of simulated resources.

Self-sustainability is in this context described as a binary property of the system; if the system is able to cover resource demands in any given moment with its own production capacities within the observed time period, such a system is considered self-sustainable for this time period. Contrariwise, if the system is not able to maintain this resource autonomy, it is considered not self-sustainable, and thus unable to function off-grid. If the simulation results show that the latter is the case, further analysis of the system and model optimizations are needed in an effort to make the system self-sustainable. The SSSHS framework offers verbose simulation output data which might point to the key factors that impede self-sustainability.

Preserving self-sustainability in any given moment within the simulation runtime is facilitated by the SSSHS framework's self-sustainability mechanisms. Two major scenarios are considered when dealing with the intermittent resource production and consumption.

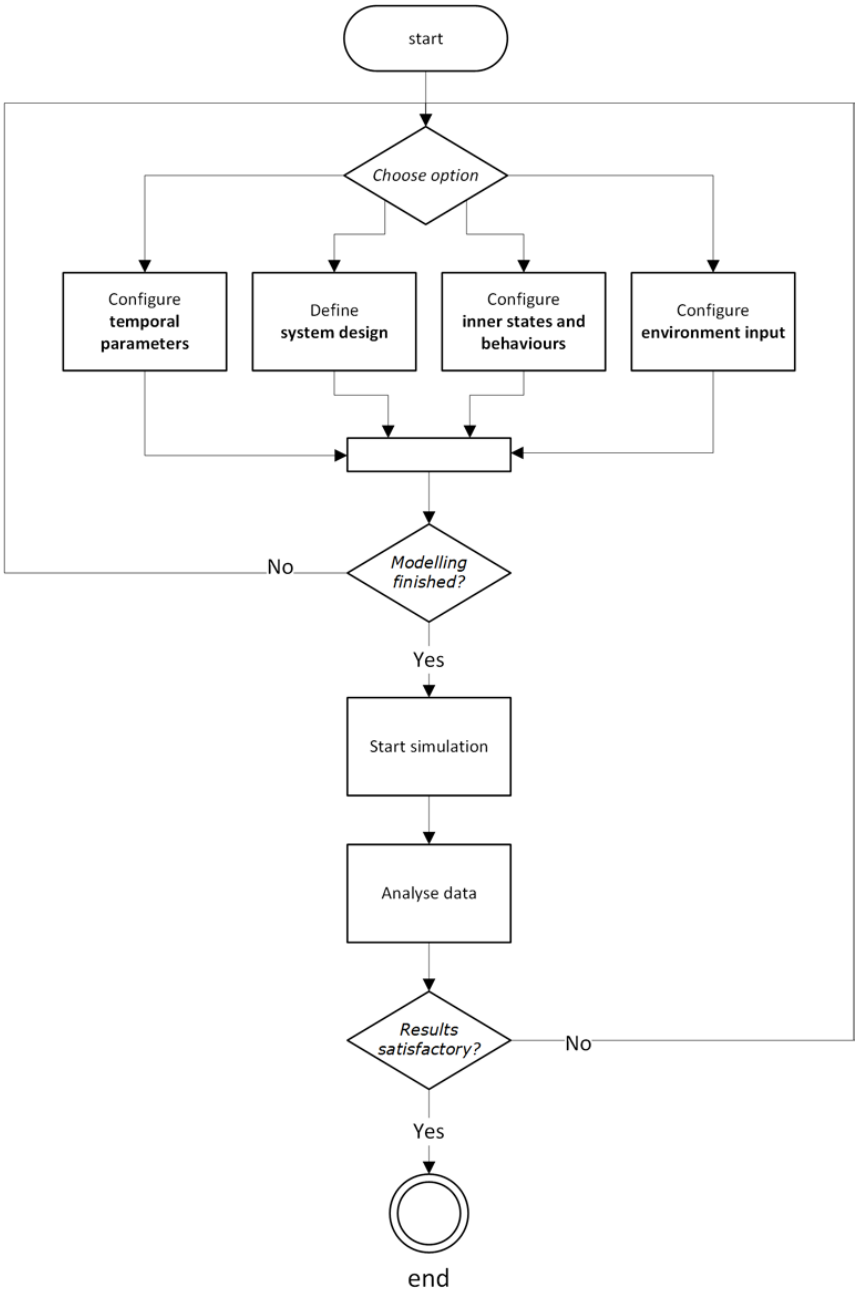


Figure 5.2: Basic building blocks of the SSSHS metamodel, as shown in our previous work [51]

In the first scenario, production rates may overflow the storage capacities within a dwelling unit, risking a loss of resources because of the inability of the unit to store this excess of produced resources. This same excess might be critically needed in one or more of the other units, or simply distributable among other units, with priority queue based on the individual's storage levels for the given resource. The SSSHS framework triggers the upper resource level alert when detecting a possible resource overflow, and mechanisms such as advancing consumption operating times, restoring default consumption rates, offering and distributing surplus of resources to other units, are able to handle such events, preventing the overall loss of resources.

In the second scenario, dwelling units may lack the production rates that would handle the increasing demand for a resource. In this scenario, a dwelling unit might completely run out of a

resource, rendering the system as not self-sustainable. The SSSHS framework is equipped to deal with such scenarios by utilizing its lower threshold mechanisms such as activating the consumers' savings modes, manipulating the consumers' operating times, or initiating a negotiation process that has the potential to result in a resource transfer from other dwelling units. Figure 5.3 depicts the self-sustainable mechanisms' activities managed by the SSSHS framework based on the detected simulation events throughout the main system roles.

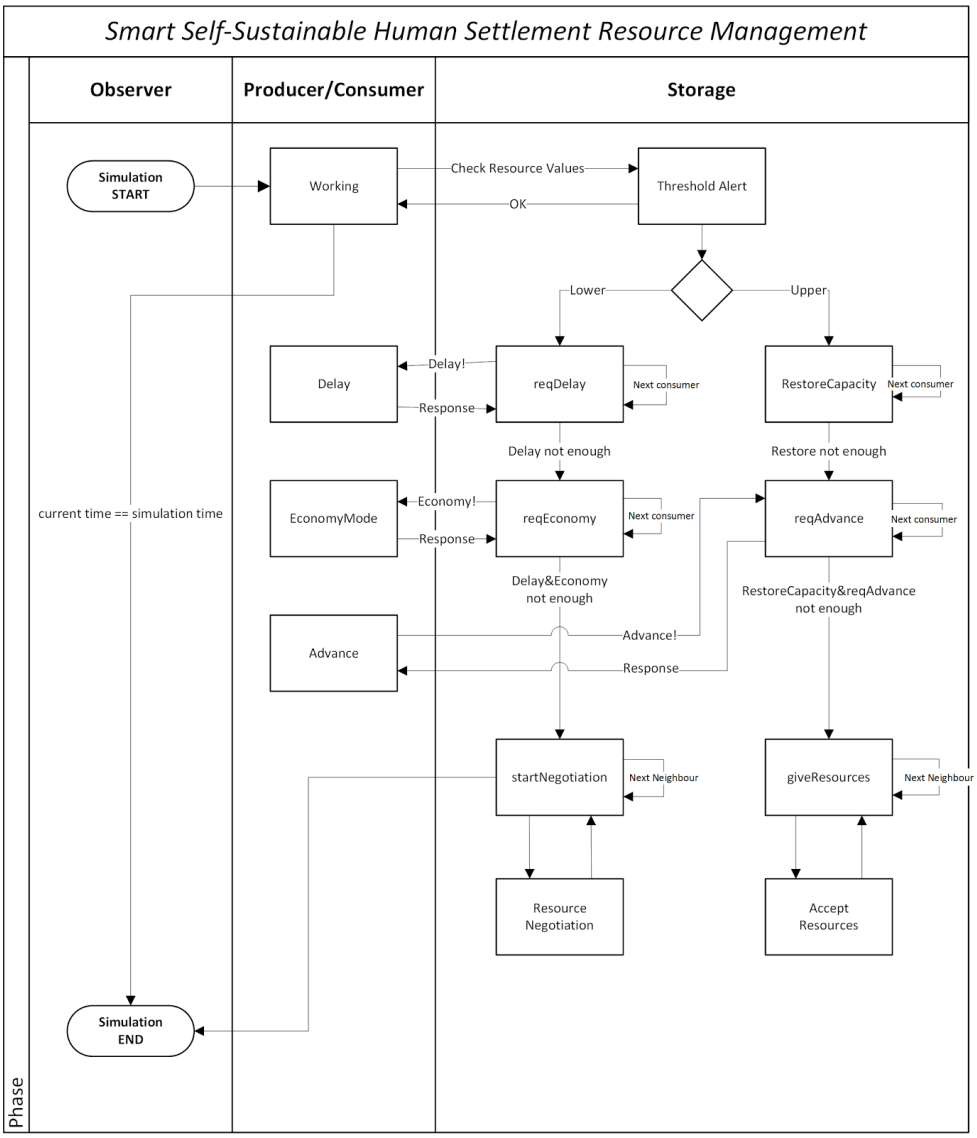


Figure 5.3: The activity diagram of the SSSHS framework, as we have shown it in [52]

The "resource negotiation" is an inter-dwellings mechanism that calculates the client's needed resource value which would bring its resource level above the lower threshold zone, and then requires a service from other agents, or servers, to initiate the resource transfer if the agreement is made between them. After the client sends its offer, the server calculates its own resource capacities by inspecting its current inner state, and decides on the counter-offer it is willing to send to the client. If the counter-offer is greater than zero and resource transfer costs are acceptable, the negotiation process is initiated. If the resource transfer costs are not acceptable, or the agreement is achieved but the counter-offer does not provide sufficient resource quantity, the client sends further requests to other agents, sequentially, until its resource levels are above the defined threshold.

The SSSHS negotiation process from the client perspective is depicted via finite state machine in Fig. 5.4.

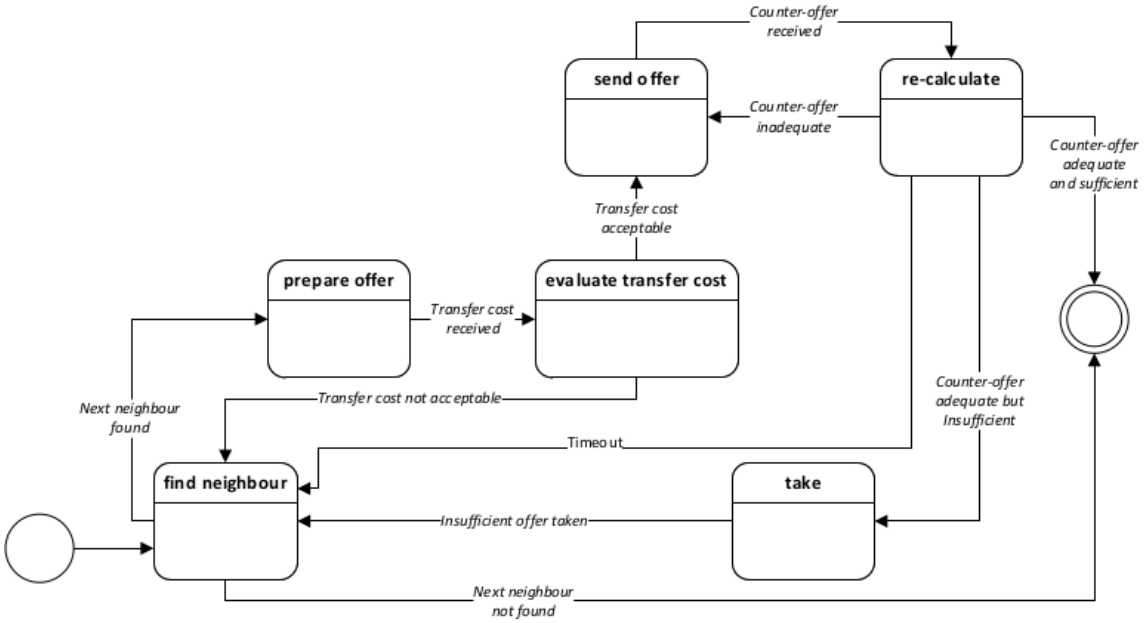


Figure 5.4: A finite state machine of the SSSHS negotiation process [51]

5.4.1 SSSHS Simulation Results Overview

The observed SSSHS simulation setup includes three dwelling units associated with three storage agents with dedicated capacities of 2000, 1440 and 1008 resource units. Resources in this example can be anything from electricity, water, fuel or any other transferable resource. Each storage agent has defined upper and lower thresholds, transfer costs, negotiation parameters, current values of resources, and other relevant data. There are four producer agents, each associated with a storage unit agent, and with a defined production distribution throughout the simulation run. Within the model, 8 consumer agents are defined, also associated with storage agents, and acting according to their own respective consumption distributions. Storage agents, consumer and producer agents are all modelled and initiated as SPADE [21] agents.

Simulation run was set to 30 time units, in this particular model observed as days. Resource units are modelled as liters of water. Consumers represent soil irrigation, residents using water in daily lives, and water usage for various utilities. Producers represent rainfall harvesting and hand collection of water from lakes, wells, streams, and other sources.

The simulation analysis shows that the system could be rendered as not self-sustainable in the early stages of the simulation, without using the SSSHS mechanisms, mainly because storage1@127.0.0.1 agent loses resources at the very beginning of the simulation.

Since the SSSHS framework has been reimplemented to run on the SPADE platform to be compatible with the modelling tool, there is a significant overhead concerning the agents' message system using the Jabber protocol; there were 1882 messages sent between agents in the simulation run. Most of the messages were exchanged between agents during the negotiation processes (a total of 10 negotiation requests were registered by the system). The negotiation parameters could be adjusted in order to decrease the number of messages, should they significantly affect the simulation performance.

There were a total of 170 system interventions registered during the simulation run, indicating the possibility of multiple self-sustainability failures within the simulation, if the SSSHS self-

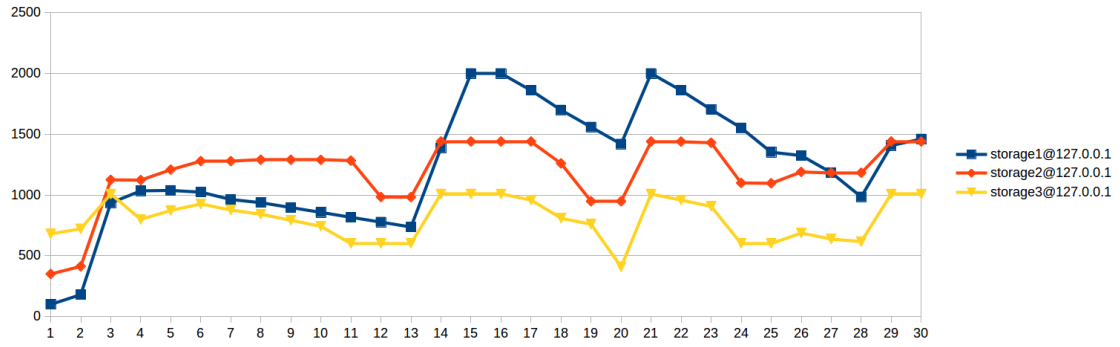


Figure 5.5: SSSHS simulation analysis: resource level values within storage agents

sustainability mechanisms were not used. There was a significantly higher number if upper threshold alerts (48) than lower threshold alerts (8), showing the inadequate storage capacities' setup of the observed system.

The inadequate storage capacities in respect to the production capacities of the observed system resulted in an irretrievable loss of a total of 9028.1 resource units, most of them lost by the storage unit storage3@127.0.0.1 (4726.6), which in the observed configuration had the smallest storage capacity (1008).

Considering further the upper threshold alerts with respect to the SSSHS self-sustainability mechanisms in the simulation run, there were 6 restore requests, 0 advance request, and 96 give requests initiated. Considering the lower threshold alerts, there was a total of 30 delay requests, 28 economy requests, and, as argued previously, 10 negotiation requests initiated in the observed simulation run. Figure 5.5 illustrates the simulation run via the most significant parameter - the resource levels within the observed storage units, governed by both the production and consumption dynamics of the observed system, and also by the self-sustainability mechanisms introduced within the SSSHS framework.

5.5 Modeling Smart Cities as LSMAS

The described SSSHS metamodel sees a self-sustainable system as a collection of organisational units comprising an organisation of individual agents enacting various roles including consumer, producer and storage. These work in unison and try to maintain self-sustainability through balancing loads of produced, stored and consumed resources. Individuals within an SSSHS dwelling unit are agents that cooperatively work not to fail the main objective of the organisation (the dwelling unit) - maintaining self-sustainability.

So far, the SSSHS framework has been used to model small-scale settlements and might be cumbersome to be used in large-scale settings directly due to a number of reasons. Firstly, the framework assumes that the various agents in a dwelling unit as well as the dwelling units mutually are connected with resource exchange and communication pipelines in form of a full graph. While this might be possible in small settlements, it is nearly impossible for the case of cities. Secondly, the used negotiation and sustainability mechanisms prescribe communication of an individual unit with all other agents in order to share resources when needed, which is impractical. To solve the first issue, we will introduce organizational units in a recursive hierarchical structure. Producer, consumer and storage agents are grouped in organizational units on a number of levels: apartment, house, building, building block, neighbourhood, part of city, city etc. Only organizational units on the same level are connected by a full resource and communication graph which reduces the number of needed connections dramatically. Also, to solve the second issue, organizational units in need of sharing resources query and negotiate only with other units on the same level, thus

reducing the number of exchanges messages dramatically as well. To connect the various levels, we introduce a dynamic leader role: each organizational unit has a leader-negotiator which, in case a resource sharing issue cannot be solved on the current level, is entitled to negotiate and query other leaders on the next level. This leader-negotiator unit also has to manage self-sustainability mechanisms as defined in the SSSHS framework on its and its upper level to provide information (like manipulation of operating times for example) from one level to another.

Figure 5.6 shows such an extended SSSHS system using the LSMAS organisational metamodel described earlier. The basic unit of the SSSHS, the dwelling unit, is modeled using an organisational unit (stickman on Fig. 5.6) on its individual level. Aggregation of such individual agents is modeled using organisational units named Building, Neighbourhood and Settlement, in a rising level of aggregation, i.e. individual SSSHS agents representing dwelling units (for example flats) are aggregated (joined or grouped) into building units which are in turn aggregated into neighbourhoods.

The described approach to modelling stems from the joined ideas of SSSHS and the organisational metamodel. SSSHS observes units as special constructs built from several other lower-level units. Coupled with the organisational metamodel, it is possible to upgrade the SSSHS idea so that various levels of aggregated units act just as the individual units do. This idea is described in more detail in [19]. Briefly, and applied to SSSHS, it can be described as follows. A dwelling unit is an organisational unit. If a set of organisational units is coupled with a set of roles for those units and specific criteria of organising (for example a particular objective of maintaining self-sustainability or a particular mission and similar), then that set is an organisational unit. This observation makes it possible to apply SSSHS ideas to organisational units on various levels of aggregation. In other words, it makes it possible to model a system comprising aggregated dwelling units, i.e. building units, and work with them as if they were basic dwelling units, thus making them able to play SSSHS roles and strive to continually reach the main goal of maintaining self-sustainability. Furthermore, it is possible to raise the level of aggregation, and observe neighbourhood units (aggregated building units) as organisational units, and let them play SSSHS roles just as the organisational units on lower levels of aggregation did. A big challenge arises here though, which is concerned with communication between units belonging to different aggregation units.

Basic SSSHS units are modeled as standalone units, meaning that, as mentioned in section 5.4, they strive to produce, use and store enough resources for themselves in order to be free from the wider environment and to be able to survive off the grid. The challenge present in levels of aggregation arises from the need for a mean of communication between various aggregated units. For example, it is defined by the SSSHS framework that storage units take care of the coordination and communication, but if several dwelling units are to be aggregated into a single building unit, in an environment already consisting of several building units comprising dwelling units, and this building unit strives to maintaining self-sustainability, which of the included dwelling units will be responsible for inter-unit communication with other building units? In other words, how will building units communicate with each other and exchange resources thus maintaining self-sustainability? Furthermore, should another level of aggregation be introduced, where neighbourhood units are created comprising a number of building units, how will they communicate between each other?

The answer to these questions is devised in the form of a new role that is meant to work as a negotiator of an aggregated unit. Since the storage role contains all the actions necessary for the process of interaction and communication in general, it is thought to be only fitting to create this new role as a part of the storage role. Furthermore, such a view is in compliance with the general description of the SSSHS framework insomuch as storage units continue to take care of one resource only. For example, a storage unit working with resource named Electricity will communicate with other such storage units in other aggregated units. Together, storage units of an aggregated unit will be coordinated into knowing the state of a given resource in the given aggregated unit. Normative elements of an organisation will be responsible for assigning negotiation role to specific storage

units, and ascertaining that there is only one such unit at any given point in time. For the sake of understanding, an example scenario is given in the following section.

Another important problem is the problem of transferring resources on higher levels of aggregation. For example, on a neighbourhood level, the amounts of resources to be shared will probably be much larger than on individual levels, exceeding storage capacities of individual storage units. Thus, another line of hierarchy has to be introduced to aggregate individual lower-level storage units into virtual larger storage units. This also implies the need to extend the current resource sharing protocols of the SSSHS framework with an additional mode of operation in which a storage negotiator unit can instruct lower level storage units to transfer or receive resources from a higher level. In this way, greater levels of a given resource can be shared between higher level units when needed (in peak consumption of production times for example).

Additional clarification is needed on Fig. 5.6 showing SSSHS example modelled using the organisational metamodel described earlier in this chapter. As mentioned earlier in this section, an individual agent can be aggregated into units of higher aggregation level. Each individual unit can play one of the SSSHS roles (blue hats in Fig. 5.6). Playing a role provides the given individual with actions necessary for undertaking the process (green squares in Fig. 5.6) associated with the given role. Processes are usually connected to objectives (circular in Fig. 5.6) they can help fulfil. Following the stated, one can see in Fig. 5.6 that for example an individual unit (stick figure in Fig. 5.6) can play three different roles one of which, the Storage role, through its Negotiator sub-role, provide the given agent with actions needed to undertake process named Negotiate that contains actions necessary for the resource negotiation process. It is worth noting here that some of the stages of aggregation can be changed depending on the specific situation, i.e. some of them can be further detailed or skipped entirely (for example, it is not necessary to consider Building aggregation level if DwellingUnits are houses, or it is not necessary to take into account Neighbourhood aggregation level if the settlement in question comprises only a couple of DwellingUnits).

5.6 Example Scenario

To clarify the model described in the previous section, consider the following (simplified) example scenario: a (smart self-sustainable off-grid) city consisting of 3 buildings, each building having 5 flats, each flat having (smart) photovoltaic system, a (smart) battery unit, and a (smart) refrigerator enacting the roles of producer, storage, and consumer, respectively (see Fig. 5.7 for an illustration). Let us assume that this city is organized into three organizational levels: (1) flat level (each individual flat is an organizational unit with the battery unit as its negotiator), (2) building level (each building's flats are an organizational unit with the flat on the ground floor as its negotiator) and (3) city level (consisting of all three buildings, with the first building being the negotiator). Note that, on the city level the actual negotiator is the battery unit from the ground floor in the first building which is enacting a role on two higher levels.

Due to geographical location, the first building has the best orientation to the sun, while the other two buildings' PV panels end up in the shadow of the first building earlier during the day. In this way the battery units of the first building are charged longer thus coming to a point where they cannot store any additional electricity. Let us assume that the battery unit on the 4th floor of the first building is the first to reach full charge and initiates negotiation to share its surplus energy to battery units on the same level - the units in the first building. For a short period of time these units can receive additional electricity from the PV panel above them, but soon, one after another, become fully charged. At that point the negotiator on the ground floor initiates negotiation on a higher level (building level) and finds that building two has enough space to take care of surplus electricity coming from the first building's PV panels. The electricity from these panels is firstly redirected to the ground floor of the first building (negotiator), then to the ground floor of the second

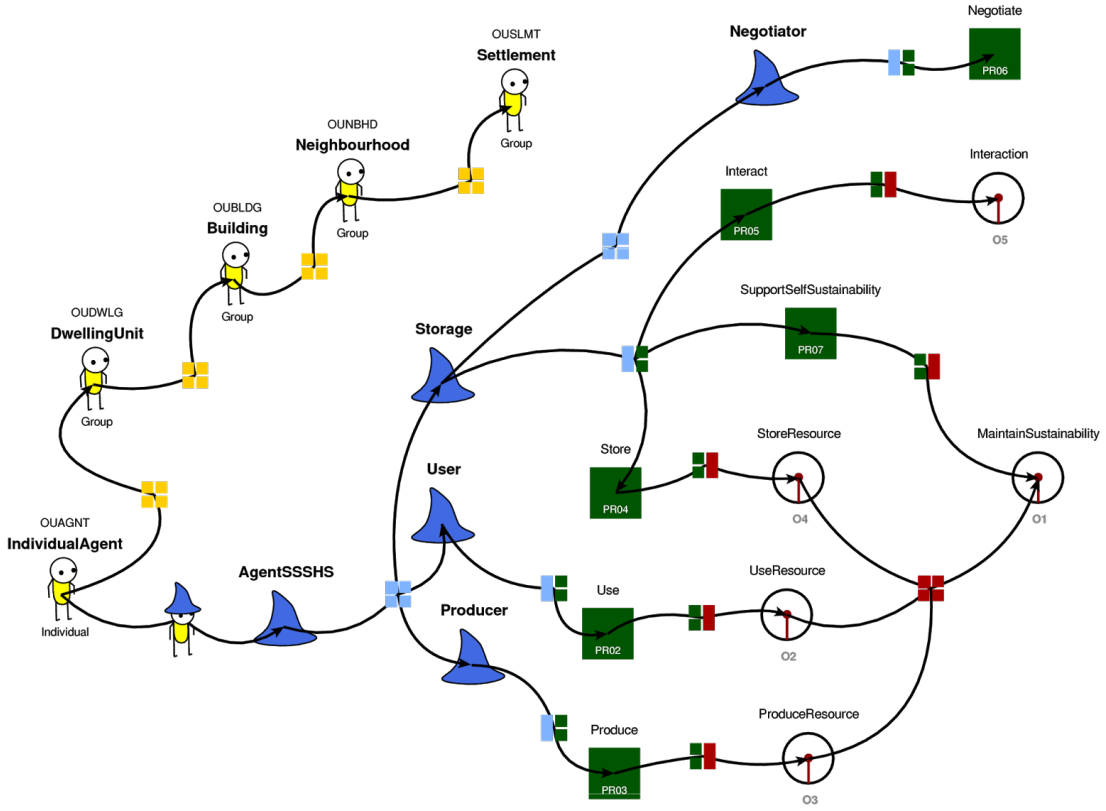


Figure 5.6: SSSHS elements modelled using the LSMAS organisational metamodel

building (second negotiator) and finally distributed by the negotiator of the second building among the battery units in the flats above.

Let us further assume that due to low production, charge runs low in all battery units in building three. For example, the refrigerator in flat on floor 2 of the third building schedules that it will need additional energy in the following period which exceeds the current level of its respective battery unity. The battery unit (negotiator) firstly initiates negotiation with the battery units on the same level (flat level), and gets informed that none of the battery units in building three has enough spare electricity to share. It informs the negotiator on this level (the battery unit on the ground floor) which initiates negotiation on a higher level (building level) and finds that the first building has surplus electricity. The negotiator from the first building instructs the battery unit which has surplus energy in its building to redirect energy to the ground floor and then redirects it to the third building's ground floor. The negotiator from the third building then redirects it to the second floor from where the negotiation was initiated.

5.7 Discussion

As one can see from the presented organizational model as well as the example, the proposed extension of the SSSHS framework allows for flexible modelling of large-scale self-sustainable smart cities using a higher level of abstraction: namely organizational structure which imposes normative roles on the agents employed in such a system. Nevertheless, such an architecture also imposes new questions that have to be dealt with. For example, a question not directly addressed herein is the question of how to physically materialize such an organization. While from a computer science perspective the implementation is merely a (possibly wireless) computer network, from an engineering perspective the proposed system becomes harder to realize and imposes additional

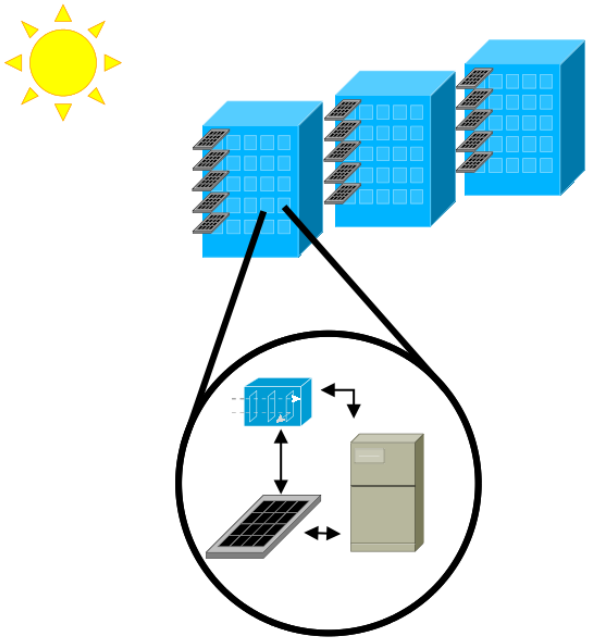


Figure 5.7: Illustration of the example scenario

constraints on the function of the system since physical movement of various resources has to be achieved. This implies additional infrastructure and additional spatio-temporal constraints.

For example, in the proposed model, it might happen that some appliance in one part of the city requires water as a resource which is available only on a quite distant other part of the city. How should this amount of water be transported from one location to the other? While such a transfer might be feasible for smaller distances, or even for greater distances for certain types of resources like electricity, for resources like water, heat or oil (especially in smaller amounts) this isn't the case. Also, even if such a transport is feasible (and cost effective), the question of how much time is needed to transport such a resource can impose additional constraints onto the system.

From this discussion, it becomes obvious that additional constraints have to be put in place in regard to negotiation, including careful design of organizational levels. A possible solution is to define feasibility zones for each agent and each type of resource. For example, one might define that a feasibility zone for the resource type water reaches only to all agents which are in a certain temporal reach (e.g. the certain amount can be transported within a given time limit). Other possibilities might include distance/proximity limits, channel type limits, or level limits (e.g. negotiate only on n levels from the current one).

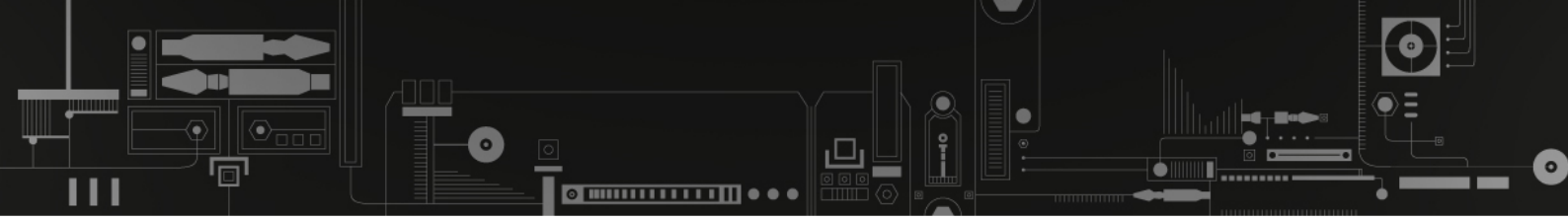
Further constraint of the current model is that modelling of other types of interactions, beyond resource management, hasn't been included into it explicitly. While this hierarchical structure might function in some types of interactions, for other types of interactions which require greater dynamics like adaptive smart flats with ambient intelligence [37], it might be too rigid. Nevertheless, such interactions can be modeled using different types of organizational forms (like the learning organization for example), which can function in parallel to the current defined model, since agents only get additional roles to enact.

5.8 Conclusion

In this chapter, we have extended the SSSHS framework with additional organizational concepts (namely organizational structure and normative roles) to allow modelling of self-sustainable smart cities with special regard to resource management using our LSMAS organizational metamodel that is still under development. We have described the current state of the LSMAS metamodel and showed its practical applicability in modeling complex multi-agent organizations.

By introducing a hierarchical structure into the SSSHS framework and defining the role of negotiator we have shown that it becomes feasible even for large-scale systems like smart cities. Additionally, we have provided an example scenario which aims on better depicting the various interactions that might take place in such a system. In the end, we have discussed the implications of the proposed model and identified a number of spatio-temporal constraints, namely feasibility zones, that have to be imposed on the model to become feasible for certain types of resources.

Our future research will focus on further enrichment of the provided models through the introduction of the proposed feasibility zones, as well as large-scale simulations of various scenarios to identify possible additional bottlenecks and problems with the model. Another line of research might be the introduction of learning techniques for smart devices in residential buildings as outlined in [37], through possible inclusion of additional organizational design techniques.



hrzz
Hrvatska zaklada za znanost

foi
SVEUČILIŠTE U ZAGREBU
FAKULTET
ORGANIZACIJE I
INFORMATIKE
VARAŽDIN

lab

Bibliography

[1] Hosny Ahmed Abbas, Samir Ibrahim Shaheen, and Mohammed Hussein Amin. "Organization of multi-agent systems: an overview". In: *International Journal of Intelligent Information Systems* 4.3 (2015), pages 46–57 (cited on pages 22, 41, 42).

[2] Amjad Anvari-Moghaddam et al. "Optimal real-time dispatch for integrated energy systems: An ontology-based multi-agent approach". In: *Power Electronics for Distributed Generation Systems (PEDG), 2016 IEEE 7th International Symposium on*. IEEE. 2016, pages 1–7 (cited on page 40).

[3] Luigi Atzori, Antonio Iera, and Giacomo Morabito. "The internet of things: A survey". In: *Computer Networks* 54.15 (2010), pages 2787–2805 (cited on page 39).

[4] Inmaculada Ayala, Mercedes Amor, and Lidia Fuentes. "An agent platform for self-configuring agents in the internet of things". In: *Third International Workshop on Infrastructures and Tools for Multiagent Systems, ITMAS*. 2012, pages 65–78 (cited on page 40).

[5] Alain Barrat, Marc Bathelemy, and Alessandro Vespignani. *Dynamical Processes on Complex Networks*. 1st. New York, New York, USA: Cambridge University Press, 2008, page 367. ISBN: 978-0-511-45558-2. URL: `www.cambridge.org/9780521879507` (cited on page 7).

[6] Chang-Sik Cho et al. "Online game testing using scenario-based control of massive virtual users". In: *Advanced Communication Technology (ICACT), 2010 The 12th International Conference on*. Volume 2. IEEE. 2010, pages 1676–1680 (cited on page 20).

[7] Chang-Sik Cho et al. "Scenario-based approach for blackbox load testing of online game servers". In: *Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC), 2010 International Conference on*. IEEE. 2010, pages 259–265 (cited on page 20).

[8] Daniel D Corkill and Susan E Lander. "Diversity in agent organizations". In: *Object Magazine* 8.4 (1998), pages 41–47 (cited on page 41).

[9] Luciano Coutinho, J Sichman, and Olivier Boissier. "Modelling dimensions for agent organizations". In: *Handbook of research on multi-agent systems: Semantics and dynamics of organizational models* 2 (2009), pages 18–50 (cited on pages 40, 42).

[10] Juan De Lara and Hans Vangheluwe. "AToM3: A Tool for Multi-formalism and Metamodelling". In: *International Conference on Fundamental Approaches to Software Engineering*. Springer. 2002, pages 174–188 (cited on page 19).

[11] Tom De Wolf and Tom Holvoet. "Emergence and self-organisation: a statement of similarities and differences". In: *Proceedings of the International Workshop on Engineering Self-Organising Applications 2004*. 2004, pages 96–110 (cited on page 41).

[12] Virginia Dignum. *Handbook of Research on Multi-Agent Systems: Semantics and Dynamics of Organizational Models: Semantics and Dynamics of Organizational Models*. IGI Global, 2009 (cited on page 41).

[13] Bogdan Okreša Đurić. "Organizational metamodel for large-scale multi-agent systems". In: *Trends in Practical Applications of Scalable Multi-Agent Systems, the PAAMS Collection*. Springer, 2016, pages 387–390 (cited on pages 5, 9, 20–22, 40, 42).

[14] Bogdan Okreša Đurić. "A novel approach to modelling distributed systems: Using large-scale multi-agent systems". In: *Software Project Management for Distributed Computing*. Springer, 2017, pages 229–254 (cited on pages 5, 40, 42).

[15] Bogdan Okreša Đurić and Markus Schatten. "Defining ontology combining concepts of massive multi-player online role playing games and organization of large-scale multi-agent systems". In: *Information and Communication Technology, Electronics and Microelectronics (MIPRO), 2016 39th International Convention on*. IEEE. 2016, pages 1330–1335 (cited on page 21).

[16] Richard E Fikes and Nils J Nilsson. "STRIPS: A new approach to the application of theorem proving to problem solving". In: *Artificial intelligence* 2.3 (1972), pages 189–208 (cited on page 24).

[17] Magda Fontana and Pietro Terna. "From Agent-based models to network analysis (and return): the policy-making perspective". In: *SwarmFest*. University of Notre Dame. 2014 (cited on pages 5, 8, 9, 18).

[18] José M Gascueña, Elena Navarro, and Antonio Fernández-Caballero. "Model-driven engineering techniques for the development of multi-agent systems". In: *Engineering Applications of Artificial Intelligence* 25.1 (2012), pages 159–173 (cited on page 21).

[19] Global Collect. *THE GLOBAL MMO GAMES MARKET: Payments, Intelligence and Trends*. 2014. URL: `http://www.globalcollect.com/the-global-mmo-games-market` (cited on page 2).

[20] Philippe Golle and Nicolas Ducheneaut. "Preventing bots from playing online games". In: *Computers in Entertainment (CIE)* 3.3 (2005), pages 3–3 (cited on page 20).

[21] Miguel Escrivá Gregori, Javier Palanca Cámara, and Gustavo Aranda Bada. "A jabber-based multi-agent system platform". In: *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*. ACM. 2006, pages 1282–1284 (cited on pages 9, 19, 24, 47).

[22] Robert E Hall et al. *The vision of a smart city*. Technical report. Brookhaven National Lab., Upton, NY (US), 2000 (cited on page 39).

[23] LA Hurtado, PH Nguyen, and WL Kling. "Smart grid and smart building inter-operation using agent-based particle swarm optimization". In: *Sustainable Energy, Grids and Networks* 2 (2015), pages 32–40 (cited on page 40).

[24] YungWoo Jung et al. "VENUS: The online game simulator using massively virtual clients". In: *Asian Simulation Conference*. Springer. 2004, pages 589–596 (cited on page 20).

[25] Stamatis Karnouskos and Thiago Nass De Holanda. “Simulation of a smart grid city with software agents”. In: *Computer Modeling and Simulation, 2009. EMS’09. Third UKSim European Symposium on*. IEEE. 2009, pages 424–429 (cited on page 41).

[26] Marlieke van Kesteren, Jurriaan Langevoort, and Franc Grootjen. “A step in the right direction: Botdetection in MMORPGs using movement analysis”. In: *Proc. of the 21st Belgian-Dutch Conference on Artificial Intelligence (BNAIC 2009)*. 2009, pages 129–136 (cited on page 20).

[27] Yifan Li, Petr Musilek, and Loren Wyard-Scott. “Fuzzy logic in agent-based game design”. In: *Fuzzy Information, 2004. Processing NAFIPS’04. IEEE Annual Meeting of the*. Volume 2. IEEE. 2004, pages 734–739 (cited on page 21).

[28] Marko Maliković and Markus Schatten. “Artificial Intelligent Player’s Planning in Massively Multi-Player On-Line Role-Playing Games”. In: *26th Central European Conference on Information and Intelligent Systems*. 2015 (cited on page 24).

[29] Mihail Mihaylov et al. “Smart grid demonstration platform for renewable energy exchange”. In: *Advances in Practical Applications of Scalable Multi-agent Systems. The PAAMS Collection*. Springer, 2016, pages 277–280 (cited on page 41).

[30] NewZoo. *The Global MMO Market; Sizing and Seizing Opportunities*. 2012. URL: http://www.newzoo.com/infographics/the-global-mmo-market-sizing-and-seizing-opportunities/ (cited on page 2).

[31] Juan Pavón, Jorge Gómez-Sanz, and Rubén Fuentes. “Model driven development of multi-agent systems”. In: *European Conference on Model Driven Architecture-Foundations and Applications*. Springer. 2006, pages 284–298 (cited on page 21).

[32] Emanuel Montero Reyno and José Á Carsí Cubel. “Model Driven Game Development: 2D Platform Game Prototyping.” In: *GAMEON*. Citeseer. 2008, pages 5–7 (cited on page 21).

[33] Mariacristina Roscia, Michela Longo, and George Cristian Lazaroiu. “Smart City by multi-agent systems”. In: *Renewable Energy Research and Applications (ICRERA), 2013 International Conference on*. IEEE. 2013, pages 371–376 (cited on page 40).

[34] Stuart J Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Edited by Stuart Russell and Peter Norvig. 3rd. Prentice Hall series in artificial intelligence. New Jersey, USA: Pearson Education, Inc., 2010, page 1151. ISBN: 9780136042594. DOI: 10.1017/S0269888900007724. URL: http://scholar.google.com/scholar?hl=en%7B%5C&%7DbtnG=Search%7B%5C&%7Dq=intitle:No+Title%7B%5C#%7D0%20http://www.cpp.edu/%7B~%7Dkding/materials/Artificial%20Intelligence%20A%20Modern%20Approach%203rd.pdf%20https://books.google.hr/books?id=8jZBksh-bUMC (cited on page 8).

[35] M. Schatten, J. Ševa, and I. Tomičić. “A Roadmap for Scalable Agent Organizations in the Internet of Everything”. In: *Journal of Systems and Software* 115 (2016), pages 31–41 (cited on pages 20, 21).

[36] Markus Schatten. “Reorganization in Multi-Agent Architectures: An Active Graph Grammar Approach”. In: *Business Systems Research* 34.1 (2013), pages 14–20 (cited on page 40).

[37] Markus Schatten. “Organizational Architectures for Large-Scale Multi-Agent Systems’ Development: An Initial Ontology”. In: *Distributed Computing and Artificial Intelligence, 11th International Conference*. Springer. 2014, pages 261–268 (cited on pages 40, 42, 52, 53).

[38] Markus Schatten. "Organizational Architectures for Large-Scale Multi-Agent Systems' Development: An Initial Ontology". In: *Advances in Intelligent Systems and Computing* 290 (2014), pages 261–268 (cited on pages 21, 22).

[39] Markus Schatten. "Smart Residential Buildings as Learning Agent Organizations in the Internet of Things". In: *Business Systems Research* 5.1 (2014), pages 34–46 (cited on page 40).

[40] Markus Schatten and Bogdan Okreša Đurić. "Social Networks in "The Mana World"-an Analysis of Social Ties in an Open Source MMORPG". In: *International Journal of Multimedia and Ubiquitous Engineering* 11.3 (2016), pages 257–272 (cited on page 20).

[41] Markus Schatten, Igor Tomičić, and Bogdan Okreša Đurić. "A Review on Application Domains of Large-Scale Multiagent Systems". In: *Central European Conference on Information and Intelligent Systems*. 2017 (cited on page 5).

[42] Markus Schatten, Igor Tomičić, and Bogdan Okreša Đurić. "Multi-agent Modeling Methods for Massivley Multi-Player On-Line Role-Playing Games". In: *MIPRO*. Opatija, HR, 2015 (cited on page 20).

[43] Markus Schatten et al. "Towards a Formal Conceptualization of Organizational Design Techniques for Large Scale Multi Agent Systems". In: *Procedia Technology* 15 (2014), pages 577–586 (cited on page 42).

[44] Markus Schatten et al. "Agents as Bots–An Initial Attempt Towards Model-Driven MMORPG Gameplay". In: *International Conference on Practical Applications of Agents and Multi-Agent Systems*. Springer. 2017, pages 246–258 (cited on pages 2, 5).

[45] Markus Schatten et al. "Automated MMORPG Testing–An Agent-Based Approach". In: *International Conference on Practical Applications of Agents and Multi-Agent Systems*. Springer. 2017, pages 359–363 (cited on pages 2, 5).

[46] Thanos G Stavropoulos et al. "A multi-agent coordination framework for smart building energy management". In: *Database and Expert Systems Applications (DEXA), 2014 25th International Workshop on*. IEEE. 2014, pages 126–130 (cited on page 40).

[47] Stephen Tang and Martin Hanneghan. "State-of-the-art model driven game development: A survey of technological solutions for game-based learning". In: *Journal of Interactive Learning Research* 22.4 (2011), page 551 (cited on page 21).

[48] *TmwAthena Packets*. URL: `https://www.themanaworld.org/index.php/Archive:TmwAthena_Packets` (cited on page 26).

[49] Igor Tomičić. "Agent-Based Framework for Modelling and Simulation of Resource Management in Smart Self-Sustainable Human Settlements". PhD thesis. Faculty of Organization and Informatics, 2016 (cited on pages 5, 43).

[50] Igor Tomičić and Markus Schatten. "Towards an Agent Based Framework for Modelling Smart Self-Sustainable Systems". In: *Interdisciplinary Description of Complex Systems* 13.1 (2015), pages 57–70 (cited on pages 40, 43).

[51] Igor Tomičić and Markus Schatten. "A Case Study on Renewable Energy Management in an Eco-Village Community in Croatia–An Agent Based Approach". In: *International Journal of Renewable Energy Research* 6 (2016), pages 1307–1317 (cited on pages 6, 43–45, 47).

[52] Igor Tomičić and Markus Schatten. "Agent-based framework for modeling and simulation of resources in self-sustainable human settlements: a case study on water management in an eco-village community in Croatia". In: *International Journal of Sustainable Development & World Ecology* (2016), pages 1–10 (cited on pages 6, 39, 40, 43, 46).

[53] Panagiotis Vlacheas et al. “Enabling smart cities through a cognitive management framework for the internet of things”. In: *IEEE communications magazine* 51.6 (2013), pages 102–111 (cited on page 40).

[54] Danny Weyns, Robrecht Haesevoets, and Alexander Helleboogh. “The MACODO organization model for context-driven dynamic agent organizations”. In: *ACM Transactions on Autonomous and Adaptive Systems (TAAS)* 5.4 (2010), page 16 (cited on page 41).

[55] Wikipedia Contributors. *Massively multiplayer online game — Wikipedia, The Free Encyclopedia*. [Online; accessed 9-February-2015]. 2015. URL: http://en.wikipedia.org/w/index.php?title=Massively_multiplayer_online_game&oldid=646153954 (cited on page 2).

[56] Wikipedia Contributors. *Massively multiplayer online role-playing game — Wikipedia, The Free Encyclopedia*. [Online; accessed 9-February-2015]. 2015. URL: http://en.wikipedia.org/w/index.php?title=Massively_multiplayer_online_role-playing_game&oldid=642752380 (cited on page 2).

[57] Wikipedia Contributors. *Role-playing video game — Wikipedia, The Free Encyclopedia*. [Online; accessed 9-February-2015]. 2015. URL: http://en.wikipedia.org/w/index.php?title=Role-playing_video_game&oldid=642919862 (cited on page 2).

[58] Jeff Yan. “Bot, cyborg and automated turing test”. In: *International Workshop on Security Protocols*. Springer. 2006, pages 190–197 (cited on page 20).

[59] K Yang and Sung-Bae Cho. “Towards sustainable smart homes by a hierarchical hybrid architecture of an intelligent agent”. In: *Sustainability* 8.10 (2016), page 1020 (cited on page 40).