UTRECHT UNIVERSITY

MASTER'S THESIS

Agent-based modeling as a tool to support decision making rules used by smart contracts in DLT based communities

Author:
Tom PEIRS
6362915

Supervisor:
Dr. Slinger JANSEN
Dr. Frank DIGNUM

A thesis submitted in fulfillment of the requirements for the degree of Master of Science

in the

Business Informatics
Department Information and Computing Sciences

May 31, 2021

UTRECHT UNIVERSITY

Abstract

Faculty of Science
Department Information and Computing Sciences

Master of Science

Agent-based modeling as a tool to support decision making rules used by smart contracts in DLT based communities

by Tom PEIRS
6362915

Various issues are surrounding the relatively new area of smart contract design. These problems are very diverse in nature ranging from performance issues to exploiting of contracts. The concerns stem from the transformation of domain information to smart contract immutability to developers coding insecure smart contracts. Hence, there is a compelling need to study smart contracts to find and address vulnerabilities before deployment on a ledger due to its immutability. This study aims to support the design of a code search community platform through agent-based modeling by simulating systems requirements. In this paper we present an abstract replication of institutional emergence patterns. We used the ADICO grammar of institutions as the basic structure to model system requirements. We establish a common pool resource of institutional statements through a standardized method, which is then used to simulate smart contracts through agent-based modeling. We demonstrated through a case study the usage of the ADICO framework. We observe common institutional patterns which are used to study smart contract design in an agent-based environment. Institutions made a major contribution to the governance of common-pool resource systems in a simplified environment. In addition, we show how different domain concepts regarding agent-based modeling and smart contract design can be mapped. Furthermore, this study generates open-source software to simulate a decentralized system as an agent-based model through Repast Simphony, from which insights regarding sustainability can be gathered.
We conclude that agent-based modeling is a foundational tool for designing smart contracts in new DLT based communities.
Keywords: Agent-Based Modeling, Smart Contracts, Complex Systems ,Institutional Grammar, Model-Driven development, Blockchain, Solidity, Repast

Contents

Abstract **iii**

1 Introduction **1**
1.1 Smart Contracts . . . 2
1.2 Problem Statement . . . 2
1.3 Research Focus . . . 3
1.3.1 Agent-Based Modeling . . . 3
1.4 Contributions . . . 5
1.5 Thesis Outline . . . 6

2 Research Approach **7**
2.1 Research Questions . . . 7
2.2 Research Paradigm . . . 9
2.2.1 Research Methods . . . 10
2.3 Literature Review Protocol . . . 10
2.3.1 Phase 1: Multivocal Literature Review Planning . . . 11
2.3.2 Phase 2: Search Process and Source Selection . . . 11
Selecting source engines and search keywords . . . 11
Application of inclusion/exclusion criteria . . . 12
Final Pool of sources . . . 12
2.3.3 Phase 3: Data Classification . . . 13
2.3.4 Phase 4: Synthesis and Review . . . 13
2.3.5 Summary . . . 13
2.4 Method Engineering . . . 13
2.4.1 Summary . . . 14
2.5 Case Study . . . 14
2.5.1 Case Study Requirements . . . 14

3 Literature Study **15**
3.1 Agent-Based Modeling . . . 15
3.1.1 Introduction . . . 15
3.1.2 Agent-Based Modeling Requirements . . . 16
To Summarize: Agent-Based Modeling Requirements . . . 17
3.1.3 Autonomous Agents . . . 17
To Summarize: Autonomous Agents . . . 19
3.1.4 Rules, Behavior, and Relationships . . . 19
To Summarize: Rules, behavior, and Relationships . . . 20
3.1.5 Agent Environments . . . 20
To Summarize Agent Environments . . . 21
3.1.6 Methods for agent-based Modeling . . . 21
(a) Agent model design . . . 21
(b) Agent model implementation . . . 22
(c) Agent model services . . . 22

3.1.7 Summary ABM . . . 23
3.2 Smart Contracts . . . 23
3.2.1 Consensus Layer . . . 24
To Summarize: Consensus . . . 26
3.2.2 Peer-to-Peer Network Layer . . . 26
To Summarize: P2P . . . 27
3.2.3 Ethereum Virtual Machine . . . 27
To Summarize: EVM . . . 28
3.2.4 Smart Contract Design with Solidity . . . 28
3.2.5 Summary Smart Contracts . . . 30
3.3 Modeling Smart Contracts with Institutional Grammar . . . 30
3.3.1 Bridging the gap between smart contracts and agent-based modeling . . . 32

4 ABM Decision Support 37
4.1 ABM Tool Characteristics . . . 37
4.1.1 Scalability . . . 38
4.1.2 Development Effort . . . 38
4.2 ABM Tool Selection . . . 39

5 SearchSECO Case Study 41
5.1 Introduction to SearchSECO . . . 41
5.2 Behaviors of the Agents . . . 42
5.3 Scenarios . . . 42
5.4 Problem and Object Formulation . . . 43
5.5 ADICO statements . . . 45

6 Simulation 49
6.1 Purpose . . . 49
6.2 Entities, state variables, and scales . . . 49
6.2.1 Agents . . . 50
6.3 Process Overview and Scheduling . . . 51
6.3.1 Scheduled Method . . . 51
6.4 Design Concepts . . . 54
6.4.1 Interface . . . 55
6.4.2 Styles . . . 55
6.4.3 Data Sets & Charts . . . 56
6.5 Input Data . . . 56
6.6 Observations . . . 56

7 Outcome 57
7.1 Scenario 1: Towards an Equilibrium State . . . 57
7.1.1 Iteration 1: Defining a neutral starting point with one agent . . . 57
Run Conditions . . . 58
Expectations . . . 58
Outcome . . . 58
Interpretation . . . 58
7.1.2 Iteration 2: Increasing Download Ratio . . . 59
Run Conditions . . . 59
Expectations . . . 60
Outcome . . . 60

Interpretation 60
7.1.3 Iteration 3: Increasing Upload Ratio 60
Run Conditions 60
Expectations 61
Outcome 61
Interpretation 61
7.1.4 Iteration 4: Increasing number of agents 62
Run Conditions 62
Expectations 62
Outcome 62
Interpretation 63
7.2 Scenario 2: Altering behaviors - adding complexity 63
7.2.1 Iteration 5: Adding new agent type empirical software engineer 64
Run Conditions 64
Expectations 64
Outcome 64
Interpretation 65
7.2.2 Iteration 6: Adding different behavior to agent types 66
Run Conditions 66
Expectations 66
Outcome 67
Interpretation 67
7.3 Scenario 3: Financial factors as incentives 68
7.3.1 Iteration 7: Financial factors as incentives 68
Run Conditions 68
Expectations 69
Outcome 69
Interpretation 70
7.3.2 Summary of Iteration 1 - 7 70
7.4 Observations 71

8 Findings and Discussion **73**
8.1 Major Findings 73
8.2 Contributions 75
8.3 Limitations 77
8.4 Future Work 78

9 Conclusion **81**

A Structure of a smart contract **85**

B Process Deliverable Diagrams **89**

C ABM Tools **91**

Chapter 1

Introduction

There have been contracts for as long as human beings have done business with one another. One can denote a contract as a binding agreement between two or more parties in its purest form. Over time, these contracts have evolved in size and complexity. Today, contracts form the backbone of modern business and trade across the globe. Lawyers and consultants are required as contracts become more complicated, longer, and detailed to frame or defend them.

In response to complicated contracts, a dramatic increase in the popularity of smart contracts has been seen in recent years (Clack, Bakshi, and Braine, 2016). Smart contracts are small programs written in blocks running on top of a blockchain. Smart contracts can autonomously receive and execute transactions without the need of trusted third parties (Grishchenko, Maffei, and Schneidewind, 2018). The most popular framework where smart contracts are deployed is Ethereum (Buterin et al., 2014).

Removing trusted third parties such as lawyers or consultants becomes paramount to delivering trustworthy smart contracts free of malicious intent. Like all software, smart contracts may contain vulnerabilities or performance issues as the rules and statements are not designed to fit the optimal solution. What makes the validation of smart contracts even more crucial is the fact that transactions are immutable. Immutability means that once the smart contract is deployed on a distributed ledger technology (DLT) such as Ethereum, the smart contract's state can not be reversed.

Unfortunately, malicious attackers or financial benefactors have proven to exploit smart contracts by abusing vulnerabilities in these contracts. Careful design and execution do not protect against poorly written or unsafe contracts. An aspect is recently seen in the massive theft of funds from the most popular Decentralized Autonomous Institution of Ethereum (DAO) (Finley, 2016). As a consequence, many vulnerabilities in smart contracts were maliciously exploited. Several statistical techniques and methods have been developed to find vulnerabilities in smart contracts (Feist, Grieco, and Groce, 2019; Luu et al., 2016; Tikhomirov et al., 2018; Tsankov et al., 2018). Many statistical analysis methods have been tested either on custom data sets or data sets with a small number of exceptions (Durieux et al., 2020; Parizi et al., 2018). Despite the prevalence of these statistic research instruments, vulnerabilities abound (Perez and Livshits, 2019). This brings into question the effectiveness and related methods of these statistical techniques. ***It is critical to find and address vulnerabilities that are thoroughly tested and evaluated before deployment on a ledger which indicates a compelling need to study smart contracts (Daian, 2016; Parizi et al., 2018).***

1.1 Smart Contracts

Vending machines are referred to as the oldest piece of technology comparable to smart contract implementation (Savelyev, 2017).

In the early 1990s, smart contracts were first proposed by Nick Szabo, who coined the term, using it to refer to "a set of promises, specified in digital form, including protocols within which the parties perform on these promises" (Schulpen, 2018).

However, Buterin defines the Bitcoin protocol as a weak version of the smart contract definition. Since the rise of Ethereum, various cryptocurrencies support scripting languages that allow for more advanced smart contracts between unreliable parties (Alharby and Van Moorsel, 2017). Smart contracts should be distinguished from smart legal contracts. Smart legal contracts refer to a conventional natural language legally binding agreement enforced in a machine-readable code (Cannarsa, 2018; Filatova, 2020).

A smart contract is a computer program or transaction protocol designed to automatically conduct, manage, or record legal events and activities in compliance with the terms of a contract or agreement (Röscheisen et al., 1998; Savelyev, 2017). The goals of smart contracts are to minimize the need for trusted intermediaries, arbitration and enforcement costs, fraud losses, and reduce malicious losses (Röscheisen et al., 1998).

> A smart contract is defined by the US National Institute of Standards and Technology as a collection of code and data (sometimes referred to as functions and state) that is deployed using cryptographically signed transactions on the blockchain network (Yaga et al., 2019).

In this interpretation, used for instance by the Ethereum Foundation (Buterin et al., 2014) or IBM (Cachin et al., 2016), a smart contract is not inherently linked to the classical smart contract. As its implementation and codified effects, such as the transfer of any value between parties, are strictly implemented and can not be manipulated, a smart contract can also be viewed as a protected, stored process after a transaction with unique contract information is stored a blockchain or distributed ledger. This is because the actual execution of contracts, not any arbitrary server-side programs connecting to the platform, are managed and audited by the platform (Vo, Kundu, and Mohania, 2018; Huckle et al., 2016).

1.2 Problem Statement

Various issues surround the relatively new area of the production of smart contracts, and these problems are very diverse in nature. For instance, smart contracts are *immutable*, meaning that the contracts are strictly implemented and can not be manipulated after deployment. Another concern with smart contracts is *transferring domain knowledge* to developers. There is no clear way to determine at which stage the issues are nested in the development stage. Formalization of requirements could add structure to reduce the transformation of domain information concerns. There are currently little to no advanced, formalized approaches to this area of development. This leads to the following pitfalls parallel to those in software engineering's overarching sector. One of these drawbacks is that the software developer's mental work fully translates the domain concepts to software technology concepts (Stahl, Voelter, and Czarnecki, 2006). This also results in a misalignment of goods and specifications. Model-Driven Engineering (MDE) may help overcome this misalignment while helping the developer build higher-quality smart contracts that account for vulnerabilities at the same time. MDE is a software engineering approach that uses models and adjustments between models to support transferring domain information and software product (Afonso, Vogel, and Teixeira, 2006).

Furthermore, smart contracts can contain vulnerabilities that could pose a danger for the organizations acting with them. Vulnerability issues in smart contracts are seen from two separate viewpoints, namely from the developer's point of view and from the point of view of the domain expert. Developers have difficulty switching to the new smart contract architecture strategy, and domain experts do not have the technical skills to convert existing contracts into smart contracts. Many of these problems seem to emerge from a lack of understanding of Solidity's programming language and a general *lack of programming expertise* in the area of smart contracts. The greatest benefits of smart contracts are that they hold the promise of low legal and transaction costs relative to conventional financial contracts and can lower the entry bar for users. However, this barrier of entry for users remains high through the complexity of smart contract growth.

To summarize, smart contracts contain security vulnerabilities caused by the complexity of developing smart contracts. This issue stems from the transformation of domain information to developers. In addition, there is a compelling need to study smart contracts to find and address vulnerabilities before deployment on a ledger due to its immutability. Offering a standardized solution to participants in the smart contract creation process.

1.3 Research Focus

Many smart contracts may contain security vulnerabilities caused by the complexity of developing smart contracts. This issue stems from the transformation of domain information to developers. In addition, there is a compelling need to study smart contracts to find and address vulnerabilities before deployment on a ledger due to its immutability. Offering a standardized solution to participants in the smart contract creation process, such as a method, may lift the concerns addressed in the problem statement. Therefore, this study aims to gain insight into the decision rules of smart contracts through agent-based modeling by simulating systems requirements.

We hypothesize that smart contract design processes can be supported by agent-based modeling

1.3.1 Agent-Based Modeling

In today's high-tech age, one naturally assumes that US President Barack Obama's economic team and its international counterparts use sophisticated quantitative computer models to guide us out of the current economic crisis. They are not. The best models they have are of two types, both with fatal flaws. Type one is econometric: empirical statistical models that are fitted to past data. These successfully forecast a few quarters as long as things stay more or less the same but fail in the face of great change. Type two goes by the name of 'dynamic stochastic general equilibrium. These models assume a perfect world, and by their very nature, rule out crises of the type we are experiencing now. As a result, economic policy-makers base their decisions on common sense and anecdotal analogies to previous crises such as Japan's 'lost decade' or the Great Depression. People on Wall Street are using fancy mathematical models. However, for a completely different purpose: modeling the potential profit and risk of individual trades. There is no attempt to assemble the pieces and understand the behavior of the whole economic system. There is a better way: agent-based models.

An agent-based model is a computerized simulation of a number of decision-makers (agents) and institutions, which interact through prescribed rules.

The agents can be as diverse as needed, from consumers to policy-makers, and the institutional structure can include everything from banks to the government. Such models do not rely on the assumption that the economy will move towards a predetermined equilibrium state, as other models do. Instead, at any given time, each agent acts according to its current situation, the state of the world around it, and the rules governing its behavior. An individual consumer, for example, might decide whether to save or spend based on the rate of inflation, his or her current optimism about the future, and behavioral rules deduced from psychology experiments. The computer keeps track of the many agent interactions to see what happens over time. Agent-based simulations can handle a far wider range of nonlinear behavior than conventional equilibrium models. Policy-makers can thus simulate an artificial economy under different policy scenarios and quantitatively explore their consequences.

Agent-based models potentially present a way to model the financial economy as a complex system while taking human adaptation and learning into account. Such models allow for creating a virtual universe in which many players can act in complex and realistic ways. In some other areas of science, such as epidemiology or traffic control, agent-based models already help policy-making.

To better understand the utility of agent-based modeling, consider one of the earliest and best-known models, created by Schelling.

> Schelling wanted to test the theory that segregated neighborhoods can arise not just by active racism but due only to a mild preference for neighbors of the same ethnicity (Schelling, 1971). The model consists of majority-group and minority-group agents living on a grid who prefer only several neighbors of the same group. When that preference is not met, they move to a different grid cell. The model demonstrates that even a mild preference for same-group neighbors leads to a dramatic degree of segregation.

This is an example of the emergence of higher-order phenomena from the interactions of lower-level entities and demonstrates the link between agent-based modeling and complexity theory, and complex adaptive systems in particular (Miller and Page, 2009).

Summarizing the foregoing, an agent-based model (ABM) is a class of computational models for simulating the actions and interactions of agents to assess their effects on the system as a whole. Agent-based modeling involves simulating the behavior and interaction of many autonomous entities or agents over time. Agents are

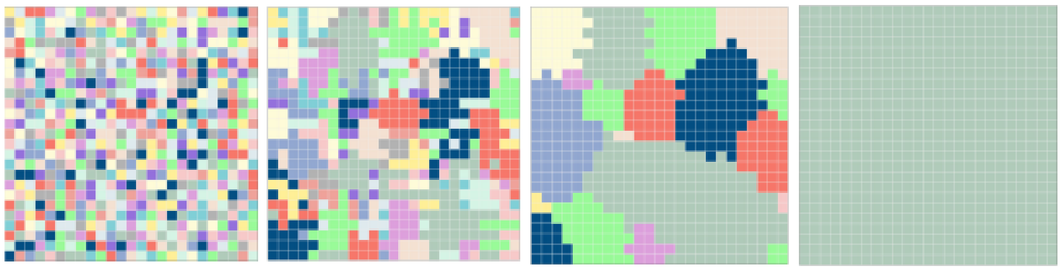


FIGURE 1.1: The agent-based model demonstrates that segregated neighbourhoods can arise due to a mild preference for neighbors of the same ethnicity over time through a population. (a) Timestep = 0 (b) Timestep = 3 (c) Timestep = 35 (d) Timestep = 137.

objects that have rules and states and act accordingly with each step of the simulation (R. Axtell, 2000). These agents may represent individual organisms, humans, entire organizations, or abstract entities.

Therefore, this research aims to support the design of smart contracts through agent-based modeling by simulating system requirements.

In conclusion, to realize the aim of this research, a series of sub-questions guide this study to answer how ABM can help translate a utility function to study the behavior of users interacting with the system. Four deliverables are required to enable this case.

1. A decentralized system, serving as an influencing agent. For example, agents interact with the central system but react differently based on the central system's incentives.

2. A data set containing user behavior. Users might interact differently with a system based on their behavior. Therefore, a data set containing users' behavior could better understand how to tweak the central system to achieve optimal utility with that system.

3. One, but preferably more smart contracts.

4. A utility function, which goal is to translate if the smart contract is successful or not. For example, if we want to enable a system's utility to reach 90%, the utility function will provide an overview of the agent-based simulation to summarize its outcome.

Through this study, we envision a decentralized system that has a rewarding mechanism that contains a smart contract. This incorporates a utility function that allows measuring the influence of the behavior of users interacting with the systems.

1.4 Contributions

Work described in this study adds value in a variety of ways to the knowledge base. First, it provides a holistic description of the principles of agent-based modeling and smart contracts and an overview of the current state of research in applying ABM to smart contracts. Work on ABM and smart contracts are partially embedded in research. Still, for the most part, it is conducted in an online open-source environment in which willing participants collaborate and expand on the work of others without the need for detailed documentation. For this purpose, it is valuable to have a scientifically written overview of the current state of smart contract design. Second, smart contract developers are encouraged by the insights presented in this research. Smart contract developers should strive to build high-quality smart contracts as efficiently as possible. The development method supporting this goal could contribute to the smart contracts' efficiency, performance, and quality.

1.5 Thesis Outline

This research includes the following contributions. The research methodology, including the research questions, the research paradigm, and the literature research protocol for this thesis, is described in Chapter 2. Chapter 3 provides concrete steps to construct a literature review, including identifying the sources needed, finding the sources, a systematic method for summarizing and synthesizing the source, and organizational and writing strategies for review to generate an excellent literature review. Chapter 4 provides an overview of agent-based modeling characteristics used to evaluate a range of various tooling and simulation programs currently available. Chapter 5 discusses the preparation and selection of the case study. Thereafter the simulations are performed in Chapter 6. Thereafter, in Chapter 7, the findings of three scenarios are analyzed. Finally, the strengths and limitations of the study are discussed, as well as recommendations for future research in addition to the conclusion, in Chapter 8 and 9 respectively.

Chapter 2

Research Approach

This chapter explains the research approach utilized in this study. It opens by addressing the decision to adopt the approach of Design Science Research (DSR) and elaborates on how aspects of this analysis integrate into the life cycle of the approach. Visualization of the research methodology is presented through a Process Deliverable Diagram (PDD) and presented in Figure B.1 in Appendix B. A PDD consists of two interconnected diagrams. The process view on the left-hand side is based on a UML activity diagram, whereas the right-hand side of the diagram displays the deliverable based on a UML class diagram (Weerd and S. Brinkkemper, 2009).

2.1 Research Questions

The design problem template presented by R. J. Wieringa (2014) structures the objectives of the study through the problem context, artifacts, requirements, and desired impact of the study as follows. To achieve the research objective, the key research questions and the five sub-questions are proposed by the goal statement. This main research question aims to evaluate the rules used by smart contracts through agent-based modeling by simulating system requirements.

> Research Question: Can agent-based modeling support the design of smart contracts by simulating the systems requirements?

A selection of sub-questions needs to be addressed to answer this research question. The first sub-question is to understand the principles used in this study related to smart contracts and ABM. A literature review is performed to understand the principles related to smart contracts and agent-based modeling. The literature review is done by reviewing previous studies on the subjects through a study on the concepts. This results in the following sub-questions:

> Sub-Question 1: How can smart contracts and agent-based modeling be defined based on prior literature?

The knowledge resulting from research sub-question 1 offers a summary of the concepts. A study on the overlapping concepts will be the next step in this research, with a clear definition and understanding of these concepts.

Based on initial research, it is expected that attempts have already been made to combine the development of smart contracts with Model-Driven Engineering (MDE), but not yet ABM in particular nor in a formalized structured way this thesis intends to provide. The second sub-question this research proposes is as follows:

> Sub-Question 2: What research into the application of agent-based modeling to smart contract development has already been conducted?

The perspectives of SQ1 and SQ2 will summarize the current state of smart contract design and ABM research. Using these insights, a method engineering process can be initiated. A method engineering approach focused on the use of existing method fragments is applied. This means that a method base is generated, and the method fragments from this method base are determined based on the criteria for the Smart Contract Development Method. To pick the most suitable fragments, it is important to set out the criteria for the method. Based on the specifications and available system fragments, the method engineering phase will lead to the development of the method. Method engineering will be discussed in a later chapter and used to build a better contract. The criteria, activities, and deliverable of this approach will be discussed in the third research sub-question:

> Sub-Question 3: What are the requirements for agent-based modeling and smart contract development?

A process deliverable diagram (PDD) that describes the process of agent-based modeling to construct smart contracts will be the outcome of SQ3. The PDD describes the activities and deliverables of agent-based modeling and smart contract development.

> Sub-Question 4: What is the most suitable agent-based modeling tool to support the development of the goal's requirements?

There exists a vast amount of ABM tools, each with its strengths and weaknesses. Some tools focus on the models' scalability level, whereas others model development effort, ease of use, or other features. Therefore, an assessment of the existing tools is required to allow for proper tool selection. This sub-question is answered in section 3.1.6.

> Sub-Question 5: Can agent-based modeling assist in creating insight into the decision rules of a smart contract for a complex system?

The goal of Sub-Question five is to investigate whether the system achieves the desired objectives. An assessment of the method is required. The main objectives of this study, defined in chapter 1.2, are (i) to bridge the semantic gap between domain knowledge and smart contract by lowering the domain expert threshold, and (ii) to help developers create less uncertain smart contracts that accurately represent the problem domain. The first objective will be assessed through a case study in which the procedure is demonstrated. The second objective will be assessed through a statistical analysis in which the interpretation will be evaluated.

To summarize, the sub-research questions will be resolved using the following approaches:

1. Sub-Question 1 is answered through Multivocal Literature Review (MLR). MLR is a systematic literature review (SLR) that involves gray literature and published (formal) literature. The literature review protocol is detailed in section 2.3.

2. Sub-Question 2 is also answered by MLR based on the information gathered from Sub-Question 1. The literature dealing with the interrelation between concepts is discussed in sub-question 2. Attempts to apply ABM to smart contracts will form a method base from which fragments of the method will be selected.

3. Sub-Question 3 provides the basis for the method by setting out the objectives of the method and the appropriate method fragments that can be used to achieve those objectives. These objectives and process fragments are the specifications for the method. Based on these criteria, a detailed overview of the agent-based smart contract optimization method will be given. Knowledge from Sub-Question 1 and Sub-Question 2 will be connected to generate a method that ranges from domain knowledge to the fine-tuning of the smart contract.

4. Sub-Question 4 is again answered through MLR. An assessment of the existing tools is required to allow for proper tool selection.

5. Sub-question 5 is answered by discussing the case study and its effects seen as a result in the agent-based model. The case study aims to evaluate the semantic gap between domain knowledge and the smart contract aspect. The experiment follows the goal-question-metric approach, which follows the experimental design framework.

Answers to the research sub-questions should provide a holistic view of creating and evaluating the model-driven Smart Contract Development Method, which will provide a context to address the key research question.

2.2 Research Paradigm

This study aims to support the design of smart contracts through agent-based modeling by simulating systems requirements.

Research is defined by either behavioral science or design science in the discipline of information systems (March and Smith, 1995). The paradigm of behavioral science seeks to establish hypotheses that explain or predict the actions of individuals or organizations. The science of design is a paradigm driven by the desire to better the world by introducing new and creative objects and constructing these artifacts (March and Smith, 1995). Behavior and design should not be seen as two distinct paradigms, although in the definition of theories (behavior) and usefulness (design), they overlap (A. R. Hevner et al., 2004). The motivation for design science is in line with the motivation for this study, as the goal is to strengthen the field of smart contract creation by adding to this field a novel artifact.

Figure 2.1 presents the DSR methodology adopted from A. Hevner and Chatterjee (2010) which serves as a guideline for this study to create a scientifically qualitative artifact.

There are three sub-cycles in Design Science Studies. The cycle of relevance, the cycle of design, and the cycle of rigor. The cycles are based on information learned during each preceding iteration. Design Science Analysis seeks to solve issues in the real world, hence why this approach suits the thesis (R. J. Wieringa, 2014). The cycle of relevance offers context, requirements, and acceptance criteria used to assess the study results. It also involves field testing of the prototypes that were produced during the design cycle. Finally, the rigor cycle provides the basis for the cycle of design science through literature review and knowledge base development. Information acquired through the rigor cycle provides feedback for the production of artifacts in the design cycle. Consequently, designs are evaluated against specifications and acceptance criteria from the relevance cycle until a final design is reached.

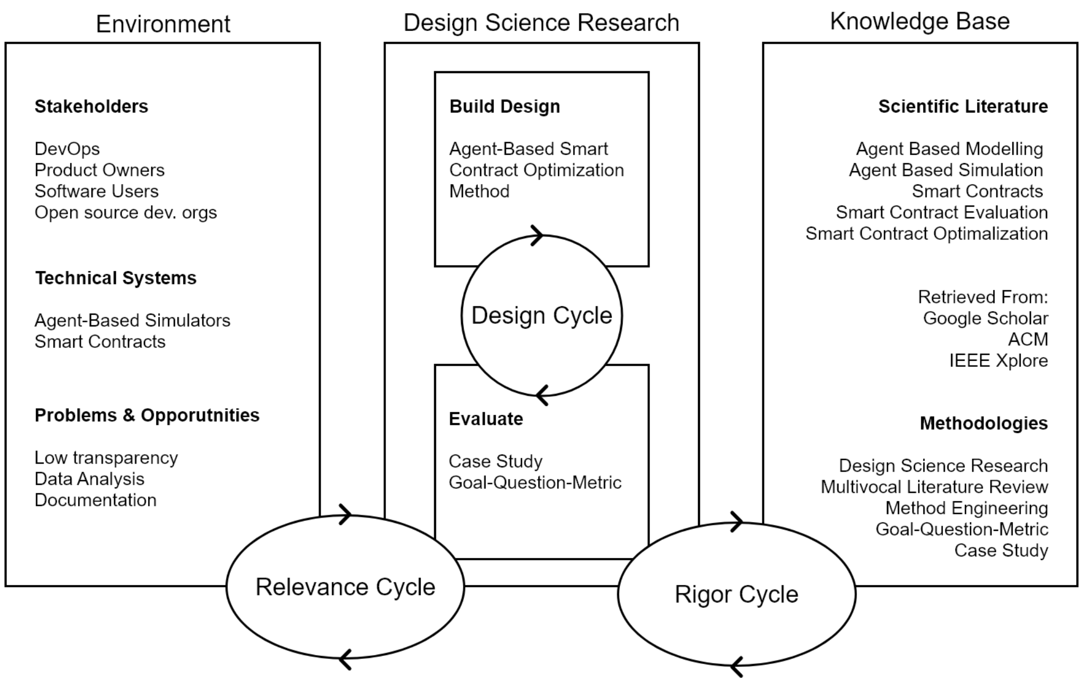


FIGURE 2.1: The Design Science Research framework seeks to solve issues in the real world though a cyclic process consisting of three sub-cycles A. Hevner and Chatterjee (2010, p. 16). The cycle of relevance, the cycle of design, and the rigor cycle are based on information learned during each preceding iteration.

2.2.1 Research Methods

The main questions of the research and its sub-questions described in section 2.1 are revealed during the iterative phase of designing and evaluating the method. This study uses four approaches to answer the aforementioned questions, as shown in Table 2.1. A Multivocal Literature Review is the first approach used to identify and map key concepts and sub-concepts required for the method design. Using the insights from SQ1 and SQ2 through MLR, a method engineering approach is taken to lead to the development of the method. Furthermore, MLR is used once again to provide decision support for the best suited ABM tool, which will answer SQ4. Data from a case study is used to answer and analyze SQ5 through Statistical Analysis.

TABLE 2.1: A comparison the three methods used to address the sub-questions of this study.

Method	SQ1	SQ2	SQ3	SQ4	SQ5	Section
Multivocal Literature Review	✓	✓	✗	✓	✗	2.3
Method Engineering	✗	✗	✓	✗	✗	2.4
Statistical Analysis	✗	✗	✗	✗	✓	
Case Study	✗	✗	✗	✓	✓	2.5

2.3 Literature Review Protocol

To plan and conduct the literature review, the popular systematic review guideline of Kitchenham and Charters (2007) is combined with the review guidelines of Garousi, Felderer, and Mäntylä, 2019. The purpose of the guideline by Kitchenham and Charters (2007) is to provide detailed recommendations for systematic literature reviews suitable for researchers in software engineering, including Ph.D. students. A systematic review of the literature reviews and analyzes all existing studies related to a specific study's issue, subject field, or unusual phenomena. Systemic reviews aim to present a fair assessment of a study subject using a trustworthy, systematic, and audit-able approach. The guidelines have been adapted to reflect the specific problems of software engineering research. The recommendations cover three stages of comprehensive literature analysis: planning of reviewing, carrying out the review, and reporting. To suit the study, this analysis translates these instructions into four steps: the planning process, selection process, classification process, and the review process.

2.3.1 Phase 1: Multivocal Literature Review Planning

According to Garousi, Felderer, and Mäntylä (2019) existing guidelines for conducting systematic literature studies in Software Engineering (SE) provide limited coverage for including practitioners' sources and conducting multivocal literature reviews (MLR), this paper filled that gap by developing and presenting a set of experiential guidelines for planning, conducting and presenting MLR studies in SE. Consequently, a review of academic and GL is necessary to fill any knowledge gaps left by the lack of academic literature. The incorporation of GL enables the researcher to benefit from various positive contributions from material produced by real-life practices (M. Adams et al., 2014). The criteria for performing an MLR as set out in Garousi, Felderer, and Mäntylä (2019) are followed to integrate academic and GL into the literature review. Garousi, Felderer, and Mäntylä (2019) defines MLRs as a type of Systematic Literature Review that includes GL and literature written.

2.3.2 Phase 2: Search Process and Source Selection

Step two of the literature review consists of three critical tasks during the selection phase. In this process, the following steps were followed:

- Selecting source engines and search keywords
- Application of inclusion/exclusion criteria
- Final Pool of sources

Selecting source engines and search keywords

As for the gray literature search engine, many literature reviews in other fields, recommendations, and experiential papers such as Godin et al. (2015), Mahood, Van Eerd, and Irvin (2014), and J. Adams et al. (2016) suggested using the standard *Google search engine* which is used in this work. The search string is built iteratively to ensure optimizing coverage for all related sources on the internet. The initial search string was: "agent-based modeling smart contracts" After one round of searches in the Google engine and Google Scholar, a need to expand the search string to identify and locate all the related sources.

To optimize the scope for relevant sources, synonyms of the word decentralized and technology are interchanged. After a few iterations, the final search string adopted to be:

Search-Query:*(agent-based modeling OR agent-based simulation) AND (smart contract OR smart contracts) AND (optimization OR evaluation)*

The pool would have missed many relevant sources without the broadening of the query. When searching Google Scholar for published literature, Google Scholar returned 19.300 results for the above query as of this writing. To cope with the candidate sources, traversing on Google Scholar results is done on several consecutive pages. Actual candidates are only added to the candidate pool if the inclusion/exclusion criteria mentioned in the next section are met. To collect grey literature, the same query to restrict the search space is used. A total of 3.640.000 results are observed upon query execution at this time of writing. Relevant results and candidate sources are included based on the exclusion criteria mentioned in the next section. Forward and backward snowballing is performed to ensure that every applicable technical paper is included as much as possible on a selection of papers already in the pool as suggested by Claes Wohlin, 2014. Snowballing refers to using a reference list of a paper (backward snowballing) or the citation to the paper to identify additional papers (forward) (Claes Wohlin, 2014).

Application of inclusion/exclusion criteria

A Quality Evaluation Checklist, including exclusion criteria, is used to ensure the validity and integrity of the literature. The evaluation is tailored to this study, as indicated by M. Adams et al. (2014). Table 2.2 indicates the criterion used during evaluation and the questions used to evaluate the requirements. Furthermore, literature that is not written in English and is not openly accessible is excluded without scoring. Requirements are addressed with a score of 1 for *YES* and a 0 for *NO*.

TABLE 2.2: The quality evaluation checklist used in applying M. Adams et al., 2014 criteria from grey literature.

Aspect	Questions
Academic Literature	• *"Is the literature relate to one of the concepts?"* • *"Is the literature written in English?"* • *"Is the literature openly accessible?"*
Authority	• *"Is an individual author associated with a reputable organization?"* • *"Has the author published other work in the field?"*
Accuracy	• *"Does the source have a clearly stated aim?"* • *"Does the source have a stated methodology?"* • *"Is the source supported by authoritative and documented references?"*
Coverage	• *"Are any limits clearly stated?"* • *"Does the work cover a specific question?"* • *"Does the work refer to a particular population?"*
Objectivity	• *"Does the work seem to be balanced in the presentation?"* • *"Is the statement a subjective opinion?"* • *"Are the conclusions biased?"*
Date	• *"Does the item have a clearly stated date?"*
Significance	• *"Does it enrich or add something unique to the research?"* • *"Does it strengthen or refute a current position?"*

Final Pool of sources

The scores from the quality assessment questions are summed up and standardized by dividing them through the total number of questions. A threshold score of 50 percent is maintained. Once the literature ranking is above the mark, it is included in the final pool of sources. This review took place in iterations as the knowledge base grew, requiring the Snowballing process to roll back to remove incorrectly added papers and adjust search parameters. Online mapping repository and details of why each source was excluded can be found in the Google Spreadsheet of this study online.

2.3.3 Phase 3: Data Classification

To develop a systematic map, the studies in the pool are analyzed, and the initial list of attributes is identified. Thereafter, attribute generalization, and iterative refinement are used to derive the final map. As studies were identified as relevant to this study, are record was created in an online spreadsheet to facilitate the work and further analysis. The next goal is to categorize the studies to build a complete picture of the research area and answer the RQs. These broad interests are refined into a systematic map using an iterative approach.

2.3.4 Phase 4: Synthesis and Review

The instructions for performing a thematic synthesis are taken from Cruzes and Dyba (2011) to ensure the data synthesis process produces the right building blocks for the framework. Braun and Clarke (2012) define thematic synthesis as a tool for the systematic detection and development of insights into context through a data set. High-level modules may be modified for the system depending on the infrastructure the application needs to support. During this process, sub-concepts for the various high-level concepts are defined. This method is suitable as it provides optimal flexible procedures for researchers, addresses research questions about need, appropriateness, and effectiveness, and copes well with pattern identification across the various subjects. This thematic research meets instructions by Cruzes and Dyba (2011) encouraging the researcher to analyze the data before a system draft is feasible gradually.

2.3.5 Summary

MLR will be used to answer Sub-Question 1, Sub-Question 2, and Sub-Question 4. MLR will answer Sub-Question 1 and Sub-Question 2 by creating a summary of concepts with clear definitions and will provide a better understanding of smart contracts and agent-based modeling. Sub-Question 4 is also answered through MLR. The aim is to provide a long-list short-list approach an overview of the best suited ABM tools for this study.

2.4 Method Engineering

As mentioned earlier, a method engineering approach is used to focus on existing method fragments. Based on the specifications and available system fragments, the method engineering phase will lead to the development of the method.

Method Engineering is the engineering discipline to design, construct and adapt methods, techniques, and tools for developing information systems (S. Brinkkemper, 1996). Situational Method Engineering (SME) is a subset of method engineering, where development is adapted to the project at hand (Harmsen, J. N. Brinkkemper, and Oei, 1994). In this case, the project at hand is relatively loosely used as a method for the creation of all kinds of smart contracts. Small and medium-sized enterprises (SMEs) seek to identify information systems methods by reusing and assembling existing process fragments (Ralyté and Rolland, 2001).

2.4.1 Summary

SME will be used to answer Sub-Question 3, and assist with creating a PDD that will describe the process of ABM to construct smart contracts.

2.5 Case Study

Evaluation in Design Science Research (DSR) focuses on assessing the results of design science, including theory and artifacts. Walls, Widmeyer, and El Sawy (1992) put forward IS Design Theories at DSR output. March and Smith (1995) presented DSR's outputs as artifacts: constructions, templates, methods and instances. March and Smith (1995) also describe evaluation as one of two tasks in the DSR. A. R. Hevner et al. (2004) describe evaluation as crucial and enable researchers to demonstrate the usability, efficiency, and effectiveness of a design artifact using rigorous methods of evaluation.

A case study can be seen as a means to prepare for data collection, collecting evidence and analysis of collected data, and reporting. Case studies are more commonly exploratory, using qualitative data (Runeson and Höst, 2009). Runeson and Höst point out five key steps for a case study:

1. Case study design
2. Preparation for data collection
3. Collecting evidence
4. Analysis of collected data
5. Reporting

2.5.1 Case Study Requirements

1. Data set needs to be available containing user behavior.
2. Statements of the type of agents need to be defined.
3. Statements of the agent's behavior need to be defined.
4. Statements of the environment need to be defined.
5. The utility goal and threshold of the system need to be defined.
6. A goal could be to evaluate if we can optimize the usage of a system. For example, can we incentivize a system so that more people vote?

This chapter describes the research methodology, including the research questions, the research paradigm, and the literature research protocol for this thesis. The next chapter 3 provides a literature review covering agent-based modeling and smart contracts and how these two components can be bridged.

Chapter 3

Literature Study

The literature review performed for this proposal is discussed in this chapter. In chapter 2, the literature review protocol has been addressed previously. The first section deals with the different principles and meanings of agent-based modeling (ABM), smart contracts, and related terms. The following section continues with the features and forms of agent-based modeling that have been described for smart contracts and related subjects in the literature. Furthermore, a summary that describes the process and requirements of ABM is listed. An assessment of existing ABM tools for proper tool selection is conducted.

3.1 Agent-Based Modeling

3.1.1 Introduction

Agent-based modeling and simulation (ABMS) is a relatively new method composed of interacting, autonomous 'agents' to model complex systems. Agents are often defined by simple rules, behaviors, and relationships with other agents, affecting their behaviors. It is possible to observe the full effects of the diversity among agents in their attributes and behaviors by individually modeling agents. It gives rise to the system's behavior as a whole. In such models, self-organization can also be observed by modeling structures from the ground up agent-by-agent and interaction-by-interaction. There are patterns, mechanisms, and behaviors that are not directly programmed into the models but occur from interactions with other agents. As opposed to simulation techniques such as discrete event simulation and system dynamics, the focus on modeling the heterogeneity of agents and a population and the development of self-organization are two of the distinctive characteristics of agent-based simulation. Agent-based modeling provides a way to model social networks made up of agents that connect, influence each other, learn from their experiences, and adjust their actions to suit their environment better.

Agent-based modeling implementations cover a wide variety of fields and disciplines. Applications range from modeling the behavior of agents in supply chains (C. Macal, Sallach, and M. North, 2004) and the stock market (Arthur, 2018) to forecasting the hazard of biowarfare (Carley et al., 2006) and the spread of epidemics (Bagni, Berchi, and Cariello, 2002) and from modeling the adaptive immune system (Folcik, An, and Orosz, 2007) to understanding customers' buying behavior (M. J. North, C. M. Macal, et al., 2010), and many others. These developments have been made feasible by advances in the production of advanced agent-based modeling tools. New approaches to agent-based model development allow for advancements in computer efficiency and data availability at increasing granularity levels.

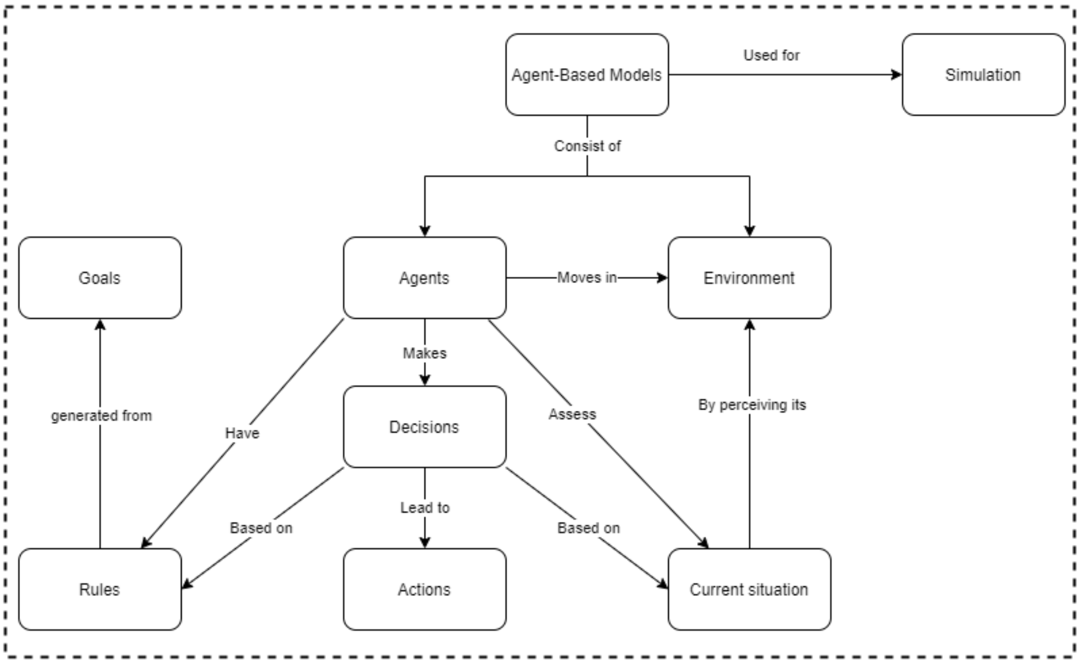


FIGURE 3.1: Visualizing the workings of an ABM through a concept diagram, derived from Hall and Virrantaus (2016)

Figure 3.1 displays the working of an agent-based model as a concept diagram. ABM aims to seek an explanatory insight into the collective behavior of agents following simple rules. Agent-based models are usually practiced in natural systems. Furthermore, agent-based models are a form of micro-scale model that simulates several agents' parallel activities and interactions to recreate and forecast the appearance of complex phenomena. The higher-level system properties arise from the lower-level subsystem interactions. Alternatively, basic behaviors (meaning rules followed by agents) create complicated behaviors (meaning state changes at the whole system level) (Bonabeau, 2002).

3.1.2 Agent-Based Modeling Requirements

ABMs are typically implemented as computer simulations, either as custom software or via ABM toolkits. This software can then test how changes in individual behaviors will affect the system's emerging overall behavior. In general, when we build an ABM to simulate a certain phenomenon, we need to identify the actors first (the agents). We then need to consider the processes (rules and relationships) governing the interactions among the agents. A typical agent-based model has three elements, as seen in Figure 3.2:

1. A set of *agents* with their attributes and behaviors. Moreover, in Section 3.1.3.
2. A set of agent *relationships* and interaction methods: An underlying topology of connectedness defines how and with whom agents interact. Moreover, in Section 3.1.4.
3. The agents' *environment*. Agents interact with their environment in addition to other agents. Moreover, in Section 3.1.5.

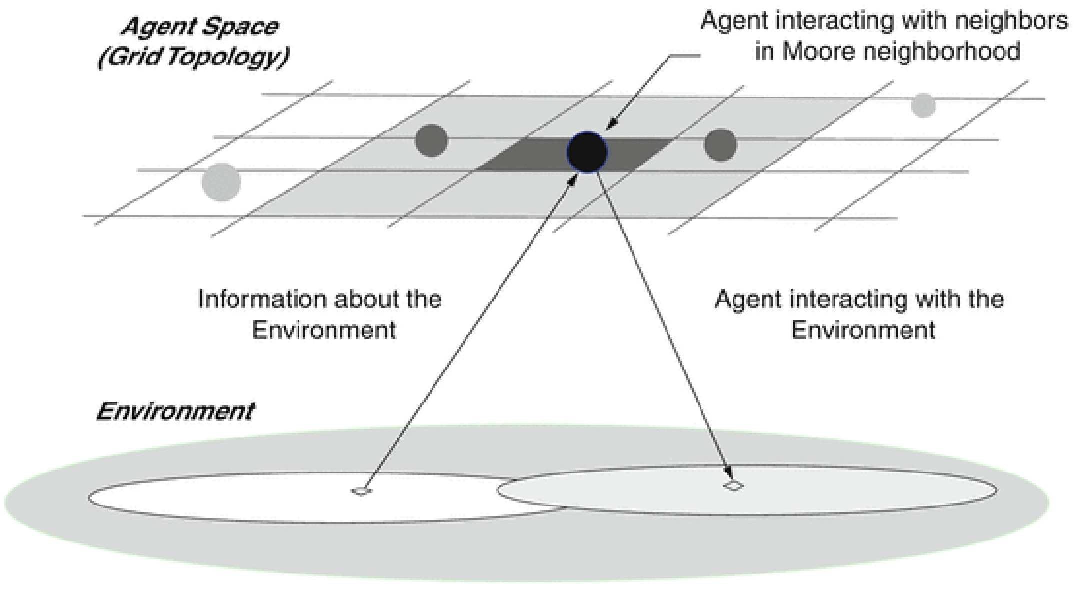


FIGURE 3.2: The structure and elements of an agent-based model, as in C. M. Macal and M. J. North (2005)

Modelers introduce some type of spatial landscape (e.g., lattice, torus) in many ABM applications that limits potential interactions with agents. This spatial environment is depicted as a strictly passive platform on which agents interact. However, in other cases, the spatial landscape is interpreted as an entity with its own internal states and behavioral laws (e.g., a region of land with naturally growing food sources).

To Summarize: Agent-Based Modeling Requirements

A model developer must define, model, and program *agents, relationships, and environment* to construct an agent-based model. In Figure 3.2, the structure of a typical agent-based model is shown. Sections 3.1.3, 3.1.4 and 3.1.5 address each of the components in Figure 3.2. A computational engine is then necessary to simulate agent behaviors and agent interactions to make the model work. This functionality is provided by a programming language, an agent-based modeling toolkit, or other implementation. To run an agent-based model is to have its activities and interactions executed by agents repeatedly. This approach frequently functions over a timeline, as in time-stepped, activity-based, or discrete-event simulation systems, but is not necessarily modeled.

3.1.3 Autonomous Agents

There is no general consensus between researchers on the exact meaning of the word agent, with researchers constantly arguing whether the definition should be by the application or environment of an agent. However, definitions appear to agree on more points than they disagree (C. M. Macal and M. J. North, 2005). Diversity in its implementation makes it difficult to extract agent characteristics from the literature coherently and concisely, as an agent-based model is mostly defined from the perspective of its constituent parts (Bonabeau, 2002).

According to Wooldridge and Jennings (1995), there are many characteristics common to most agents from a pragmatic modeling point of view. This standpoint is further clarified and expanded by Franklin and Graesser (1996) and C. M. Macal and M. J. North (2005). These characteristics are presented briefly below:

- **Self-Contained**: An agent is an entity that is self-contained, modular, and uniquely identifiable. The modular criterion means there is a boundary for an agent. Agents have characteristics that allow other agents to differentiate the agents from and identify them.
- **Autonomous**: Agents are autonomous units (i.e., managed without centralization), capable of processing and sharing information with other agents for independent decision-making purposes. At least in a limited range of cases, they can communicate with other agents, which does not (necessarily) affect their autonomy.
- **State**: An agent has a state that changes over time. As a system has a state comprising state variables, an agent often has a state representing the main variables associated with its current situation. The state of an agent is made up of a set or a subset of its attributes.
- **Heterogeneity**: Agents allow autonomous individuals to evolve, e.g., an agent representing a human being may have characteristics such as age, sex, work, etc. There could be groups of agents, but they are spawned from the bottom-up and embody similar autonomous individuals.
- **Active**: Agents are active because, in a simulation, they have independent influence. These agents active features can be:
  - Goal-directed: Agents are also called goal-oriented, having goals to accomplish (not necessarily goals to maximize) in terms of their behavior.
  - Perceptive: agents can be constructed with an awareness of their surroundings.
  - Bounded Rationality: agents can be constructed through rational choice models with access to data, foresight, and analytical abilities to allow for adaptive choices that move them towards achieving goals (Parker et al., 2003).
  - Adaption/Learning: Agents can also be built to generate Complex Adaptive Systems to be adaptive (Holland, 1996). Agents may be programmed to alter their state, allowing agents to adapt to a mode of memory or learning based on previous states.

This list is not exhaustive or exclusive; other characteristics can occur within the application agent, and, for certain applications, some features may be more relevant than others (Wooldridge and Jennings, 1995). Sometimes, there are several different types of agents within a single simulation. However, the characteristics depicted above are considered essential for this study from a practical modeling standpoint. In Figure ,3.3 a typical agent structure is illustrated. An agent-based model, an agent attribute, or even an agent method that operates on the agent is associated with an agent. As the simulation progresses, agent attributes could be static, meaning they are not subject to change during the simulation, or dynamic, meaning they can change. A static attribute, for example, is the name of an agent; a dynamic attribute is an agent's memory of past experiences. Methods of agents include behaviors like rules or more abstract representations like neural networks that connect the agent's

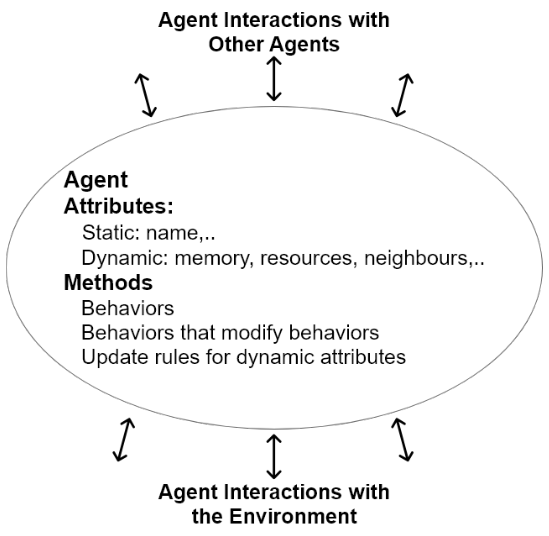

Agent Interactions with Other Agents
Agent
Attributes:
Static: name,..
Dynamic: memory, resources, neighbours,..
Methods
Behaviors
Behaviors that modify behaviors
Update rules for dynamic attributes
Agent Interactions with the Environment

FIGURE 3.3: A typical agent structure modified from (C. M. Macal and M. J. North, 2005).

situation to its action or a set of possible actions. The method that an agent uses to classify its neighbors is an example.

Agents may be representations of an autonomous body of some kind. For example, this may be persons, plots of land, houses, vehicles, insects, or water droplets. It should be noted that, while the object-oriented paradigm offers an effective medium for the creation of agent-based models, ABM is not the same as object-oriented simulation. ABM systems are invariably object-oriented for this purpose (Gilbert and Terna, 2000).

A collection of multiple interacting agents within a model or simulation environment, such as the artificial world, is called an agent-based model. Agents may be depictions of *animate entities* such as humans who can roam freely around, or agents may be *inanimate entities* such as a petrol store, which has a fixed location but can alter the state of rules or prices.

To Summarize: Autonomous Agents

Typically, individual agents are characterized as boundedly rational, assumed to be acting in what they consider their own interests, such as reproduction, economic gain, or social status, using heuristics or simple rules for decision-making (R. L. Axtell, Andrews, and Small, 2003). ABM agents may have learning knowledge, adaptation, and reproduction (Bonabeau, 2002).

3.1.4 Rules, Behavior, and Relationships

Each of the above-mentioned inanimate and animate agents has rules that will influence their behavior and relationships with other agents and/or their environment. Usually, laws are derived from expert experience, written literature, numerical work, or data analysis and are the basis of the behavior of an agent.

You can apply one rule-set to all agents, or each agent (or agent category) can have its own specific ruleset. For instance, in A. Heppenstall, A. Evans, and M. Birkin (2006) retail petrol agents all work on the same basic rule set based on a desire to optimize profits. In addition to their own "realistic" rule-sets based on reported actions, data analysis, and numerical analysis, various forms of retailer agents, such as supermarkets, international, national, and independent stations, were seen in further work (A. Heppenstall, A. Evans, and M. Birkin, 2006).

Typically, rules are based on if-else statements with agents taking action after a specified condition has been met. However, in ignorance of other agents' actions, rules may be implemented. Agents can also be integrated into evolutionary computation with a notion of learning and 'intelligence' (Alison J Heppenstall, Andrew J Evans, and Mark H Birkin, 2007).

More recently, there has been a shift towards integrating behavioral systems to reflect human behavior within agent-based models better. For instance, to reflect the motives and desires of offenders, Malleson, A. Heppenstall, and See (2010) used the PECS (Physical conditions, emotional states, cognitive capacities, and social status) paradigm. This style of work represents a step towards more nuanced handling of the actions of agents. In agent-based models, Kennedy (2012) offers an overview of various mechanisms for managing human behavior.

To Summarize: Rules, behavior, and Relationships

Agents can communicate with each other and with the world. Relationships can be characterized in several ways, from purely reactive (i.e., agents only conduct acts when some external stimulus triggers them to do so, e.g., actions by another agent) to goal-directed actions (i.e., seeking a particular goal). Agents' activity can be scheduled to occur synchronously (i.e., each agent performs actions at each specific time phase, all adjustments occur simultaneously) or asynchronously (i.e., agent actions are scheduled by other agents' actions and/or clockwise).

3.1.5 Agent Environments

Environments describe the area in which agents work to facilitate their interactions with the environment and other agents. For instance, relying on the defined space for agent interactions, proximity can be defined by continuous space spatial distance, grid cell adjacency, or social network connectivity. Agents can be spatially explicit within an area, meaning agents have a position in geometric space, while the agent itself may be static. For instance, agents will be required to have a particular location to test their route strategy within a route navigation model. On the other hand, agents within an environment can be spatially implicit; this implies that their location is meaningless.

Agent-based models may be used in a modeling context as research tools for conducting and analyzing agent-based simulations. To this point, they can be regarded as a miniature laboratory where agents' characteristics and behavior, and the atmosphere in which they are located, can be altered and the impacts observed over several simulation runs.

The ability to simulate individual activities of several different agents and monitor the resulting system behavior and results over time (e.g., shifts in traffic flow patterns) means that agent-based models can be useful tools to research the impact on multi-scale and organizational-level processes (Brown and Geist, 2006). ABM's origins are in particular within the simulation of human social activity and individual decision-making (Bonabeau, 2002). ABM has transformed social science research in this sense by enabling scientists to replicate or produce the development of scientifically complex social phenomenon from a collection of relatively simple micro-level agent-based rules (Luke et al., 2003).

To Summarize Agent Environments

Environments are the space inside which there are agents. An environment could be abstract (empty space) or a layer representing actual geographical information from your GIS. There are many environments in most simulations. Environments can either be dynamic or static. When an environment is dynamic, it will change over time.

3.1.6 Methods for agent-based Modeling

Building an ABM can be facilitated by using an object-oriented programming language, modeling development toolkits, and platforms. These methods are briefly discussed here, explaining their benefits and drawbacks. However, it is helpful to ask a set of questions while designing an agent-based model, contributing to an initial agent model design.

(a) Agent model design

1. Which particular problem should the model solve? **What particular questions should be answered by the model**? What added value can ABM bring to the problem that other approaches do not bring?
2. What should the agents be in the model? Who are the **decision makers** in the system? What are the entities that have **behaviors**? What data on agents are simply descriptive (static attributes)? What agent attributes would be calculated endogenously by the model and updated in the agents (dynamic attributes)?
3. What is the agents' **environment**? How do the agents interact with the environment? Is an agent's mobility through space an important consideration?
4. What agent **behaviors** are of interest? What **decisions** do the agents make? What behaviors are being **acted upon**? What **actions** are being taken by the agents?
5. How do the agents **interact** with each other? With the environment? How **expansive** or focused are agent interactions?
6. **Where might the data come from**, especially on agent behaviors, for such a model?
7. How might you **validate** the model, especially the agent behaviors?

An integral part of the ABM design process is to address these questions. To design and incorporate agent-based models, there are several approaches. M. J. North and C. M. Macal (2007) discusses in detail both methodologies of design and chosen environments of implementation. An initial methodology for identifying agent behaviors inside a framework for unmanned autonomous vehicles is given by Marsh and Hill (2008). Overall, bottom-up, iterative development design methods seem to be the most successful for functional model creation.

Although developing from the ground up enables full control over any element of the agent-based model, unless the scientist is an experienced programmer, this could be a time-consuming choice. Implementation of the model can be tedious, and it is possible to spend considerable time on non-content-specific elements like graphical user interfaces, presentation, and import of data.

(b) Agent model implementation

ABM can be achieved using general, all-purpose tools or programming languages or can be done using specifically developed toolkits that solve the special requirements of agent modeling. ABM can be performed on a small scale, on a laptop, large scale, or any scale between these extremes. Projects typically begin simple, using one of the ABMS desktop tools, and then develop into larger ABMS toolkits in stages. General programming languages such as Java, Python, C++, and C can also develop a simulation. However, development from scratch may be prohibitively costly, considering that this will entail developing many of the available resources by advanced modeling tools. Many large-scale ABMs use specialized toolkits or development environments based on accessibility, cross-platform compatibility, ease of understanding, and the need for sophisticated database connection features, graphical user interfaces, and GIS. Widely used toolkits include the most common Repast, MASON, and SWARM, although Crooks and Castle (2012) note that more than 100 toolkits are currently available. These toolkits are also provided by predefined method and feature libraries that can be easily integrated into an ABM and connected to other software libraries.

Using a toolkit will significantly reduce model construction time, allowing more time to be spent on research. However, risks include a significant time commitment for the researcher to learn how to develop and incorporate a model throughout the toolbox and the programming language used by the software. However, it is possible that after this time investment, the desired functionality may not be available.

(c) Agent model services

A range of services is commonly required for implementing large-scale models regardless of the selected design methodology. This could include real data and geospatial environments, which are becoming more prevalent. Some of the more common capabilities include project specification services; agent specification services; input data specification and storage services; model execution services; results storage and analysis services. There are three common approaches, depending on how much support the implementation environment provides for the modeler:

1. The library-oriented approach;
   In a library-oriented approach to project specification, an agent modeling tool consists of a routine library structured into an application programming interface (API). Modelers build models through a series of calls to the various functions of the modeling toolkit. It is the duty of the modelers to ensure that the right call sequences are used and that all the appropriate files are present. In return, modelers have a great deal of versatility in the way they describe their models. Examples include the binary libraries used by Swarm (Terna et al., 1998), the Java archives (JAR) used by Repast for Java (M. J. North and C. M. Macal, 2007) or MASON (Dunham, 2005), and the Microsoft.NET assemblies used by Repast for the Microsoft.NET framework (M. J. North, Collier, et al., 2013).

2. The integrated development environment (IDE) approach;
   The IDE project specification method uses a model editing program or code to coordinate model creation. IDE also offers an integrated framework for compiling or interpreting, and then implementing models.

3. The hybrid approach;
   The hybrid approach enables modelers to use the environment either as a stand-alone library or a multi-file IDE factor. Examples include AnyLogic (Borshchev et al., 2014) and Repast Simphony (M. J. North, Collier, et al., 2013). In return for this added versatility, these environments can require more expertise to be used than other types of IDEs, but they also appear to be the most efficient. Agent specification systems offer a way for modelers to describe the characteristics and behaviors of agents.

3.1.7 Summary ABM

Agent-based modeling is composed of interacting, autonomous agents to model complex systems. Agents are defined by simple rules and have behaviors and relationships with other agents, which might affect their behaviors. The focus of ABM lies in modeling the heterogeneity of agents and a population and the development of self-organization. ABM provides a way to model social networks that are made up of agents that connect, influence each other, and adjust their actions to suit their environment better. Furthermore, ABMs aim is to seek an explanatory insight into the collective behavior of agents following simple rules. They can go from small-scale models to large-scale models or anything extreme in between. ABMS are typically implemented in computer simulations, either via custom software or via ABM toolkits which assist the researcher in developing. However, before selecting an implementation environment, the researcher should answer a series of particular questions that the model should answer to allow for the optimal design of the ABM. Questions include the goal statement of the model and the environment in which the behaviors of the agents are denoted. Furthermore, data sources and validation need to be taken into account before choosing an environment.

3.2 Smart Contracts

To date, Bitcoin is the most popular example of blockchain implementations. Bitcoin uses consensus rules and reward mechanisms to reach consensus in an open-ended peer-to-peer scheme. While discussions are commonly associated with Bitcoin as the most popular example of blockchain technology, the implications of blockchain technology extend well beyond digital currencies (Nakamoto, 2019).

Whereas earlier blockchain implementations focused on distributed state management, the most recent blockchain applications, such as Ethereum Buterin et al. (2014) enable distributed not only state management but also the execution in the form of *smart contracts*.

In a transparent, trustworthy, and verifiable peer-to-peer system, the ability to express logic provides opportunities for organizations and individual users to delegate parts of their operations into a public blockchain. Voting-based decision-making, crowdfunding, management of assets, or management of workflows are examples hereof. However, the open nature of such mechanisms provides opportunities that go beyond the traditional integration of information systems or the coordination of human actors. Smart contracts may be used by artificial entities that participate in contractual commitments on their own or build contracts to provide an open audience with services. However, contractual specifications should be available to all engaging individuals, whether artificial or human, while communicating in an open environment. Careful design and execution do not protect

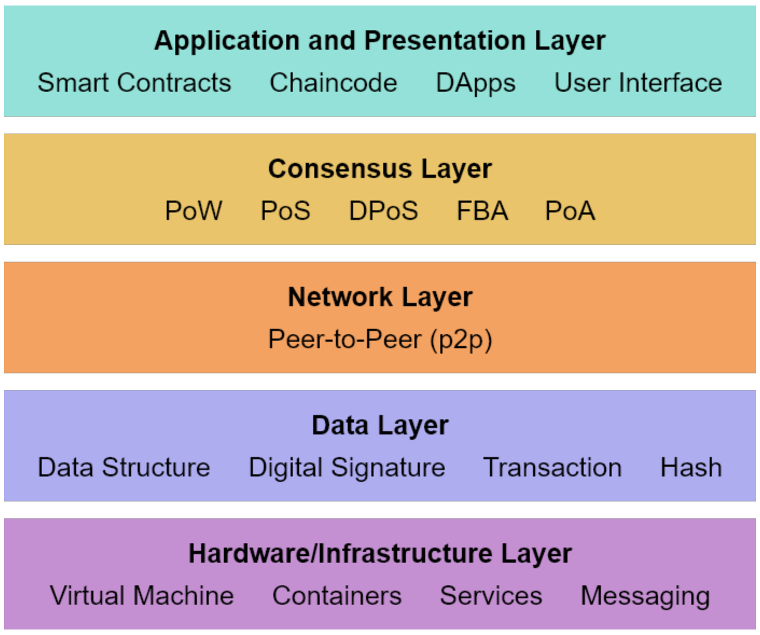


FIGURE 3.4: A traditional blockchain layered architecture

against poorly written or unsafe contracts. An aspect is recently seen in the massive theft of funds from the most popular Decentralized Autonomous Institution of Ethereum (DAO) (Finley, 2016). Therefore, the human-readable contract and associated responsibilities must be codified, as with any programming activity, and then checked to ensure that the machine-readable representation conforms to the stated behavior. Therefore, as in every programming activity, the human-readable contract and related responsibilities must be codified and subsequently verified to ensure that machine-readable implementation complies with the specified behavior.

Figure 3.4 aims to overview a traditional blockchain layered architecture to which smart contracts can be deployed. Some of these layered components need clarification to understand better how smart contracts are built and deployed. In contrast, other components are not relevant for this study. As displayed in Figure 3.4, Smart contracts are deployed on top of the application layer and create an agreement in the consensus layer. Consensus is reached when nodes on the peer-to-peer network layer reach an agreement based on the type of consensus mechanism. The execution of the smart contracts is executed on a specially designed blockchain virtual machine called Ethereum Virtual Machine. Here the code is executed in a distributed fashion across all the nodes that validate the transactions.

As mentioned, a trustworthy, transparent, and verifiable peer-to-peer system has the ability to express logic and provides opportunities to delegate parts of its operations into a public blockchain. Thus it becomes important to understand some consensus mechanisms described in section 3.2.1, as well as peer-to-peer networking, which is highlighted in 3.2.2. The execution of smart contracts through the Ethereum Virtual Machine is detailed in section 3.2.3. Lastly, in section 3.2.4, the means to smart contract development is detailed.

3.2.1 Consensus Layer

How to achieve consensus among the untrustworthy nodes in the blockchain is a transformation of the problem of the Byzantine Generals (BG), as posed in Lamport, Shostak, and Pease (2019). Byzantine Generals refers to a group of commanders who control a portion of the army circled the Byzantine area. While some generals prefer to withdraw, some generals prefer to attack. The attack would, however, fail if only a portion of the generals attacked the city. Thus, to strike or retreat, they have to reach an agreement. Reaching consensus in distributed systems strikes as a problem, just like the blockchain network is distributed. There is no central node in the blockchain that guarantees that ledgers on distributed nodes are the same. Several protocols are required to ensure the integrity of ledgers in various nodes. Farshidi et al. (2020) describes several common approaches to reach consensus in a blockchain in Table 3.1. There are several approaches to reach consensus, as seen in

TABLE 3.1: Consensus Mechanisms of blockchains and their definition depicted by Farshidi et al. (2020)

Blockchain Feature	Description
Proof-Of-Work (PoW)	is a consensus mechanism employed to confirm transactions and generate new blocks to the chain. With PoW, miners compete against each other to perform transactions on the network and gain rewards.
Proof-of-Stake (PoS)	is a consensus algorithm by which a cryptocurrency blockchain network aims to achieve distributed consensus. In PoS-based cryptocurrencies, the next block producer, are chosen based on diverse combinations of random selection, wealth, or age.
Delegated Proof of Stake (DPoS)	is a consensus mechanism for maintaining the final agreement across a blockchain network, validating transactions, and performing as a form of digital democracy.
Federated Byzantine Agreement (FBA)	is a form of Byzantine fault tolerance where each byzantine general is responsible for their blockchain. An FBA has high throughput, scalability, and low transaction costs. Notable cryptocurrencies using the FBA include Stellar and Ripple. Stellar was the first cryptocurrency that implemented a secure FBA
Proof-of-Authority (PoA)	is a replacement for PoW, which can be utilized for private blockchains. It does not depend on nodes solving arbitrarily severe mathematical problems but instead uses a set of "authorities" - nodes explicitly permitted to generate new blocks and secure the blockchain.

Table 3.1. However, only a few spark interest in this study. In this research, a deeper look into the Proof-of-Work (PoW) consensus mechanism is described as it is used by Ethereum, which is a good candidate for deploying smart contracts due to its popularity.

A consensus technique used for the Bitcoin network is Proof-of-Work (Nakamoto, 2019). One has to be appointed to record transactions in a decentralized network. Random selection is the simplest process. Random selection, though, is vulnerable to attacks. So, if nodes want to publish a transaction block, much work needs to be done to show that the network is not likely to be targeted by the node. The role normally includes machine calculations. In PoW, the hash value of a block header is determined for each node of the network.

A nonce is contained in the block header, and to get various hash values, miners will regularly change the nonce. Consensus demands that the value measured must be equal to, or less than, a certain value. As one node achieves the target value, the block will be broadcast to other nodes, and all other nodes must mutually check the validity of the hash value. Other miners will add this new block towards their own blockchains when the block is validated. Nodes that measure hashing values are depicted as miners, and in Bitcoin, the PoW process is called mining. Once multiple nodes find a suitable nonce almost simultaneously in a decentralized network, valid blocks could be generated simultaneously. As a consequence, branches can be generated. It is unlikely, however, that competing forks would simultaneously produce the next block.

In PoW, miners do many machine calculations, but these operations waste so much money. Some PoW protocols wherein operations have been built to minimize the loss of computational power. For instance, Primecoin looks for unique prime number chains that could be used for mathematical analysis (King, 2013).

To Summarize: Consensus

Different consensus algorithms have different advantages and disadvantages. Table 3.1 gives a comparison between different consensus algorithms. We denote that the consensus mechanism for PoW is based on computational work, whereas PoS consensus is based on ownership of the currency. PoA consensus is based on currency discovery, whereas FBA is based on majority voting systems.

A successful consensus algorithm implies performance, security, and comfort. A variety of attempts have recently been made to develop blockchain consensus algorithms. Novel consensus algorithms are being designed to solve some unique blockchain problems. For example, PeerCensus's idea is to decouple transaction confirmation and block formation so that the speed of consensus can be dramatically increased (Decker, Seidel, and Wattenhofer, 2016). A novel consensus approach was also proposed by Kraft (2016) to guarantee a block is produced at a reasonably stable pace. It is understood that high rates of generation blocks weaken the security of Bitcoin.

3.2.2 Peer-to-Peer Network Layer

Peer-to-peer is a network in which nodes establish an autonomous network in which data is requested and given on an equal footing between these nodes. A node is a physical/virtual machine that communicates with other nodes via TCP/IP and UDP (Prieto et al., 2017). It is in opposition to the classic client-server system in which one party is responsible for providing and storing the data. This leading party is a centralized server, and clients ask for and receive their data from this centralized server.

Any elements of a network become much more complicated by not providing a client/server architecture. For example, it is important to consider how data is transmitted and the transmission of data between peers (Fox, 2001). The network is open for all nodes for a blockchain to run properly and that replication functions proficiently. Peer-to-peer communication techniques are used to form a distributed network, removing a single point of failure (Levy and Silberschatz, 1990). Thus it plays a crucial role in verifying and generating blocks that are appended to the blockchain (Zheng et al., 2017).

Schollmeier (2001) refers to a peer-to-peer network as a distributed network architecture if users share part of their own hardware resources. These mutual resources are required to provide the Infrastructure and the content provided by the network. They are immediately available to other peers without going through intermediary bodies. The members in such a network are also both resource users and resource providers.

To Summarize: P2P

The overall cost of a P2P network is relatively inexpensive as there is no central configuration. Furthermore, the network is not dependent on a centralized system which means that connected nodes can function independently while reducing bottlenecks. P2P offers great scalability while the performance of the network remains the same.

3.2.3 Ethereum Virtual Machine

On a specially built blockchain virtual machine, named Ethereum Virtual Machine (EVM), smart contracts are executed (Buterin et al., 2014). The EVM is a virtual machine built on a stack that functions in the byte code language. Ethereum nodes all share the same EVM specification, and nodes that validate the transactions execute the code in a distributed fashion. Program execution is often bound by the up-front supply of the consumable energy, a unit of execution cost incurred by each machine code command, and by each byte of storage used by the contract, in time and space. The fees underlie the same concept as in Bitcoin but instead rely on the ether denomination.

Although the byte code language of Ethereum is designed for efficient delivery and execution, several high-level programming languages have evolved to simplify contract development, such as Serpent, Solidity, and LLL. With the de facto norm for contract production being Solidity (Delmolino et al., 2016). Python, JavaScript, and C++ influence the syntax of Solidity, but it incorporates inheritance, statically typing libraries, and other beneficial features.

Furthermore, the EVM facilitates events for callback notification or execution. Notice that although the EVM needs code execution on all the connected nodes, no type of parallel computation as part of the code is supported. Solidity recognizes structs as well as enumerations, in addition to functions. Within the contract code, you can access the invoking message properties, like sender address, function execution funds, numerous cryptographic controls, as well as a predefined contract destruction self-destruct() function. It is worth noting that the object in Ethereum itself can only accomplish the termination of the life of an entity. *Therefore, the public existence of the blockchain makes careful creation and management of the life cycle important.*

New contracts are compiled to EVM byte code to instantiate. Users must receive an appropriate amount of ether to finance the execution of a contract based on the expected complexity of the code submitted. Transactions will then trigger features at the address of the contract. Ethereum handles two types of transactions for this purpose, which consist of the following essential attributes:

- Ether amount;
- Receiving address;
- Byte array containing the payload;
- The sender's private key signature;

When a receiving address is missing, a new contract added to the network must be the transaction payload. The EVM will determine the validity of the transaction requests dependent on the sender's signature before executing a transaction and will validate the added ether required to sponsor the execution of the EVM code.

To Summarize: EVM

The Ethereum Virtual Machine is an efficient sandbox virtual stack responsible for contract byte code execution, embedded within each complete Ethereum node. Contracts, such as Solidity, are commonly described in higher-level languages, then compiled into EVM byte code.

This means that machine code is totally disconnected from the host computer's network, file system, or any operation. Each node runs an EVM instance on the Ethereum network, enabling them to agree to execute the same instructions.

In the Ethereum Protocol, the EVM is central and is instrumental in the consensus engine of the Ethereum network. It enables everyone to execute code in an unreliable environment in which it is possible to guarantee the outcome of execution and is completely deterministic (i.e.) to execute smart contracts.

3.2.4 Smart Contract Design with Solidity

Contracts represent classes in object-oriented programming languages and can contain typed state variables. Furthermore, contracts can include functions that are invoked externally. To carry out preliminary checks (e.g., for validation) in a declarative way, function modifiers may also be added to one or more functions, thus representing the features of aspect-oriented programming (Kiczales et al., 1997).

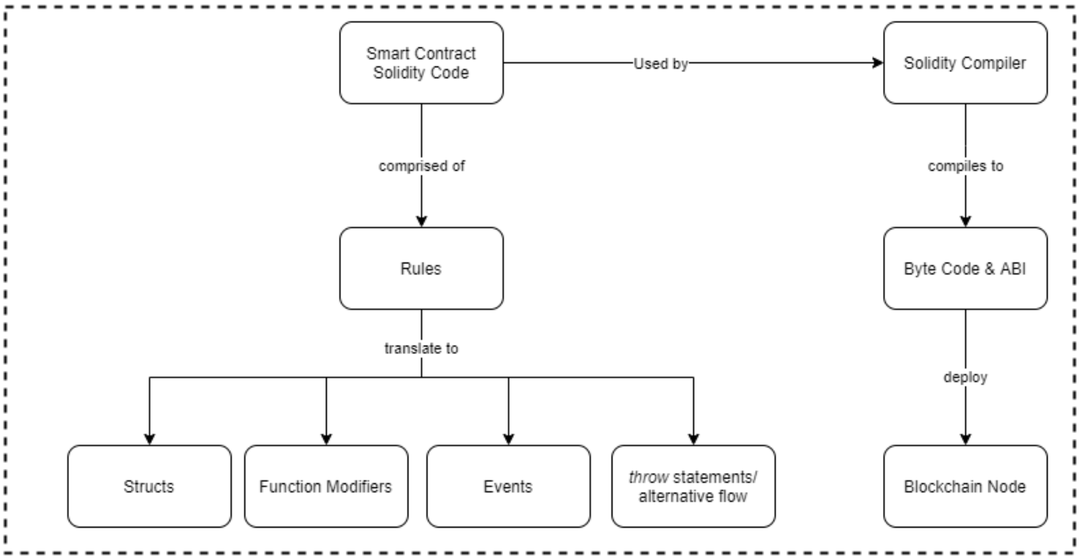


FIGURE 3.5: Visualizing the workings of a smart contract through a concept diagram, derived from Hall and Virrantaus (2016)

Figure 3.5 displays the working of a smart contract as a concept diagram. A Smart Contract comprises rules, which could translate to structs, function modifiers, events, or alternative flows. The smart contract is compiled with a Solidity compiler which compiles to Byte Code. This byte code can, in its turn, be deployed on a blockchain node.

The use of high-level programming languages makes smart contracts fairly easy to establish and encode. However, possible ***use of smart contracts as intermediaries in open platforms makes them a vital asset***. If blockchain technologies and smart contracts were to be implemented to provide critical public services, such as land ownership records, this might refer to the operation of a single company, a government, and even countries. The consequences can be far-ranging if contracts are poorly written or exploitable. ***In addition, few methods are available at the current stage that endorses the systemic modeling of contracts to resemble real-world entities.***

If source code is available, confidence in a contract can be better developed. The Solidity compiler promotes the use of SPDX license identifiers that are machine-readable because having source code available often affects legal issues concerning copyright. In addition, Pragmas makes the critical design of a Solidity source file for function tests, whereas versions are used to reject conflicting changes. The example Source Code 3.1 gives an overview of the layout to which a Solidity source file should respect.

SOURCE CODE 3.1: Layout of a Solidity Source File

```
1  // SPDX-License-Identifier: GPL-3.0
2  pragma solidity >=0.4.21 <0.9.0;
3
4  /** @title Shape calculator. */
5  contract ShapeCalculator {
6      /// @dev Calculates a rectangle's surface and perimeter.
7      /// @param w Width of the rectangle.
8      /// @param h Height of the rectangle.
9      /// @return s The calculated surface.
10     /// @return p The calculated perimeter.
11     function rectangle(uint w, uint h) public pure returns (uint s, uint p)
        ↪ {
12         s = w * h;
13         p = 2 * (w + h);
14     }
15 }
```

As seen in Source Code 3.1, every source file should begin with a comment indicating its license. Thereafter, the SPDX license identifier on line two is followed. As seen in line two, a pragma keyword is used to make certain compiler features or tests. The Pragma Directive applies to the source file only. It is important to apply the Pragma Directive to all files to allow it for the whole project. If you import another file, the importing file does not immediately add a pragma from that file. Source files should be annotated with a version pragma to deny compatibility with forthcoming compiler versions that could incorporate incompatible modifications. Only because of that, reading at least for changes that need breaking modifications through the changelog is always a good idea. These releases also have a version also seen on line two in the form of 0.x.0 or x.0.0.

The Solidity contracts structure is comparable to classes in object-oriented programming languages. Function Modifiers, State Variables, Events, Struct Forms, and Enum Types declarations may be included in each contract. In addition, contracts can be inherited from other contracts. Special types of contracts, namely libraries and interfaces, also exist. Source Code A.1 in Appendix A gives an overview of these declarations.

Contracts can be generated via Ethereum transactions from outside or through Solidity contracts from within. When a contract is produced, it is executed once by its builder. The final code of a contract is saved on the blockchain after the constructor has executed it. This code contains both public and external functions that function calls from there can access. If a contract wishes to produce another contract, the creator must be aware of the contract's source code.

3.2.5 Summary Smart Contracts

In a transparent, trustworthy, and verifiable peer-to-peer system, the ability to express logic provides opportunities for organizations and individual users to delegate parts of their operations into a public blockchain. Smart contracts may be used by artificial entities that participate in contractual commitments on their own or build contracts to provide an open audience with services. Careful design and execution do not protect against poorly written or unsafe contracts. An aspect is recently seen in the massive theft of funds from the most popular Decentralized Autonomous Institution of Ethereum (DAO) (Finley, 2016). Consensus on a blockchain is reached when nodes on the peer-to-peer network layer reach an agreement based on the type of consensus mechanism. The execution of the smart contracts is executed on a specially designed blockchain virtual machine called Ethereum Virtual Machine. Here the code is executed in a distributed fashion across all the nodes that validate the transactions. A successful consensus algorithm implies performance, security, and comfort. The use of smart contracts as intermediaries in open platforms makes them a vital asset. In addition, few methods are available at the current stage that endorses the systemic modeling of contracts to resemble real-world entities.

It can be denoted that smart contracts offer many advantages to automate or facilitate as a decentralized entity. Besides automation, also transparency, joint ownership, and immutability are potential improvements smart contracts bring. However, it is important to be aware of significant drawbacks. Smart contracts are highly technical contracts, meaning that faulty code should be a big concern. Owning to the *immutability* of blockchain technology, a bug or security concern in a deployed smart contract cannot be removed. At the same time, they might process high costs or be resource-intensive. Furthermore, implications *scalability* brings, such as increased interaction with the smart contract, might impact the system, which cannot always be easily predicted due to the complexity of the infrastructure.

We hope to find valuable findings through this study by translating smart contracts to agents in an agent-based model. This agent-based model could potentially evaluate factors like scalability, faulty or improvable code of the smart contract. The number of agents carrying such smart contracts can be modified and simulated easily.

3.3 Modeling Smart Contracts with Institutional Grammar

Crawford and Ostrom present an agent-based approach to the modeling of smart contracts with institutional grammar in the paper "From Institutions to Code: Towards Automated Smart Contract Generation." (Crawford and Ostrom, 1995). It is based on Helbing (2012)'s definition of Grammar of Institutions, which lies in the field of institutional study. For decomposing institutions into rule-based statements, this grammar of institutions is used. In a systematic formalization, these statements are compiled below.

In agent-based modeling, the grammar of institutions is rooted in (ABM). ABM is a paradigm of computational modeling in which phenomena are modeled as interacting agent stochastic systems (Helbing, 2012). The model consists of a collection of agents that encompass the behaviors of the different people that make up the system, and the execution emulates these behaviors (Parunak, Savit, and Riolo, 1998). Sometimes, agents' behavior is modeled by collections of sentences in which the behavior becomes explicit. The statements are constructed from five components, in this case, abbreviated jointly to ADICO:

- **A**ttribute: The actor's characteristics or attributes;
- **D**eontic: the nature of the statement as an obligation, permission or prohibition;
- A**i**m: the action or outcome that the statement regulates;
- **C**onditions: The contextual conditions under which the statement holds;
- **O**r else: Describing the consequences associated with non-conformance to the statement

Frank Dignum and Conte (1997) specifies that the use of deontic reasoning provides the ability to specifically decide what should occur in cases of infringement of obligations in these cases. Deontic logic could be used to model the principles that agents communicate with each other according to. Deontic logic provides the ability to define specifically the principles that can be used to enforce agent-to-agent interactions. Dignum, Meyer, and R. Wieringa (1994) reasons about sub-ideal states (in which an obligation is violated). In the description of this method, an example of a voting system is given using these components:

```
People (A) must (D) vote (I) every four years (C), or else they face a fine
(O).
```

We can define the important domain-specific constructs by exploring its structural elements between contracts and institutional grammar and suggest a mapping that simplifies contract generation. For instance, the conceptual equivalent of the attributes portion of ADICO is the structure of Solidity. In analogy, objectives essentially describe functions and events, while the combination of Deontic and the part of the corresponding conditions are reflected. In function modifiers, which implement declarative checks preempting the execution of functions. Table 3.2 illustrates the ADICO-Solidity mapping proposed.

TABLE 3.2: Mapping of ADICO components to Solidity declarations

ADICO component	SOLIDITY construct
Attribute	Structs
Deontic	Function modifiers
Aim	Functions, Events
Conditions	Function modifiers
Or Else	*throw* statements/alternative control flow

This mapping of core constructs gives the basis for converting institutional specifications to solidity contracts. ADICO's component structure, however, can be optimized in addition to high mapping. A transparent collection of rules attached to the component attributes (i.e., the actor properties) is added to translate the elements of the Solidity struct. Aims (representing the behavior and institutional statement controls) are refined by allowing the object and aim associated with a given action to be defined, all of which correspond to the Solidity function parameters. A special case is activated based on encoded conditions being met (e.g., reaching a deadline). A further refinement requires the possible annotation of the property data types of attributes to enhance code generation. Operationalization of ADICO is possible by defining attributes specifically and by further allowing objects associated with actions to be specified as well as possible targets of action. In addition, in theory, several conditions may be merged into a single argument. Using logical operators such as AND, OR, and XOR, several ADICO statements can be combined.

Example: Voting with Result Notification To further clarify how the ADICO mapping could work, the syntax for ADICO-Solidity is shown, and an example is given and reflected in Figure 3.6. First, a collection of statements is made representing simple rules that permit registered voters to cast a vote. The individual statements decompose to individual permissible actions and constraints. Take the following example: *Voters need to be registered and may only vote once but may only vote before a given deadline.*

1. Statement 1: Voters need to be registered,
2. Statement 2: may only vote once,
3. Statement 3: But may only vote before a given deadline.

The above statements can be translated into the following ADICO mapping:

SOURCE CODE 3.2: ADICO Mapping based on three statements

```
Adico(
    A("Voters"),
    D(may),
    I("cast", object("vote", "string"), target("candidate")),
    C(IF("voter", "is" , "registered") AND
      IF("vote", "before", "deadline"))) AND
Adico(
    A("Voters"),
    D(may),
    I("cast", object("vote", "string"), target("candidate")),
    C("only", "once")) AND
Adico(
    A("System"),
    D(must),
    I("notify", object("vote count"), target("contract",
        "address")),
    C(IF("votes.length", Operator.>, "100")))
```

The mapping as seen in Source Code 3.2 generates the corresponding contract illustrated in Figure 3.6, including both the event generated and the associated function.

Figure 3.6 illustrates the corresponding contract generated, including both the event generated and the associated function. The classification of events could be of limited accuracy, similarly to other constructs, such as omitting type parameters. To represent the necessary semantics, the contract approach involves manual refinement. At this point, the contract skeleton created reflects an interface that captures the intent of the contract at an abstraction level specified by institutional statements without revealing details of execution.

3.3.1 Bridging the gap between smart contracts and agent-based modeling

ADICO itself is not a framework nor an automated method to writing either smart contracts or architecture for defining agents and their rules for an agent-based model.

```
   // SPDX-License-Identifier: GPL-3.0
   pragma solidity >=0.4.21 <0.9.0;

   contract AdvancedVotingSystem {
       address voter;
       struct Voters{
 A         address voter;
       }
       event notifyVoteCount(*Define Type* voteCount, address contract);
       function AdvancedVotingSystem() {
           owner = msg.sender;
       }
 C     modifier voteLength$greater100() {
 D         if(! votes.length > 100)
 O             throw;
       }
 C     modifier voterIsRegistered(){
 D         if(! voterIsRegistered)
 O             throw;
       }
 C     modifier voteBeforeDeadline(){
 D         if(! VoteBeforeDeadline)
 O             throw;
       }
 C     modifier onlyOnce(){
 D         if (! onlyOnce)
 O             throw;
       }
       function notify(*Define type* voteCount, address contract) voteCount$greater100 {
       notifyVoteCount(voteCount, contract);
       }
 I     function cast(String vote, *Define type* candidate) voterIsRegistered voteBeforeDeadline onlyOnce {
           // TODO: Implement code to cast vote for target candidate
       }
       // Capture malformed payload
       function() {
           throw;
       }
       //Destroys contract, but only if called by original creator of contract
       function kill(){
           if (msg.sender == owner)
               selfdestruct(owner);
       }
   }
```

FIGURE 3.6: Contract generated based on ADICO statements.

However, it could bridge the gap between designing agents for an agent-based model and designing smart contracts starting from a collection of ADICO statements that correspond to the rules of both these entities. This approach is proven to have potential, which can be seen in the paper "from Institutions to Code: Towards Automated Smart Contract Generation." (Crawford and Ostrom, 1995). Also Smajgl, Izquierdo, and Huigen (2008) propose a sequence where one could model endogenous institutional rule change for agents. This is done by identifying the structural components of a general rule from a modeling perspective, followed by providing an agent architecture overview. Parts of the proposed sequence are implemented in a NetLogo model.

The first sub-question of this research asks how smart contracts and agent-based modeling be defined based on prior literature. We find that both concepts heavily depend on a set of rules translated from functional requirements through literature review.

An agent-based model is a computerized simulation of many decision-makers (agents) and institutions that interact through prescribed rules. Figure 3.7 visualizes how the different domain concepts relate to each other.

Figure B.2, in Appendix B, illustrates how a smart contract definition influences the simulation outcome based on the rule-based statements defined through a case study. Firstly, the case study criteria are defined, which results in a goal statement, a collection of rules, and a utility threshold the infrastructure should answer to. Secondly, These statements from the case study are used as criteria for the agent-based model's agents, relationships, and environment. These statements could be translated to ADICO statements translated in smart contracts and agent definitions in the ABM. Thirdly, the ABM tool selection procedure is executed corresponding to the case study requirements at hand. Finally, the simulation is performed, an evaluation is done through a utility function with summarizes. It exports the results of the data to which analysis can be done, such as statistical analysis. Therefrom, a conclusion can be formed.

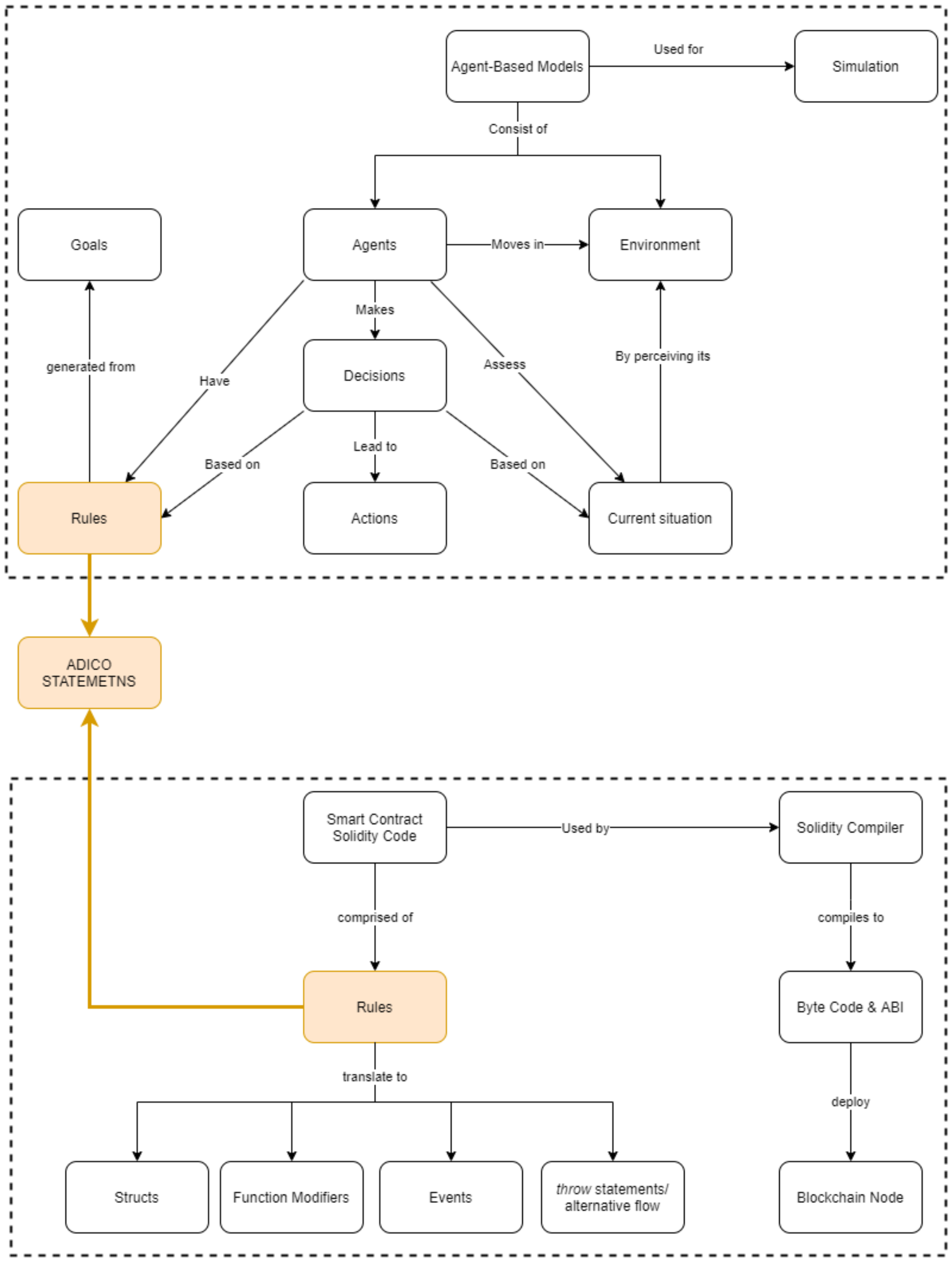


FIGURE 3.7: Commonalities found between smart contracts and agent-based modeling.

Chapter 4

ABM Decision Support

This chapter discusses the ABM Tool Selection process as described in B in more detail. The table4.1 provides an overview of agent-based modeling characteristics used to evaluate various tooling and simulation programs currently available.

Abar et al., 2017 introduces the state-of-the-art Agent-Based Modeling and Simulation techniques with a detailed comparative analysis. A detailed characterization of almost the entire range of agent-based modeling and simulation tools is described in the paper, outlining the salient features, merits, and limitations of such application software. Consequently, this paper uses a valuable guide to enable engineers, researchers, and learners to easily pick an acceptable agent-based modeling and simulation toolkit for the requirements addressed in this paper to design and develop system models and prototypes.

The computational algorithms, mathematical expressions, and equations that encapsulate a system's action and output in real-world scenarios are referred to in a simulation model. There is a minimal set of basic characteristics that describe a software agent. A software agent is autonomous, can function as a standalone mechanism, and performs actions without user intervention (Ballard, 2011).

ABMS 'theory is to model complex systems using a bottom-up approach to simulate practical scenarios with a group of self-governing agents, starting with the individual agents (Moon, 2017). Each approach marks the agents with specific programming syntax and semantics and has a different basis concerning generality, usability, modifiability, scalability, and performance. Based on the aforementioned features Abar et al. (2017) mapped the following ABMS based on ease of model development effort and computational modeling strength/scalability level.

The study of Abar et al. (2017) provides a great foundation of state-of-the-art agent-based modeling and simulation tools. At the time of this writing, the aforementioned synthesis of ABMS might be outdated. Therefore, we will review and update the resources and provide a few extra ABM tool characteristics gaining traction. These characteristics are detailed in Section 4.1. The full list of ABM tools is enclosed in Appendix C.

4.1 ABM Tool Characteristics

The main objective is to define almost the entire range of various agent-based modeling and simulation programs currently available and position them in a repertoire, effectively refining these multifaceted application software's noteworthy features, merits, and demerits. Overall, the significant features, merits, shortcomings, and limitations of the agent-based modeling and simulation software tools surveyed are considered in this analysis. Evaluation criteria are described in Table 4.1 and include:

TABLE 4.1: Definitions of agent-based modeling and simulation characteristics used as criteria for the agent-based modeling domain.

ABM Characteristic	Definition
Development Effort	*The availability of approaches to reduce LOC to achieve the desired outcome.*
Scalability	*The computational modeling strength or scalability of the models developed through the use of particular toolkits.*
Type of License	*The source-code specification, online availability, distribution license/legibility as an open or closed source.*
Type of Agents	*The implementation types of agents are primarily based on the interaction mechanism during simulation activity.*
Programming Language	*The requirements and provision of an Application Programming Interface (API) together with the availability of built-in libraries for incorporation within the user's source code for developing agent-based models.*
Application Domain	*The assessment of coverage of application areas/domains covered by the ABMS tool*
Source Code	*The identification of the compiler, operating system support, and platform/hardware requirements/constraints for the model implementation.*
GIS capabilities	*A geographic information system (GIS) is a conceptualized structure that enables spatial and geographic data to be collected and analyzed.*
3D Capabilities	*3D modeling is the method of constructing a three-dimensional mathematical representation of the surface of an entity.*

4.1.1 Scalability

Scalability is called "the ability to handle increased workload" (Lorig et al., 2015). From distributed systems, getting better scalability has become an important challenge for developers for years (Neuman, 1994). While performance is described as how quickly and effectively a software can perform a specific task, scalability describes changes in performance when increasing system load.

A system is deemed 'not scalable' in the event of a dramatic output decline resulting from increased load (Liu, 2011). Korba and Song (2002) defined load and complexity as two key factors affecting the scalability of agent-based systems. Load is extracted from the number of CPU and Memory threads used by the multiagent framework. Complexity, by contrast, defines the computation time of an agent-based system(Korba and Song, 2002).

Scalability can be considered vertically and horizontally through two distinct viewpoints, often referred to as the scale-up and scale-out. The deployment with one large server providing several core processors and sufficient RAM is defined by vertical scalability. In comparison, horizontal scalability incorporates a range of smaller servers into one device to spread the load of a system (Michael et al., 2007).

The scalability of an ABM is dependent on validating population size based on time. The level of scalability depends on incorporating distributed computing frameworks, the ability to map cluster and supercomputer architectures, and deployment complexity. Furthermore, using data-parallel algorithms on GPUs and the number of multi-processor GPUs plays an important factor. To summarize, ABM-Tools that incorporate features such as distributed computing frameworks, cluster and supercomputer architectures, light deployment complexity, data-parallel algorithms on GPUs, and multi-processor GPUs increase the level of scalability.

4.1.2 Development Effort

Evaluation of the development effort regarded the size of simulation models to measure the effort necessary to generate them. We used code lines (LoC) more explicitly, which are also used as a metric for the software scale (Boehm et al., 1995; Jensen, 1983; Putnam and Myers, 1991). In fact, LoC is a metric used in MDD approaches to assess and evaluate development and design implementation efforts (Osis and Asnina, 2010).

There are several approaches to reduce LOC to achieve the desired outcome. A *graphical programming interface* is available for code editing and simulation running or an *intuitive visual programming user interface* for the model rendering tasks. When using MDD, *transformation engines and code generators* decrease or suppress the development effort. Model-to-code changes are the foundation of source code engines that, in turn, simplify the creation of low-level software objects (e.g., lines of code). *General-purpose languages*, such as Java, may be used to simulate these modules, but constructing such abstractions from scratch will require additional effort and programming skills. ABMS is expanding its application domain to a wider community of users by reducing the amount of programming knowledge needed(Mernik, Heering, and Sloane, 2005).

4.2 ABM Tool Selection

We defined a big set of various agent-based modeling and simulation programs currently available and represented in Table C.1 in Appendix C. The evaluation criteria are described in Table 4.1. The meaning of scalability and development effort is detailed in section 4.1.1 and 4.1.2 subsequently. To allow the ABM community to benefit from this research, a repository is created on Github. Furthermore, as seen in Figure C.1 in Appendix C shows a website that has been deployed to allow for easy searching based on features.

The website provides a table with over 80 records. It is possible to filter data for each column to refine your search based on your interests. As a proof of concept, this website will create a shortlist of possible ABM-Tools for this study.

In case the requirements state that we would like to simulate a **high population number**, then scalability is an important factor to consider. Furthermore, if one likes to simulate a **complex system** as there are multiple different types of agents with different types of behavior, then complexity becomes an important factor.

For this study Repast, Simphony is chosen as the respective toolkit as the scope or application domain involves The latter two traits are significant application domains that lay in line with this study. Moreover. Repast Simphony uses Java Runtime Environment with Eclipse platform-based user interfaces, which I am more familiar with.

Though similar simulations can be done in various tools, the decision was made to perform the simulation in Repast Simphony for research beyond this study. This simulation is performed for the SearchSECO project. SearchSECO is a project that tries to maintain a hashed index of software methods of the worldwide software ecosystem. The simulations done during this study will serve as a guide for SearchSECO. However, more complex contexts and behaviors will be added in the future to gain a deeper understanding of the SearchSECO data. Hence, by choosing Repast Simphony, we allow the simulations to scale and become more complex. Moreover, SearchSECO in the following chapter.

Chapter 5

SearchSECO Case Study

5.1 Introduction to SearchSECO

In this chapter, we address the case study SearchSECO. SearchSECO is a project that tries to maintain a hashed index of software methods of the worldwide software ecosystem (Jansen et al., 2020).

Repository mining research is a data-intensive domain with a focus on source code. There are many ways to search for code in the worldwide software ecosystem, but these search methods are inefficient and only cover small parts of the software ecosystem. One of the problems is granularity: it is possible to search through code on a file level and cover a significant part of the software ecosystem or search for a line of code and only cover a small part of the software ecosystem, but not both.

SearchSECO proposes a language-agnostic search engine and research platform that searches through abstract representations of source code methods. SearchSECO is used to search across the worldwide software ecosystem and index the encountered methods. With SearchSECO, the field is advanced because it (1) provides finer-grained and more efficient searches, (2) covers more of the software ecosystem than other search mechanisms, and (3) provides mechanisms for source code provenance.

The main challenge of SearchSECO is to make it sustainable. With the billions of code lines currently made available in public repositories, it is a monumental task to download, parse, and store the data needed to make SearchSECO successful.

SearchSECO assumes the following actors in the SearchSECO community.

1. **SearchSECO Project Team -** The SearchSECO project team is made up of multiple companies and universities that collaboratively try to establish the SearchSECO platform. Currently, the team is carrying all the costs of the platform.

2. **Empirical software engineers -** many researchers wish to use the platform for their research. Empirical software engineers will occasionally use the platform for their research.

3. **Software producing organizations -** The organizations that produce software will want to use SearchSECO to establish that their software is theirs, to check for copyright violations in their own code, and to co-evolve their systems with other systems.

SearchSECO must create a model in which they reach equilibrium, i.e., where the whole community carries the major costs of the SearchSECO platform.

There are three main costs in this proposal: storage of data fragments (terabytes), parsing of these fragments (CPU), and spidering and downloading the fragments (network). SearchSECO assumes that these costs are constant. Currently, they envision the platform as a distributed system, in which the tasks modeled in Figure 5.1 are distributed over the different parties.

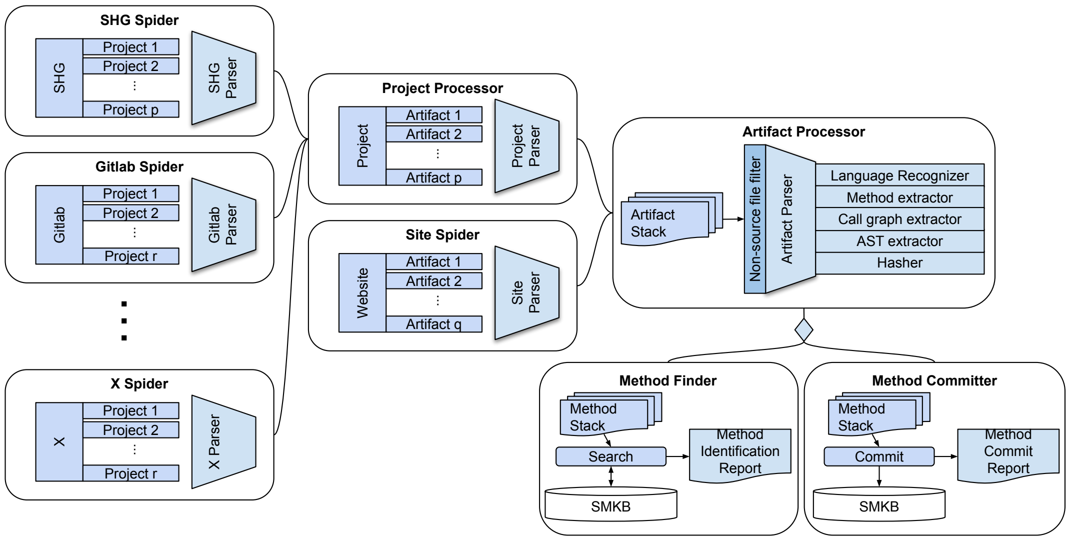


FIGURE 5.1: *The extraction process follows a three-step process. First, projects are identified on different project forums, such as the Software Heritage Graph, Gitlab, and other project repositories. Different artifacts are extracted from web sites such as stackoverflow.com. Secondly, these projects are parsed into artifacts from which fragments are extracted, including their out-calls to other methods. Thirdly, this information is stored in the SMKB.*

TABLE 5.1: Assumed constants in the simulation.

Item	**Value**	**Unit**
Costs of Storage	0.20	euro per GB per month
Costs of CPU	1.00	euro per 10000 data items collected
Number of empirical software engineers	50,000	people
Number of SPOs	200,000	companies
Number of data items accessed by SPOs	10,000	Code fragments per month
Number of data items accessed by ESERs	100,000	Code fragments per year
Initial funding	50,000	euro
Speed per client	15,000	GitHub calls per hour

5.2 Behaviors of the Agents

The project is initiated by the SearchSECO project team, who initially carry the brunt of the work. As the platform grows, SearchSECO becomes more interesting to empirical software engineers in the community, to the point where around 50% of all active empirical software engineers have used the platform. Software producing organizations expect to mostly explore whether their own code is legal and will therefore perform far fewer reads from the system. However, for these software-producing organizations, the reads are very valuable.

5.3 Scenarios

SearchSECO has the following scenarios available:

1. **Force empirical software engineers to Perform Part of the Work -** A demand-based approach is taken. When an empirical software engineer wishes to read an item from the database, she also needs to add an item.

2. **Force empirical software engineers to Store the Data in a Distributed Manner -** It is asked that empirical software engineers run a server with 10-20GB

to ensure that the data is collected and stored redundantly. However, as empirical software engineers are "flaky", they might not provide the persistence necessary.

3. **Ask software producing organizations to Pay for Reading Access -** To make the infrastructure sustainable, SearchSECO can ask software producing organizations to pay for reading access to the database.

4. **Ask software producing organizations to Provide Data Stores -** SearchSECO provides read access for free if the software producing organizations provide data stores in which we can store the data.

5.4 Problem and Object Formulation

To better understand the scenario's requirements, a comprehensive list of questions is used as a guideline to translate some statements. These statements will aid in the development of the agents' criteria and rules to which the corresponding contracts should adhere. Furthermore, the questionnaire is grouped into sections to create an overview of the related goal these questions aim to solve.

Identifying agents, accurately specifying their behaviors, and appropriately representing agent interactions are key to developing useful agent models. One begins developing an agent-based model by identifying the agent types (classes) along with their attributes. Agents are generally the decision-makers in a system, whether they be human, organizational, or automated. Next, the behaviors of the agents are depicted, followed by the relationships between the agents. Finally, the agents' environment is discussed.

Topic: Model Purpose

1. **What specific problem is the model being developed to address?**
   SearchSECO is a complex system as multiple user types (with different behaviors) interact with the system. Different types of users will use SearchSECO differently for different purposes, and the behavior of these users is therefore different. The load on the system and the number of transactions cannot be forecasted with traditional statistical analysis to understand the usage.

2. **What specific questions should the model answer?**
   How can SearchSECO become sustainable? Can we create a model where the community carries the major costs of the SearchSECO platform?

3. **What kind of information should the model provide to help make or support a decision?**
   Reward per action of:

   - *Storage of data fragments*
   - *parsing of data fragments*
   - *spidering and downloading data fragments*

   Costs for accessing:

   - *code fragments per month as a software producing organization*
   - *code fragments per year as an empirical software engineer*

4. **Why might agent-based modeling be a desirable approach?**
   With agent-based modeling, we want to evaluate metrics such as usage, workload, and transactions on the system in case the incentive mechanism on which the platform relies changes. For example, if the amount of GAS or Ether changes, this might impact the number of users

interacting with the system. These rules rely on the behavior of the users interacting with the system, hence why agent-based modeling is an ideal fit to evaluate such changes beforehand to optimize the utilization of the system.

5. **What value-added does agent-based modeling bring to the problem that other modeling approaches cannot bring?**
As the system matures, we will collect more user behavior. Hence once we mapped the basic behavior of our user base, we can further optimize our system. We can easily monitor the impact on the system when certain costs for the system would change based on the behavior of our users. This could help us in decision-making processes to avoid a decrease of utility on our system.

Topic: Agents

6. **Who should be the agents in the model?**

   - *SearchSECO Project Team - The SearchSECO project team consists of companies and universities which are the founders who collaboratively aim to establish the platform. This team carries the setup costs of the platform.*
   - *empirical software engineers - Researchers who wish to use the platform for their research. These researchers will occasionally use the platform for their research.*
   - *software producing organizations - these organizations will use the platform to establish that their software is theirs, check for copyright violations in their own code, and co-evolve their systems with other systems.*

7. **Who are the decision-makers in the system?**
The SearchSECO platform aims to be a self-sustaining platform with no central authority, hence no central decision-makers. The platform aims to be open. Hence the community will further develop the platform, and decisions might be made through decision protocols. Examples of such protocols can be Proof-Of-Work.

8. **What are the entities that have behaviors?**
Both the empirical software engineer and the software producing organization will have monitored behaviors, as they are the main stakeholders for creating a self-sustainable platform. The SearchSECO project team is not modeled with behavior as the predicted involvement of these agents will descale once the system grows. Furthermore, it is expected that the behavior of the SearchSECO Project Team has a low impact on the sustainability of the platform.

9. **Where might the data come from, especially for agent behaviors?**
The initial model will be modeled through toy data as no historical data for this specific platform is available. As user interaction grows, more data is collected through the SearchSECO infrastructure, which iteratively optimizes empirical software engineers and software-producing organizations' behavioral rulesets.

Topic: Agent Data

10. **What data on agents is simply descriptive (static attributes)?**
neither exhaustive nor exclusive

11. **What agent attributes are calculated endogenously by the model and updated for the agents (dynamic attributes)?**
Time spent seeding data, number of commits, number of fixes, volume seeded (GB), volume downloaded (GB)

12. **What is the agents' environment?**
SearchSECO Platform - The SearchSECO platform itself will be an autonomous platform envisions as a distributed system that holds its own rule sets upon which it can interact. This allows for automation of the system.

13. **How do the agents interact with the environment?**
The agents in the model are spatially implicit as their location is not important.

14. **Is agent mobility in space an important consideration?**
Yes, we deal with phenomena in space and time. More concretely, the system aims to be decentralized, which will deploy smart contracts on Ethereum. Ethereum is fluctuating; it often depends on factors such as the valuation of Bitcoin, which is valued based on the cost of electricity as these are the basic costs needed to perform heavy calculations with machines. Furthermore, factors such as expert forecasts and political factors influence the crypto market.

Topic: Agent Behaviors

15. **How would we represent the agent behaviors?**
By If-Else Statements. More concretely by ADICO statements which can be translated to agent definitions as well as smart contracts.

To summarize, we identified a set of agents, their behaviors, and their interactions which are specified based on natural text. These descriptions are vital for developing a useful agent model. Based on these questions and answers, we can derive a set of ADICO statements, which is a great starting point for developing an agent-based model as the types and their attributes are clearly mapped out.

5.5 ADICO statements

We can define the important domain-specific construct based on the case study goal. However, many interesting use cases were discussed and listed below. Unfortunately, for the scope of this research, we cannot fulfill all the use-cases of SearchSECO.

- Use case 1: “How can SearchSECO maintain a self-governed entity and reach an equilibrium as an endogenous system.”
- Use case 2: “How can SearchSECO win over end-users to start using the system.”
- Use case 3: “What is the effect of a token-based approach, and how is the value of a token calculated?”
- Use case 4: “What is the adoption rate of people committing to SearchSECO after big events such as conferences and publications domains of interest.”

Use case one is chosen because SearchSECO lacks the behavioral data of its end-users. As use cases two, three, and four rely more on behavioral data, we decide to get involved in use case 1. The goal of the case study is defined as follows:

> **Use case**: How can SearchSECO maintain a self-governed entity and reach an equilibrium as an endogenous system.

The goal of SearchSECO is to maintain self-governed. This means that SearchSECO can be seen as a non-profit organization maintained by the agents interacting with the system. This can be accomplished by reaching a balance where users add value and resources to the system in return for fragments of interest. We consider endogenous factors, meaning that data on agents is not simply descriptive (static), but agent attributes are calculated by the model and updated (dynamically).

Table 5.2 gives an overview of the ADICO statements derived from the systems requirements. To allow for separating the components, a color scheme is used as follows.

```
ADICO = Attribute Deontic Aim Conditions Or Else.
```

TABLE 5.2: Mapping of ADICO statements based on case study criteria.

#	ADICO statement
1	Software producing organizations must contribute to the system if they wish to demand resources from the system or else the agent runs out of credits and possibly restricted access.
2	SearchSECO should remain sustainable under the influence of agents with varying behavior.
3	Empirical software engineers should pay for reading access after losing all credits or else the agent is removed from the network.
4	SearchSECO could invest where 50,000 euro for the initial funding of the framework.
5	SearchSECO must provide an infrastructure where two or more PCs (agents) can connect AND share resources without going through a separate server Or Else SearchSECO cannot exist.
6	software producing organization might access 10,000 code fragment per month Or Else.
7	empirical software engineer might acess 100,000 code framents per year Or Else.
8	Each agent interacting with the SearchSECO platform could perform 15,000 GitHub calls per hour or else a ratelimit will safeguard the infrastructure.
9	SearchSECO might host 200,000 software producing organizations on the platform.
10	SearchSECO might host 50,000 empirical software engineers on the platform.
11	It is contingent that SearchSECO defines 0.20 euro per GB per month as a cost of storage or else no self-sustainable business model exists .
12	It is contingent that SearchSECO defines 1 euro per 10,000 items collected as a cost of processing power (CPU) or else no self-sustainable business model exists .

The statements aim to describe the rules upon which agents act. The aforementioned rules serve as a common source pool for both agent-based simulations and smart contracts. This study aims to evaluate if agent-based modeling can help in designing a smart contract. A smart contract is written on rules which stem from the systems requirements. Hence, by mapping the system requirements through ADICO statements, we inherently create a source pool that allows easy mapping between agent-based modeling and smart contracts.

Chapter 6

Simulation

In this chapter, the performed steps in the evaluation process (as described in B.2) are discussed in more detail. Three scenarios based on the case study discussed in Chapter 5 have been conducted. The first scenario aims to represent a peer-to-peer sharing network where data supply and demand are balanced. This scenario aims to evaluate if such a framework can be built upon the ADICO statements conducted from the case study as seen in Table 5.2. A second scenario aims to add complexity by bringing multiple agent types into the context. The third scenario aims to bring economic factors to evaluate whether or not the system could remain sustainable once money comes into play.

This simulation is detailed in full using the Overview, Design Concepts, and Details protocol (Grimm et al., 2006). Repast Simphony is used to develop the simulation.

6.1 Purpose

The study aims to seek an approach to support smart contract development. Hence, the simulation aims to understand how different types of agents and their respective properties influence the models' outcomes. The output of the simulation provides findings that can be used to adapt decision-making policies on smart contracts. These decision-making rules stem from ADICO statements derived from the systems requirements at hand. ADICO components can be mapped to SOLIDITY constructs which are used to generate smart contracts. The dynamics that emerge from the system are that there will be a sustainable network. Every agent needs an incentive to contribute to the network to profit from the network. This dynamic can be disturbed if the incentives are not optimized.

6.2 Entities, state variables, and scales

The entities that make up the simulation are agents interacting as a P2P network. The agent-based model demonstrates the main features in a peer-to-peer network on an abstract level. SearchSECO agents, software producing organization agents, and empirical software engineer agents are three agent types that comprise a virtual network.

At the core, peer-to-peer network architectures enable resource sharing directly between autonomous individual network users. Hence, an abstract peer-to-peer network is a simulation for the SearchSECO case study as they share a similar infrastructure. The simulation's resources are most commonly files containing information content such as code. No actual data is shared in this simulation but rather arbitrary data in the form of integer values. This decision is made to capture findings based on the ADICO statements were monitoring the content of data (code fragments) explicitly is not mentioned. More valued are the defining characteristics of the network, which are resource availability, consumption patterns, and most importantly, reaching an equilibrium state so that the network remains stable under varying conditions.

6.2.1 Agents

All agents are participants of the network. They use many state variables (Table 6.1). Each agent has a unique *IP address*. The *starting credits* indicate the number of credits an agent receives before initializing to the network. This parameter indicates an incentive that serves to lower the entry barrier to participate in the network. The *starting money* indicates the capital this type of agent possesses at the start of the initialization, consumed once consumption for that agent is particularly high and exceeds the balance ratio. *Download rate* and *upload rate* are variables that can be set to manipulate the equilibrium state of that agent. Lastly, a *request distribution* and *upload distribution* are variables that map to a convenience method that gets a yes/no decision based on the p-value set when the distribution is created.

TABLE 6.1: Overview of state variables for agents

Parameter	Type	Default Value
IP	Float	Random
startingCredits	Double	20
startingMoney	Double	2
downloadRate	Double	0.1
uploadRate	Double	0.1
requestDistribution	Distribution	p-value = 0.1
UploadDistribution	Distribution	p-value = 0.1

Software producing organizations and empirical software engineers are inherited from the same abstract class, meaning their properties are equal. This allows mapping goals for both agents while configuring different properties for those agents. For instance, software-producing organizations could have a higher upload ratio to the network as they produce more code respective to empirical software engineers. The goal of SearchSECO agents is to establish a network to which different types of agents can connect and interact. Furthermore, insights over the whole model are stored through the SearchSECO agent. For instance, an overall number of uploads and downloads or the economic balance over time.

Each agent is configured through a series of parameters that can be changed before initializing the model. Like this, multiple different scenarios can be simulated. For the first scenario to obtain equilibrium across different agent types, a configuration has been provided where software producing organizations and empirical software engineers have an equal upload and download rate. Furthermore, both agent types started with an equal amount of credits, and an equal amount of nodes were configured. However, through a random seed as well as a uniform distribution based on a probability value, a decision for each node will be made as to whether or not the node will contribute in a given iteration to the network or not.

6.3 Process Overview and Scheduling

The model performs discrete steps that come in one temporal form, namely ticks. This process is executed in a serial order. The algorithm listed as Listing A.2 in the appendix aims to provide a simplified overview of the scheduled process and serves as a starting point for the simulation. Listing A.2 serves as a reference to the flowcharts that represent an abstract visualization of the code.

6.3.1 Scheduled Method

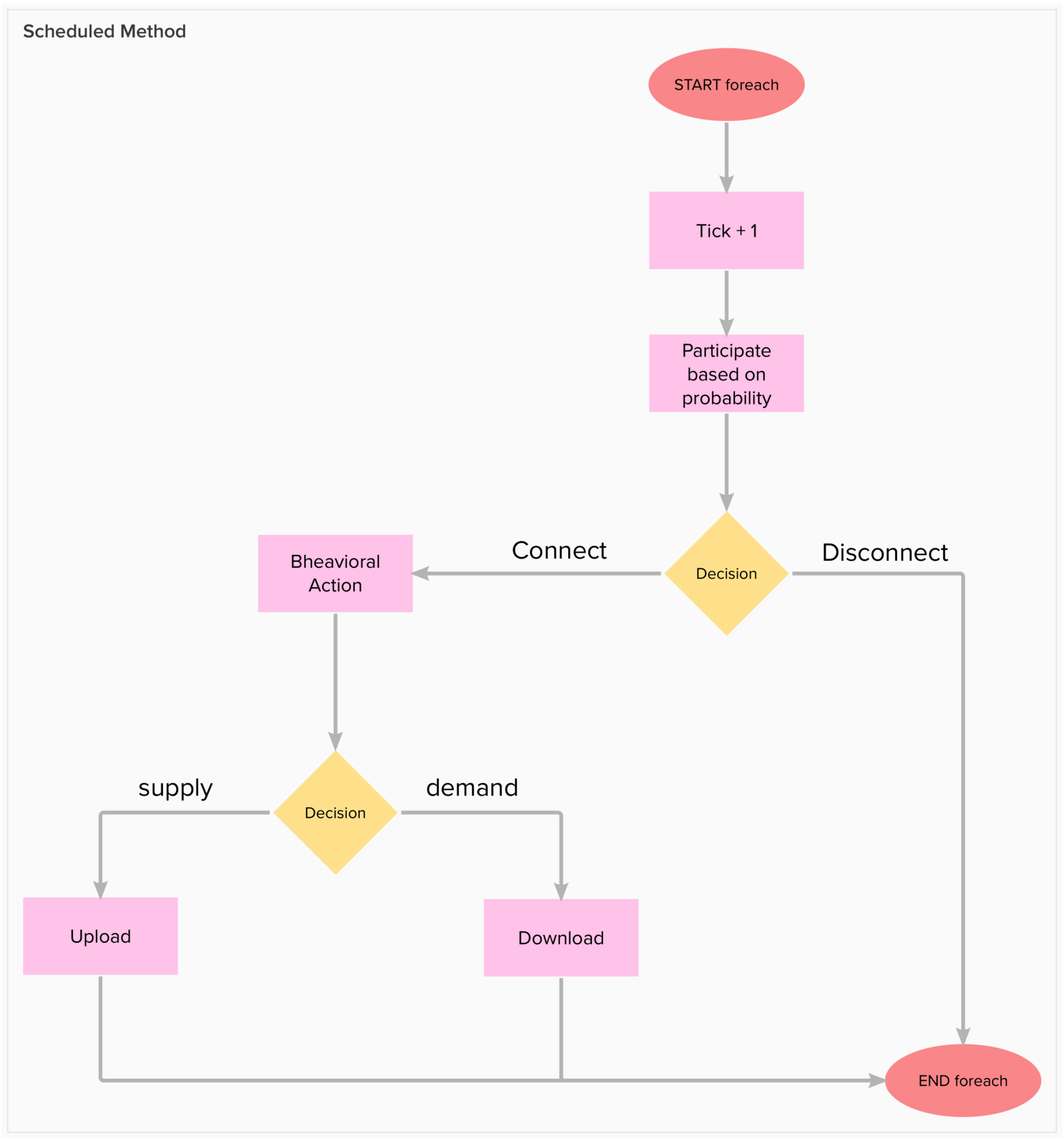


FIGURE 6.1: Process flow of the scheduled method responsible for established a connection.

The model initializes with a process starting from the first tick. This method is annotated with the *scheduled* annotation, which can be used to configure and schedule tasks. This translates into a method that is executed at a fixed interval of time. Figure 6.1 initializes the context of the agent and obtains the projection in which this agent will serve. Nodes can randomly participate in the network based on a probability configured via parameter. The decision either lets the agent connect or disconnect from the network. By connection, the agent will perform an action based on the configuration of that agent. The behavior of the agent is configured through parameters. A resulting upload or download method is called. Once the download or upload is completed, the process ends. The next process flow details the upload/download method in more detail.

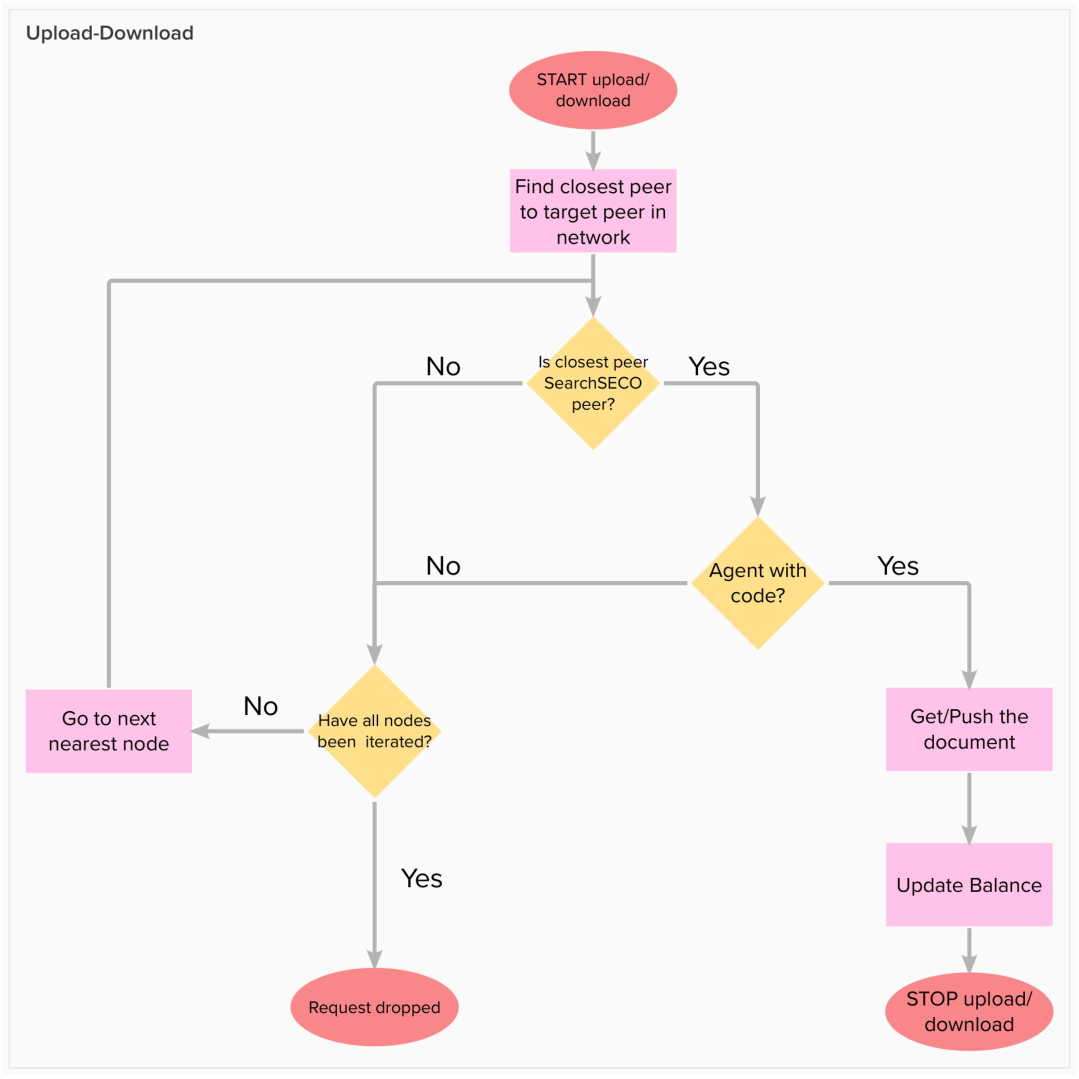


FIGURE 6.2: Process flow of upload-download scenario.

The process displayed in Figure 6.2 builds further on the scheduled method. This listing displays how the agents iterate over a collection of objects which are chosen at random. However, the object must be of the *SearchSECOAgent* type, meaning that this type of agent can only establish a relationship with SearchAgents. If the agent type is not of the SearchAgent type, then an evaluation is done to detect whether or not all nodes have been iterated. Iteration through nodes either happens as long as the node is not of the SearchSECO type or if the agent in question cannot supply the demanded data. Fortunately, if the agent type is of the SearchSECo type and the agent contains the requested data, the process can continue. A new balance will be calculated after the agent established a connection to the SearchAgent. Note that the calculation of the balance depends on a series of factors. This will be further discussed in Figure 6.3.

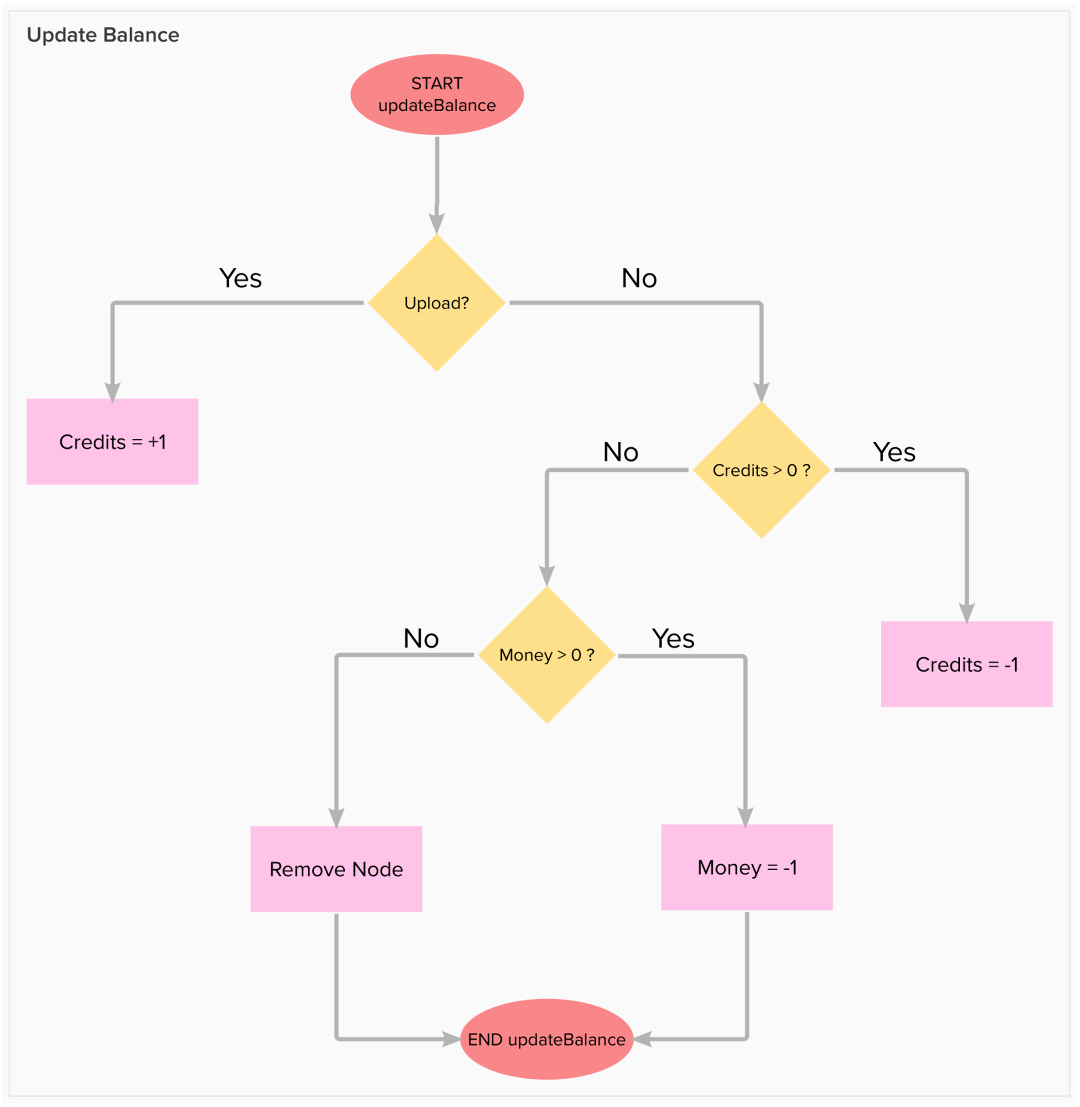


FIGURE 6.3: Process flow of the balance update.

The process displayed in Figure 6.3 is called once the agent connects to the search-SECO network, or in other words, the SearchAgent. The responsibility of this method is to reward or deduct credits to the agent-based upon the action performed. If the agent uploads to the network, then this agent will be rewarded with credits. Vice versa, consuming data or downloading results in a deduction of credits. Note that if the agents do not hold credits, money should be used instead. Once the reserves of credits and funds are gone, this agent will be removed from the context.

6.4 Design Concepts

The simulator is implemented over Repast Simphony 2.8.0, extending its capabilities to deal with the P2P scenario. Its architecture has three components: a Graphical User Interface (GUI), an Agent Simulator, and a Network Simulator. Figure 6.4 depicts a GUI screenshot that illustrates its general appearance. The Control toolbar pertains to the original Repast GUI and allows, among other features, to play the simulation, pause it or execute it step by step. On the left area. Users can modify different agent options.

Emergence
In theory, agents will change their behavior based on their reward upon contributing to the network. The difference in value satisfaction leads to different behavior in the network. For instance, if an agent needs to perform much work to obtain a small piece of data or information in return, then the drop-out rate might increase.

Adaption
The model could implement several adaptive traits within agents. Agents might perform certain actions based on dependencies. For instance, if tasks A and B results in an equal amount of effort, but the reward for task A is higher, then the agents might prefer to perform an action on task A over B. The agent might change their behavior based on such treats allowing them to satisfy certain values. The model at hand did not implement such adaption parameters.

Objectives
The agent's objective is to participate and demand value from the network by contributing a certain effort. The objective might defer from agent to agent. A software-producing organization might have more interest in uploading to the network but receiving validation rather than credits. In contrast, an empirical software engineer might have more interest in gathering data for research.

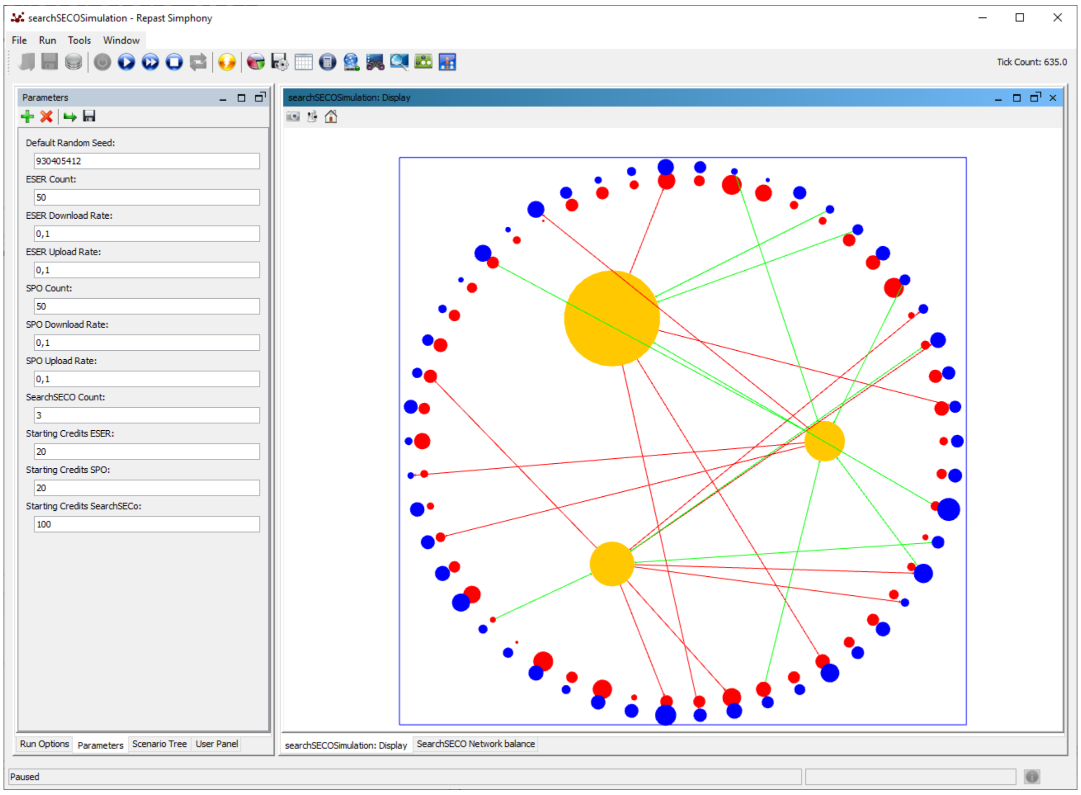
Learning
The agents in this model do not learn. However, one could learn from the output by changing the number of agents and their parameters. For instance, when there is no network balance left after two steps based on altering the number of agents, one could conclude that more agents are required for a stable or sustainable network.

Stochasticity
There are several stochastic choices. For instance, through a random seed as well as a uniform distribution based on a probability value, a decision for each node will be made as to whether or not the node will contribute in a given iteration to the network or not. The probability is not completely random as they depend on a probability factor given at the configuration. The probability is a value of one by

ten. The search for a relationship between the Upload distribution and download distribution is randomly determined.

6.4.1 Interface

The model's interface is illustrated in Fig 6.4, where the layout comprises a context, data loaders, a 2D projection display, and model parameters. The context describes the hierarchy of the model. This context hierarchy is composed of the projections associated with the contexts the model uses. The projection display comprises a continuous space, a grid, and a network. The order of the projection elements does not matter, and the order of the attributes within also does not matter.


searchSECOSimulation - Repast Simphony
File
Run
Tools
Window
Tick Count: 635.0
Parameters
searchSECOSimulation: Display
Run Options
Parameters
Scenario Tree
User Panel
searchSECOSimulation: Display
SearchSECO Network balance
Paused

Default Random Seed: 930405412
ESER Count: 50
ESER Download Rate: 0,1
ESER Upload Rate: 0,1
SPO Count: 50
SPO Download Rate: 0,1
SPO Upload Rate: 0,1
SearchSECO Count: 3
Starting Credits ESER: 20
Starting Credits SPO: 20
Starting Credits SearchSECo: 100

FIGURE 6.4: The Model in Repast Simphony

6.4.2 Styles

The model incorporates different styles for different agents. Software producing organizations are displayed as blue nodes, whereas empirical software engineers are displayed as red nodes. The network nodes which represent SearchSECO are displayed in orange. Furthermore, the size of each node changes dynamically based on the number of credits each node owns. In addition, the connection between nodes is displayed with edges which can either be green for contributing to the network or red for retrieving from the network.

6.4.3 Data Sets & Charts

To interpret the model and the output, thereof an aggregate data set is configured. The selected data sources added to the data set are of the Agent Type 'SearchSECOAgent' and a Method 'getSearchCredits' on which a sum operation is performed. This aggregate data can be written to a text sink used in analytical tools such as RStudio. Furthermore, Time Series Charts inside the tool can be created in Repast Simphony. The data represents the SearchSECO Network balance over time. The first scenario aims to hold a stable number of credits so that no power nodes are created.

6.5 Input Data

The model does not use input data to represent processes over time.

6.6 Observations

For the initialization, the default parameters in section entities, state variables, and scales are used. There can be exceptions, but these will be denoted. To analyze the simulation, an initialization run will first be done. The abstract values can now be varied, and by using the initialization run data as a starting point, it is possible to see how those values influence the simulation. Thereafter the simulation will be run for a couple of hundred ticks. The model's output is written to a text sink that can be interpreted through charts in the simulation or interpreted through statistical tools to perform deeper statistical analysis.

Chapter 7

Outcome

This chapter describes the outcome of the simulations based on the stated scenarios in section 5.3. The chapter describes three scenarios. First, a brief introduction describes the context of the scenario. Second, the run conditions of the scenario are given, which describes the agents' properties. Third, the expected behavior is detailed. Followed by the outcome. Subsequently, the outcome is analyzed, and an interpretation of the results is given.

7.1 Scenario 1: Towards an Equilibrium State

The scenario in question aims to **force software producing organizations to contribute to the system if they wish to demand resources from the system**. We can take a demand-based approach, that when a software producing organization wishes to read an item from our database, she also needs to add an item. This scenario stems from one of the systems requirements discussed in the case study. This requirement has been mapped as the first ADICO statement, as seen in Table 5.2. These decision-making rules stem from ADICO statements derived from the systems requirements at hand. ADICO components can be mapped to SOLIDITY constructs which are used to generate smart contracts.

Software Producing Organizations must contribute to the system if they wish to demand resources from the system or else the agent runs out of credits and possibly restricted access.

The design of the framework is based on a peer-to-peer sharing network. We learned that these networks perform well when there the law of supply and demand is in balance. Supply and demand is an economic model of market value determination in microeconomics. It assumes that, in a competitive market, the unit value of a particular good, or other traded object such as labor or liquid financial assets, will fluctuate until the quantity demanded equals the quantity supplied, resulting in an economic equilibrium for value and quantity transacted (Gale, 1955). It forms the theoretical basis of modern economics Gale, 1955, and it forms the theoretical basis of the first scenario at hand.

7.1.1 Iteration 1: Defining a neutral starting point with one agent

In a first iteration, one SearchSECO agent is simulated together with one software producing organization agent type. The simulation aims to measure the network balance in the form of arbitrary credits within a network. In this context, there is little to no influence on the agent types. The software producing organization agent type will connect with the SECO agent based on a ten percent probability. The simulation is run for three different periods. That being 100 ticks, 200 ticks, and 500 ticks. The settings share no difference across each run.

Run Conditions

TABLE 7.1: Run Conditions Iteration 1 - The effect of one agent on the network balance.

	SECO		SPO			
Ticks	Nodes	Credits	Nodes	Credits	Up:	Down:
100-500	1	30	1	10	.1	.1

In the first iteration, the run conditions serve as a neutral starting point for the upcoming iterations where more complex scenarios are modeled. The first iteration displays one SECO node and one software-producing organization node. The SECO network balance initiates with 30 credits, whereas the software producing organization network balance initiates with only ten credits. Furthermore, the upload ratio is equal to the download ratio, which is 0,1. The translates into a contribution to the network of ten percent. The simulation is run multiple times between a range of 100 to 500 ticks.

Expectations

Based on the run conditions described in 7.1, one would expect a stable network balance. Though the network balance might slightly go up or down during the simulation, one would expect the sum at the end to deviate little from the starting credits.

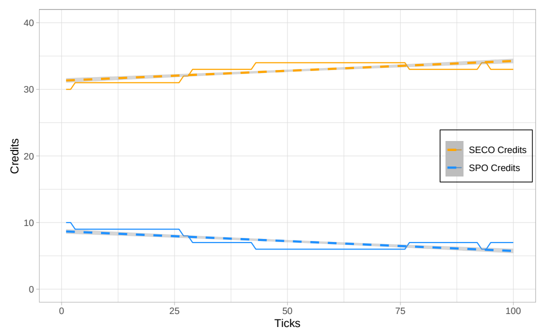
Outcome

Figure 7.1 Indicates the number of network credits during the run based on different time periods while one software producing organization agent is connected to the network. Figure 7.1d displays the variance of a long-term simulation. The network balance of the software-producing organization decreases steadily over time, as seen in Figure 7.1c. At about 300 ticks, the agent loses all of its initial ten credits and can no longer request data from the SearchSECO network. Hence the curve remains unchanged after 300 ticks. Note that only long-term simulations are displayed in future iterations to keep a constant pattern to ease comparison between iterations.

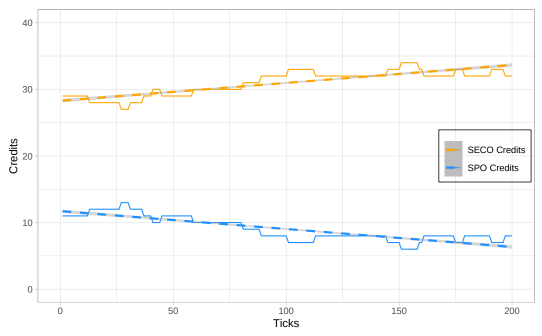
The mean value of the SECO network balance is 36,54, with a minimum value of 29 and a maximum value of 40. Furthermore, the mean value of the software-producing organization is 3,46, with a minimum value of zero and a maximum value of 11.

Interpretation

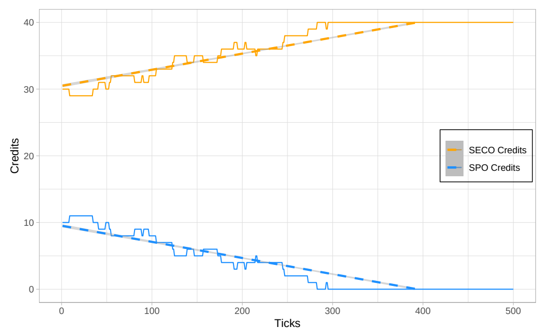
It was expected to find a stable curve with little to no deviation from the mean value (36,54) of the network balance. However, as shown in Figure 7.1, it can be concluded that the network balance could either increase or decrease steadily over a longer time span. This can be explained by the fact that simulating only one agent does not contribute enough value to evaluate the effect described as a network. Furthermore, the simulations are run based on a random seed, which does not guarantee that an



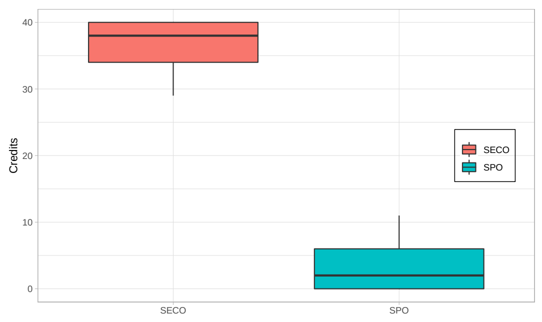
(A) Balance: short-term



(B) Balance: medium-term



(C) Balance: long term



(D) Comparing balance of software producing organization with SECO

FIGURE 7.1: Iteration 1 - The effect of one agent on the network balance.

exact normal distribution is delivered through the upload and download distribution function. The distribution function allows a node to connect and download or upload to the network based on a probability.

7.1.2 Iteration 2: Increasing Download Ratio

In the second iteration, we aim to evaluate if behavior can be added to an agent. Iteration two aims to evaluate the effect of an increased download ratio of the software-producing organization agent. Other run conditions remain unchanged in comparison to iteration 1.

Run Conditions

TABLE 7.2: Run Conditions Iteration 2 - Increasing download ratio of software producing organization

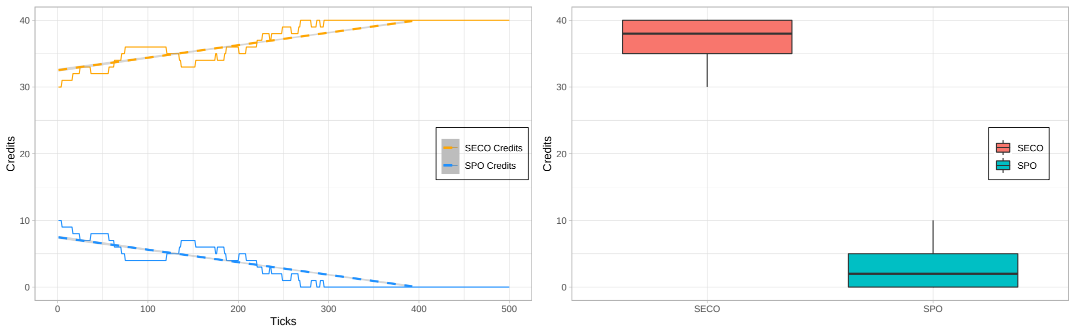
	SECO		SPO			
Ticks	Nodes	Credits	Nodes	Credits	Up:	Down:
100-500	1	30	1	10	.1	.2

The parameters for iteration two remain mostly unchanged to iteration one. However, the Download ratio of the software producing organization agent is doubled to 20 percent instead of the ten percent seen in iteration one.

Expectations

It is expected that for every run performed, the network balance of the software producing organization agent depletes to zero, while the network balance for the SECO agent increases to 40. This can be explained by the fact that the software-producing organization agent obtains only ten credits at the start of the run. Hence these credits will be added to the SearchSECO network balance.

Outcome



(A) Balance increase download ratio

(B) Comparing variance software producing organization and SECO

FIGURE 7.2: Iteration 2 - Increasing download ratio of software producing organization

Figure 7.2a displays the network balance of both the software producing organization agent and SearchSECO agent. After roughly 300 ticks, the software producing organization credits deplete to zero, while the SearchSECO network balance totals 40 credits. Figure 7.2b, indicates the variance of iteration 2 through a box plot.

Interpretation

The outcome of the simulation poses no unexpected results based on the expectations of the model. The simulation shows that by increasing the download ratio of the software-producing organization agent, the network credits of that agent will with certainty deplete to zero. This simulation was run multiple times to confirm this conclusion. Hence the simulation shows evidence that the download behavior of one agent can be successfully configured.

7.1.3 Iteration 3: Increasing Upload Ratio

Similar to the second iteration, we aim to evaluate if behavior can be added to an agent. As opposed to iteration two, where the download behavior of an agent has been configured, the upload ratio of an agent is configured. Hence, the upload ratio of the software producing organization agent is doubled to twenty percent in iteration three. In contrast, the download ratio is against set back to ten percent, as seen in Table 7.3.

Run Conditions

Following up on iteration two, the parameters for iteration four remain mostly unchanged. As opposed to iteration two, the download ratio will be set back to ten

TABLE 7.3: Run Conditions Iteration 3 - Increased Upload Ratio for software producing organization

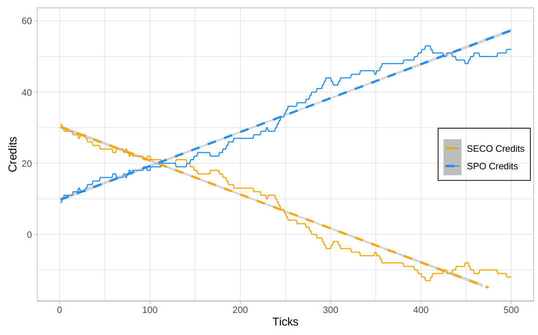
	SECO		SPO			
Ticks	Nodes	Credits	Nodes	Credits	Up:	Down:
100-500	1	30	1	10	.2	.1

percent, whereas the upload ratio will be double to twenty percent. The simulation is run between a range of 100 to 500 ticks. The number of nodes for each agent type remains unchanged, which is one.

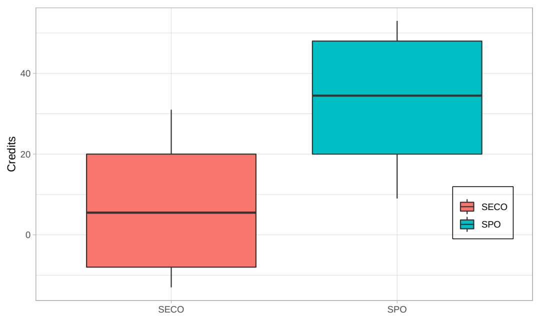
Expectations

Based on the run conditions described in 7.3, one would expect the network balance of SearchSECO to decrease to zero with certainty. On the contrary, the network balance of the software-producing organization agent should, however, increase.

Outcome



(A) Balance increase download ratio



(B) Comparing variance software producing organization and SECO

FIGURE 7.3: Run Conditions Iteration 3 - Increased Upload Ratio for software producing organization

Figure 7.3a Indicates the number of network credits during a run of 500 ticks while one software-producing organization agent is connected to the network with an increased upload ratio. Figure 7.3b displays the variance of a long-term simulation. The network balance of the software-producing organization increases steadily over time, as seen in Figure 7.1c. At about 100 ticks, the software producing organization agent doubles its ten credits to twenty. In a very steady fashion, this curve keeps increasing.

Interpretation

Although it was expected that the balance of the software producing organization agent would increase, it was not considered that the network balance of SearchSECO would deplete below zero. However, as no fallback logic is configured if SearchSECO goes below, the network will have to bear the consequences. It can be concluded that it could potentially be unwanted that a user becomes a power used if one keeps performing upload tasks which will damage the network balance of the network. At this point, we find that finding a good balance is important for a network to survive.

We find that one agent might not be enough to evaluate the effects of a certain behavior on a network. Hence, in the following iterations, an increased number of agents will be configured to obtain a more realistic scenario and evaluate the effects of multiple agents performing tasks on a network.

7.1.4 Iteration 4: Increasing number of agents

As concluded in the previous iterations, it is possible to configure a certain behavior for an agent type. However, the effects on the network could be rather extreme and bring the network out of balance since only one node was simulated while striving for one goal only. That was either uploading or downloading. Hence, to overcome this barrier, more agents will be added in this iteration.

Run Conditions

TABLE 7.4: Run Conditions Iteration 4 - Increased number of nodes in the network.

	SECO		SPO			
Ticks	Nodes	Credits	Nodes	Credits	Up:	Down:
100-500	5	30	5	10	.1	.1

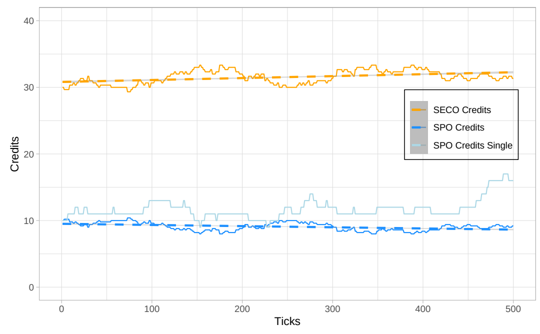
The parameters for iteration four display an equal download and upload ratio for the software-producing organization agent. The number of nodes for each agent type increased from one to five. The simulation is run multiple times for different time periods ranging between 100 to 500 ticks.

Expectations

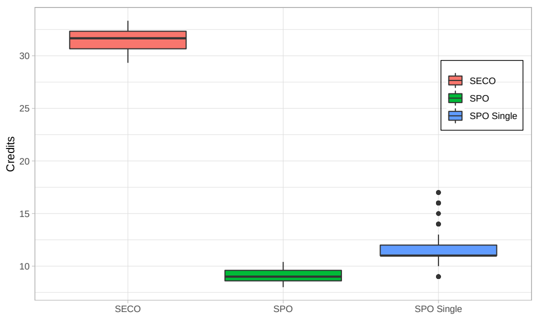
By increasing the number of nodes of each agent type, one would expect a more stable network balance during iteration four. As could be seen in the first iteration, there was much variance in the network. The simulation could not be predicted accurately due to the nature of randomization and a probability factor of the distribution to which the agents connect to the network. However, by adding more nodes to the network and aggregating the network balance of these agents, one would expect a more stable scenario. During this simulation, the network balance of a single software-producing organization agent is simulated and makes even more interpretations.

Outcome

Figure 7.4a displays a stable network balance for both the software producing organization agents and SearchSECO compared to Figure 7.1c. Figure 7.4b concludes these findings by summarizing this data. The SearchSECO network balance reaches a minimum of 29 credits and a maximum of 33 network credits. The mean value of the SearchSECO network credits is 31,54, which is only an additional 1,54 credits from the starting point. In addition to the SearchSECO network balance and the software producing organization network balance, the balance of one single software producing organization agent in the simulation is simulated. Looking at the data of the Single software producing organization agent, a minimum number of credits is measured to be 9. In contrast, the maximum network credits are to be measured at 17 credits towards the 500 tick mark.



(A) Stable balance



(B) Comparing variance software producing organization and SECO

FIGURE 7.4: Iteration 4 - Increased number of nodes in the network.

Interpretation

As we increase the agents, we can tell there is less variance in the data. Hence, the box plots are very narrow as the min value of 29 credits and the max value of 33 is very close to the mean value of 31,54 credits. Although the same number of credits per agent is used as the starting point, the box plots vary greatly due to increased agents, stabilizing the curve and box plots. Hence, an interesting finding, more agents equal more stability on the network. Furthermore, one agent's extreme behavior has less effect on the network, which can be seen at the 500 tick mark where a single agent's software-producing organization credits increase heavily. However, the aggregated number of credits and, therefore, the linear model remains very flat.

7.2 Scenario 2: Altering behaviors - adding complexity

The first scenario simulates a set of agents connected to the network who share some data by either uploading or downloading the data. The upload and download ratio of the empirical software engineer agent type is ten percent. This translates into a convenience method that gets a yes or no decision. The probability is set when the distribution is created based on the upload and download ratio for that agent. The equilibrium state is evaluated in terms of time and network actions during the sharing process. The longer the network remains stable, the better the trustworthiness of the model. We conclude that the model simulates a fairly stable outcome as the deviation of the mean remains within ten percent of the starting point. Only one agent type interacted with the network in the previous scenario, which prevents gathering more complex insights. The simulation should provide capabilities that can aid decision-makers in gaining insight into different types of behaviors.

In this scenario, the second type of agent is introduced, which is the empirical software engineer. The empirical software engineer aims to bring a varying behavior as opposed to the software producing organization. The empirical software engineer will aim to gain knowledge from the network on a theoretical level and thus download more data from the network. In contrast to the empirical software engineer, the software producing organization aims to potentially upload more data to the network to gain insights into their data from SearchSECO. For instance, the software producing organization would push data to the network, evaluated against security metrics, code quality checks, and potentially a risk score by the SearchSECO network. Therefore, this scenario aims to keep **remain a sustainable infrastructure**

under the influence of actors with varying behaviors. This scenario stems from one of the case study requirements. This requirement has been mapped as the second ADICO statement, as seen in Table 5.2. These decision-making rules stem from ADICO statements derived from the systems requirements at hand. ADICO components can be mapped to SOLIDITY constructs which are used to generate smart contracts.

SearchSECO should remain sustainable under the influence of agents with varying behavior.

7.2.1 Iteration 5: Adding new agent type empirical software engineer

With the fifth iteration, the empirical software engineer is added to the simulation. Iteration five aims to evaluate if the network could remain stable when multiple agent types are present. In further iterations, the behavior will be altered to gain more interesting findings, but for now, the simulation is kept simple to monitor if any problems arise.

Run Conditions

TABLE 7.5: Run Conditions Iteration 5 - Adding the empirical software engineer to the network.

	SECO		SPO				ESER (New)			
Ticks	Nodes	Credits	Nodes	Credits	Up:	Down:	Nodes	Credits	Up:	Down:
100-500	5	30	5	10	.1	.1	5	10	.1	.1

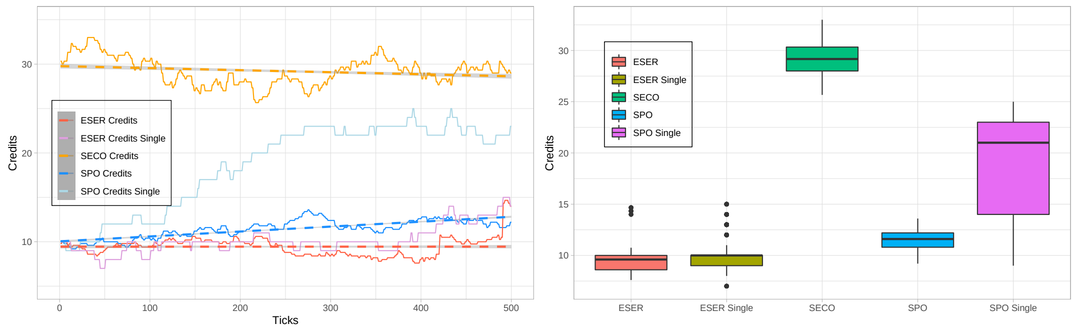
As seen in Table 7.5 run conditions remain similar as seen in Table 7.4. The number of agents per agent type remains five. In addition, the upload and download ratio remains stable. However, it can be denoted that the empirical software engineer agent is added to the network with identical properties as the software producing organization agent type.

Expectations

In the fifth iteration, it is expected that the network balance of SearchSECO has more variance, as seen in iteration four. This can be expected as the total number of the agents is doubled. Hence there is a lot more interaction with the network. Comparing this to iteration one, we concluded that a stable network could not be guaranteed for long-term iterations due to the nature of randomness, as one agent could show more interest in gaining from the network rather than contributing. However, in iteration four, we concluded that by adding more agents to the network, a more stable network is expected as the balance of the network is aggregated with the number of agents. Even though that there might be twice as much interaction with the network, the outcome of SearchSECO's network balance should remain stable despite more interaction.

Outcome

Figure 7.5a displays the output of the fifth iteration where the empirical software engineer is added to the network. In contrast to Figure,7.4a this simulation shows a



(A) Stable balance

(B) Comparing variance software producing organization and SECO

FIGURE 7.5: Iteration 5 - Adding the empirical software engineer to the network.

wider variety in the network balance of the SearschSECO agent type (orange line). However, the final network balance of SearchSECO totals 29,8 credits.

Furthermore, the output of iteration five displays the balance of the software producing organization agents on the network and the balance of the empirical software engineer. In addition to the aggregated sum of network credits, the simulation displays the network credits of one single agent in addition. The network balance of the empirical software engineers remains very stable over time, whereas the network balance of the software-producing organizations slowly increases over time. The network balance of the single software-producing organization, which is simulated, seems to display a high increase in network credits.

Figure 7.5b displays a box plot for every agent type in the network. Both the aggregated network balance of the empirical software engineer and the single node contains some outliers. The mean value of the SearchSECO network balance is 29,2, with a minimum value of 25,67 and a max value of 33 network credits. The mean value of the empirical software engineer is 10,2, with a minimum value of seven and a maximal value of 15. Furthermore, the mean network balance of the software producing organization is 11,4, with a minimum value of 9,2 and a maximum network balance of 13,6. However, note that the software producing organization network balance of a single node is displayed. The mean value of that node is 18,6, with a minimum network balance of nine and a maximum network balance of 25.

Interpretation

As described in the output of iteration five, it can be concluded that the overall network balance of the SearchSECO network remains stable despite having many variations in it. This behavior was, however, expected. This is since the network totals twice as many nodes in the network by adding five empirical software engineer agents.

As described in the output of iteration five, and as shown in Figure 7.5b, there is much variance on the single node of the software producing organization agent. However, having such an outlier is balanced out as the aggregated sum of software-producing organization agents cancels this extreme behavior. Hence, why the blue linear line in Figure 7.5a only increases slightly and not extremely.

7.2.2 Iteration 6: Adding different behavior to agent types

Previous iterations display the capabilities of the simulation when all agent types have equal properties. The upload and download ratio is tuned to be in balance. Both agent types (software producing organizations & empirical software engineers) have an equal amount of credits.

The following scenario shows the system's capabilities by configuring the agents with different properties, making it a complex scenario. The aim is to bring a varying behavior as opposed to the software producing organization. The empirical software engineer will aim to gain knowledge from the network on a theoretical level and thus download more data from the network. In contrast to the empirical software engineer, the software producing organization aims to potentially upload more data to the network to gain insights into their data from SearchSECO. In iteration five, only the empirical software engineer agent was introduced but displayed very similar behavior to the software producing organization agent type.

Hence, in the sixth iteration, the configuration of the empirical software engineer and software producing organization will be altered to answer the aim of the second scenario, which is to introduce a demand-based driven behavior for the empirical software engineer and a producing-based behavior for the software producing organization.

Run Conditions

TABLE 7.6: Run Conditions Iteration 6 - Demand based empirical software engineer configuration in contrast to producing based software producing organization configuration.

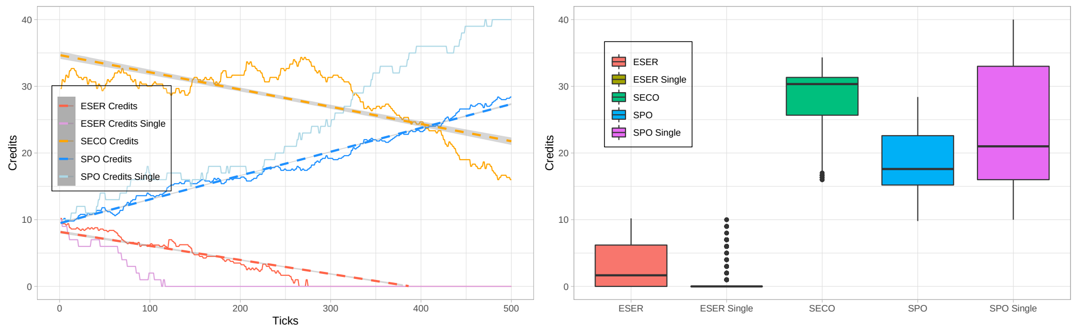
	SECO		SPO				ESER (New)			
Ticks	Nodes	Credits	Nodes	Credits	Up:	Down:	Nodes	Credits	Up:	Down:
100-500	5	30	5	10	.2	.1	5	10	.1	.2

In contrast with iteration five, the configuration for iteration six displays a higher upload ratio of 0,2 for the software producing organization, which is twice as much as seen in iteration five. Furthermore, the download ratio of the empirical software engineer is also doubled to 0,2 in contrast to iteration five. Other parameters remain unchanged in respect to the previous iteration. For each agent type, there will be five nodes representative in the network. The software producing organization and the empirical software engineer agent will start with ten network credits, whereas the SearchSECO agent starts with 30 credits. The simulation is run multiple times over a time span between 100 and 500 ticks.

Expectations

By configuring the simulation so that software producing organization agents represent a producing behavior in contrast to the empirical software engineer agent who represents a demanding behavior, it is expected that the linear models for these agents are very different from each other. It is expected that the software producing organization network balance will steadily increase, at the same rate the software producing organization network balance will decrease. It is expected that this rate at which there is a demand concerning producing is as high. Hence, one could expect that the total network balance of SearchSECO would remain stable as the behavior of the software producing organizations is configured so that it would lift the behavior of the empirical software engineer. empirical software engineer agents will download twice as fast as usual, but software-producing organizations will balance this behavior by uploading twice as fast.

Outcome



(A) Stable balance

(B) Comparing variance software producing organization and SECO

FIGURE 7.6: Iteration 6 - Demand based empirical software engineer configuration in contrast to producing based software producing organization configuration.

Figure 7.6a displays the output of the sixth iteration where the empirical software engineer demands twice as much from the network as opposed to the software producing organization who produces twice as much to the network as described in the expectations the network balance of the software producing organization, which is displayed in blue increases twice as much as seen in iteration five. In addition, the network balance of the empirical software engineer, which is displayed in red, decreases twice as much as seen in iteration five.

At around 300 ticks, the empirical software engineer runs out of network credits. At this point, the SearchSECO network balance, which is displayed as an orange line, will decrease heavily and no longer be stable. Also, the SearchSECO network balance displayed in orange will decrease and eventually deplete to 15 network credits at 500 ticks in contrast to the 30 network credits this agent started with.

Figure 7.6b displays a box plot for every agent type in the network. The mean value of the SearchSECO network balance is 28,2, with a minimum value of 16 and a max value of 34 network credits. The mean value of the empirical software engineer is 2,9, with a minimum value of zero and a maximal value of 10. Furthermore, the mean network balance of the software-producing organization is 18,4, with a minimum value of 9,8 and a maximum network balance of 28,4.

Interpretation

Unlike the expectations of this iteration, it was expected that the network balance of SearchSECO would remain stable. However, as the empirical software engineer Agents reach a below zero credit score, they are eliminated from the network; hence, the software producing organizations will overtake the network and gain power. We will see an imbalance, and the SECO network will, from that point on, lose credits and deplete direction zero. This effect can be seen at around 300 ticks in the simulation where the empirical software engineer will hold no more credits and cannot maintain the contribution to the network.

Therefore, users of the network must have the possibility to participate again in the network despite their behavior. In response to this conclusion, a third scenario is simulated where financial factors play to mitigate this problem.

7.3 Scenario 3: Financial factors as incentives

The scenario in question aims to **request empirical software engineers to pay for reading access**. To make the infrastructure sustainable, we can ask empirical software engineers to pay for reading access to our database. The second scenario displays the effect of complex systems as various agents are performing actions on the network. This scenario stems from one of the case study requirements. This requirement has been mapped as the third ADICO statement, as seen in Table 5.2.

Empirical software engineers should pay for reading access after losing all credits or else the agent is removed from the network.

These decision-making rules stem from ADICO statements derived from the systems requirements at hand. ADICO components can be mapped to SOLIDITY constructs which are used to generate smart contracts.

A financial factor has been added, which the agent will consume if that agent leans towards a higher download or consumption rate. Hence, if the agent is out of credits, the agent will be forced to use actual funds. If the agent cannot contribute to the network and has no more funds, that agent will be removed from the network.

7.3.1 Iteration 7: Financial factors as incentives

In the sixth iteration, behavior is added to both the software-producing organization to perform more supply to the network. In contrast, empirical software engineers demand more from the network. One of the findings in iterations six is that the total network balance of SearchSeco depletes to zero. The reason for this is that agents can run out of virtual credits. In response to running out of credits, the user will have no opportunity to demand or participate in the network anymore. Therefore, in this iteration, a fallback mechanism is added to which agents can use financial credits to participate with the network.

Run Conditions

TABLE 7.7: Run Conditions Iteration 7 - Financial factors as incentives

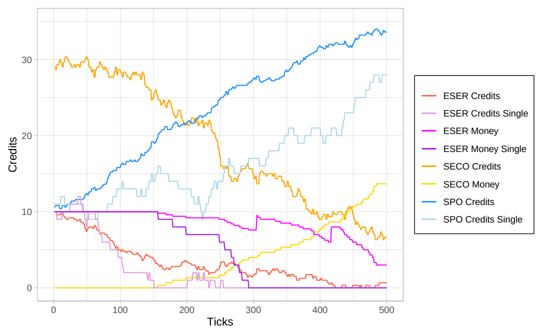
	SECO		SPO				ESER (New)			
Ticks	Nodes	Credits	Nodes	Credits	Up:	Down:	Nodes	Credits	Up:	Down:
100-500	5	30	5	10	.2	.1	5	10	.1	.2

In this scenario, all agents obtained ten network credits as an incentive to participate in the network. This translates into the fact that after performing ten consecutive downloads without any upload (or gathering data twice without a return), the agent will be forced to use actual funds to make further use of the network. In addition to the network credits, an additional 10 financial credits are given to overcome the obstacle of network depletion, as seen in iteration six. Other run conditions remain the same as in iteration six. The software producing organization keeps a supply-based behavior, whereas the empirical software engineer keeps the demand-based behavior. This is seen through the configuration. The upload ratio is twice as high as the download ratio for the software producing organization and vice versa for the empirical software engineer. Multiple simulations are run between a time span of 100 to 500 ticks.

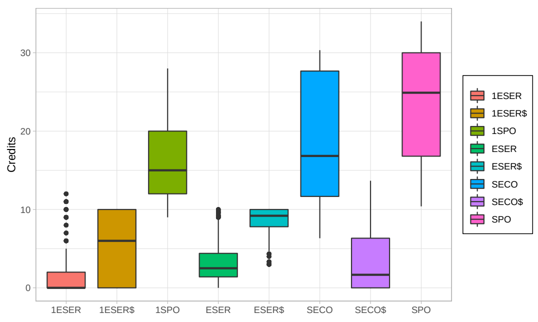
Expectations

It is expected to have a very similar output for the network balance for all types of agents to iteration six, up until 300 ticks. This was when empirical software engineers ran out of virtual credits, and in response to that, could no longer participate in the network. As an incentive, the agents are given ten financial credits, which can overcome this obstacle. Therefore it would be expected that the financial balance would rise from that point on for the SearchSECO network.

Outcome



(A) Stable balance



(B) Comparing variance software producing organization and SECO

FIGURE 7.7: Iteration 7

Figure 7.7a displays the output of the seventh iteration where the empirical software engineer demands twice as much from the network as opposed to the software producing organization who produces twice as much to the network. In addition, the agents are given a financial balance that can be used once they run out of virtual credits to enable engagement with the network.

Firstly the behavior of the single empirical software engineer is described. The single empirical software engineer is modeled with a light pink color in Figure 7.7a. At tick 140, an interesting finding can be picked up as the single empirical software engineer simulated and displayed in light pink loses all his credits. For losing his virtual credits, this node needs to use financial funding, which can be seen directly as the yellow line starts inclining. The yellow line represents the financial balance of SearchSECO, and so this node uses funding instead. This allows the single empirical software engineer to upload to the network once again, which can be seen at tick 200. At tick 200, the single empirical software engineer contributes to the network again as its credits rise again. This phenomenon was not possible until the inclusion of financial reserve. Unfortunately, due to the nature of this agent's behavior after about 255 ticks will have demanded so much of the network that both virtual credits, as well as financial credits (purple line), are used up at tick 300 to which the node can no longer participate to the network.

Secondly, the collective behavior of all empirical software engineers is described through the red curve, which represents the network balance, as well as the dark pink curve, which represents the financial balance in Figure 7.7a. It can be concluded that the collective behavior is the same as for a single empirical software engineer with one difference. The difference is that the curve is stalled to a later point in time. This can be explained as the collective nodes will survive simply longer as some nodes contribute more to the network, as seen for the behavior of the single node. Nevertheless, in conclusion, similar to the behavior of a single node, it can be seen that the financial balance of the empirical software engineers starts declining at about 160 ticks once the virtual credit balance is near zero. The collective empirical software engineers can contribute longer than 500 ticks in contrast to iteration six, where these agents already ran out of business.

Thirdly, it was expected that the network balance of SearchSECO would decline heavily at 300 ticks as this was the result of iteration six. This iteration is also true where a little bit earlier, at about 250 ticks, the network balance decreases heavily, as seen in the orange curve. However, at this point, the yellow curve, which represents the financial credits balance of SearchSECO, started inclining already. SearchSECO would remain once again stable by aggregating the financial balance with the network credits.

Figure 7.7b shows that the first quartile starts at zero, but the third quartile has a value of 6,3 financial credits. However, the maximum value is twice as much at about 13,6 because the financial credits in this scenario can only rise. This would not be true by adding more complicated behavior to the network. However, the empirical software engineer will run dry of both network credits and financial credits in this scenario. There is no way to earn financial credits from the network at this point. However, this could be an interesting finding for the SearchSECO case study who could use this finding to fund its network maintainability.

Interpretation

The seventh iteration aims to prevent the SECO network from collapsing under the influence of actors who purely gain from the network. As seen in iteration six, the empirical software engineer has a very high demand-based behavior, which leads to the effect that this group of nodes cannot balance out the opposite behavior. That is the software-producing organization that contributes greatly through supplying data to the network. In its turn, the software-producing organization is rewarded with credits, making the network unstable once there are no nodes left who can balance this behavior out. This can be seen in Figure 7.7a. After about 250 ticks, the network balance decreases heavily. Fortunately, by introducing financial capabilities, the network can remain stable for a longer period of time and allow the agents to participate and contribute to the network again. Although not discussed in detail, the values of the network credits are purely arbitrary, meaning that they do not hold an actual value. The same goes for the financial credits. In this simulation, both hold an equal value. However, one could state that by aggregating the SearchSECO financial balance to the SearchSECO network balance, the network becomes stable once again.

7.3.2 Summary of Iteration 1 - 7

Table 7.8 describes, in short, the main findings of each iteration. Every iteration is built on top of the previous iteration to add complexity and better descriptions of insights into what is happening in that simulation.

TABLE 7.8: Main findings and lessons learnt from SearchSECO simulations.

Iteration	Finding
Scenario 1:	**Towards an equilibrium state**
1	One would expect a stable network balance if the configuration of the agent type holds an equal upload to download ratio. However, the aforementioned is not true. Due to a distribution function configured through probability and a random seed, the agent's behavior cannot be guaranteed to be stable.
2	By doubling the demand behavior, the network balance of SearchSECO depletes to zero more rapidly.
3	By doubling the supply behavior, the network balance of SearchSECO increases to a maximum value of 40 more rapidly. This can be explained as the software-producing organization starts with 10 network credits transferred to the 30 network credits SearchSECO initialized with.
4	Increasing the number of nodes for each agent type increases more network stability. Furthermore, we conclude that extreme behavior of one agent has less effect on the network as the aggregated sum of network credits cancels this behavior out.
Scenario 2:	**Altering behaviors - adding complexity**
5	Adding an empirical software engineer and the existing software producing organization displays more variance on the network balance. The total number of nodes can explain this is doubled as opposed to the previous iteration. However, even though there is more variance during the run, the final network balance of SearchSECO totals 29,8 credits which is only a 0,2 difference from the starting point. Hence, the network is even more stable than it was before.
6	Different from the previous iteration, behavior is added to each agent type. The empirical software engineer holds a demand-based behavior, whereas the software producing organization holds a supply-based behavior. It is expected that the network balance of SearchSECO would remain stable as the agents configured are each other's negatives. However, an imbalance could be picked up after a while, and the SECO network will deplete direction zero. This can be explained as the users who performed a few consecutive demands from the network could no longer participate in the network as they ran out of funding. In respect to this problem iteration, seven or scenario three is introduced.
Scenario 3:	**Financial factors as incentives**
7	The last iteration aims to request agents to pay for access to the network after demanding consequently from the network. This to make the infrastructure sustainable. Hence, by providing a financial fallback mechanism, we found that SearchSECO could benefit in new ways from their user base by collecting financial credits once the nodes run out of artificial network credits. In addition, the nodes have a possibility to contribute to the network rather than purely demanding from the network by paying a financial amount.

7.4 Observations

The current implementation of an abstract P2P system in Repast Simphony is shown in Fig 6.4. The model serves as a demonstrably useful educational tool where insights can be obtained by modifying various nodes within a network. The tool is suitable for learners to show that data is propagated through the nodes in the overlay according to predefined behavior. The model also includes analyses facilities through charts and text-sinks. The text sinks serve as good input for deeper statistical analysis.

The proposed model demonstrates the important features, such as topology adaption and an unbiased random selection. As seen in Figure ,6.4 the interface design displays how the nodes interact differently, indicating their relative capacity (in terms of funds). Furthermore, the model includes a different collection of modifiers to control the number of nodes in the network and their respective properties.

Chapter 8

Findings and Discussion

This chapter forms the discussion around the main findings of studying an agent-based simulator for optimizing smart contracts. The advantages and the limitations of the research are discussed based on the findings of the case study scenarios. Implications and Limitations are presented, followed by recommendations which state future directions for this research.

To reveal the answer to the main research question, a series of findings will be discussed by answering the sub-questions. These insights indicate how well the study objectives are achieved.

8.1 Major Findings

> Sub-Question 1: How can smart contracts and agent-based modeling be defined based on prior literature?

The features of smart contracts, as well as agent-based modeling, are defined in Chapter 3. Figure 3.1 visualizes the working of an ABM through concept diagrams. Figure 3.5 visualizes the working of a smart contract through a concept diagram. The concepts of smart contracts and agent-based modeling are defined through definitions that are mapped to concept diagrams.

> Sub-Question 2: What research into the application of agent-based modeling to smart contract development has already been conducted?

Kolvart, Poola, and Rull (2016) argues that ICT applications work for and against social interaction, so powerful tools are devised to understand the complexities of the current socioeconomic system. The study emphasizes observing sociological interactions with a system. Boogaard, 2018 makes use of a model-driven approach to develop smart contracts. While Boogaard (2018) mentions ADICO as a possible model-driven approach to design smart contracts, the study in question used a state machine approach instead and did not dig deeper into the topic of agent-based modeling. The work of (Frantz and Nowostawski, 2016), displays a modeling approach that supports the semi-automated translation of human-readable contract representations into computational equivalents is proposed. Frantz and Nowostawski (2016) identifies smart contract components that correspond to real-world institutions and proposes a mapping that is operationalized using a domain-specific language to support the contract modeling process. The work of Frantz and Nowostawski (2016) serves as a great source of inspiration for this thesis. However, it does not dedicate enough attention to linking agent-based modeling and smart contracts. When looking at previous work in software development, there are several of the same concerns with smart contracts, and various ideas have been suggested (Pressman, 2005). Ideas range from sociological analysis to statistical analysis to model-driven engineering. Nevertheless, it can be denoted that the aforementioned concerns stem from a lack of understanding of requirements, the transformation of domain information (Lucassen et al., 2016). Despite a broad range of interesting research in the domain of agent-based modeling and smart contract design, there deems little concrete research that brings these domains together. To the best of our knowledge, there is no systematic method to evaluate smart contracts regarding their effectiveness in finding the optimal method by using agent-based modeling.

> Sub-Question 3: What are the requirements for agent-based modeling and smart contract development?

A set of *agents* with their attributes and behaviors. Moreover, in Section 3.1.3. A set of agent *relationships* and interaction methods: An underlying topology of connectedness defines how and with whom agents interact. Moreover, in Section 3.1.4. The agents' *environment*. Agents interact with their environment in addition to other agents. Moreover, in Section 3.1.5.

In conclusion, for both agent-based models and smart contracts, **the higher-level system properties arise from the lower-level subsystem interactions. Alternatively, basic behaviors (meaning rules followed by agents) create complicated behaviors (meaning state changes at the whole system level)** (Bonabeau, 2002).

> Sub-Question 4: What is the most suitable agent-based modeling tool to support the development of the goal's requirements?

A big set of various agent-based modeling and simulation programs currently available and represented in Table C.1 in Appendix C is defined. The evaluation criteria are described in Table 4.1. The meaning of scalability and development effort is detailed in section 4.1.1 and 4.1.2 subsequently. To allow the ABM community to benefit from this research, a repository is created on Github. Furthermore, as seen in Figure C.1 in Appendix C shows a website that has been deployed to allow for easy searching based on features.

> Sub-Question 5: Can agent-based modeling assist in creating insight into the decision rules of a smart contract for a complex system?

There is no direct evidence that agent-based modeling creates a concrete insight into smart contract development. However, we found various interesting findings on how agent-based modeling and smart contracts can be linked. Both agent-based models and smart contracts have higher-level system properties that arise from the lower-level subsystem interactions. Alternatively, basic behavior rules create complicated state changes at the whole system level. We found that institutional grammar could be used to extract functional requirements and system requirements.

> Research Question: Can agent-based modeling support the design of smart contracts by simulating the systems requirements?

This study shows how different domain concepts regarding agent-based modeling and smart contract design can be mapped, as seen in Figure 3.7. A novel approach displays how collective ADICO statements are created from functional requirements at the operational level. Figure B.2, in Appendix B illustrates a method for gathering insights from agent-based simulations which can guide smart contract development. Hence, agent-based modeling could support the design of a code search community platform by simulating common source rules on which the platform functions.

This method is summarized as follows:

1. Define case study goals, requirements, and validation criteria.
2. Transform case study criteria to ADICO statements.
3. Select agent-based modeling toolkit.
4. Transform ADICO statements to functional code.
5. Evaluate the Utility function by forming a statistical summary resulting in conclusions.

First, case study goals, requirements, and validation criteria are transformed into ADICO statements. Thereafter an agent-based modeling toolkit is selected. As Solidity code for smart contracts can be mapped to ADICO statements and share a similar abstract level, one could transform these statements to functional code. In this study, this transformation has been done manually. However, the automatic transformation could potentially reduce development efforts through transpiling. Transpiling is the act of taking source code written in one language and transforming it into another language. Meaningless errors will be made if the agent-based model code is automatically mapped to Solidity code. Lastly, the utility function of the model is based on the evaluation criteria defined with the case study requirements and validation criteria. The goal of the utility function is to write results that can be interpreted either through a summary, text analysis, or statistical analysis. *Therefrom, we conclude that ABM aids decision-makers in smart contract design.*

8.2 Contributions

Mapping ABM and smart contract domain concepts
The first sub-question of this research asks how smart contracts and agent-based modeling be defined based on prior literature. We find that both concepts heavily depend on a set of rules translated from functional requirements through literature review. An agent-based model is a computerized simulation of many decision-makers (agents) and institutions that interact through prescribed rules. Figure 3.7 visualizes how the different domain concepts relate to each other.
Abstraction through Institutional Grammar
A novel approach was utilized to extract architectural information from requirements. Institutional grammar was used to comprise a list of ADICO statements from which conditions could be derived on both an operational and a collective level. The operational level affects actions and outcomes at the core of the operation. Collective statements are created to be used at the operational level.

ADICO itself is not a framework nor an automated method to writing either smart contracts or architecture for defining agents and their rules for an agent-based model. However, it **could bridge the gap between designing agents for an agent-based model as well as designing smart contracts starting from a collection of ADICO statements that correspond to the rules of both these entities**. This approach is proven to have potential, which can be seen in the paper “from Institutions to Code: Towards Automated Smart Contract Generation.” (Crawford and Ostrom, 1995). Also Smajgl, Izquierdo, and Huigen (2008) proposes a sequence where one could model endogenous institutional rule change for agents. This is done by identifying the structural components of a general rule from a modeling perspective, followed by providing an agent architecture overview. Parts of the proposed sequence are implemented in a NetLogo model.

To shortly recap, for both agent-based models and smart contracts, the higher-level system properties arise from the lower-level subsystem interactions. Alternatively, basic behaviors (meaning rules followed by agents) create complicated behaviors (meaning state changes at the whole system level) (Bonabeau, 2002). Institutional grammar could be used to extract architectural information from requirements in the form of ADICO statements. We can define the important domain-specific constructs by exploring its structural elements between contracts and institutional grammar and suggest a mapping that simplifies contract generation.

Mapping smart contract domain with agent-based modeling domain
Figure B.2 in Appendix B illustrates how a smart contract definition influences the simulation outcome based on the rule-based statements defined through a case study. Firstly, the case study criteria are defined, which results in a goal statement, a collection of rules, and a utility threshold the infrastructure should answer to. Secondly, These statements from the case study are used as criteria for the agent-based model's agents, relationships, and environment. These statements could be translated to ADICO statements translated in both smart contracts and agent definitions in the ABM. Thirdly, the ABM tool selection procedure is executed corresponding to the case study requirements at hand. Finally, the simulation is performed, an evaluation is done through a utility function with summarizes. It exports the results of the data to which analysis can be done, such as statistical analysis. Therefrom, we conclude that ABM aids decision-makers in smart contract design.

ABM Software for simulating P2P networks
This thesis shows how a decentralized system can be implemented in an agent-based modeling tool and provides agents with unique properties. The ABM models a decentralized system with different agents that can resemble a particular behavior based on different input types. Furthermore, different group types can be modeled. For instance, in this work, software producing organizations, empirical software engineers, and the SearchSECO network are modeled with their typical behavior. The model gives insight into the sustainability of the network. Furthermore, the model gives insight into the effect of ratios of nodes connecting to the network. The aforementioned insights could help the owner of the network to mitigate imbalance when composing smart contracts. Insights could be the evaluation of growing power nodes, an imbalance of the upload/download ratio of the network, a depletion of the financial balance of the network. The ABM software serves as a good starting point for evaluating decentralized networks on which can build further.

Case Study
This study contributes by performing a case study for SearchSECO by gathering insights on the sustainability of the SearchSECO network. The development of agent-based models and smart contracts is partially embedded in research. However, for the most part, it is conducted in an online open-source environment in which willing participants collaborate and expand on the work of others without the need for detailed documentation. A valuable scientifically written overview of the current state of smart contract design is presented through this case study. Furthermore, smart contract developers are encouraged by the insights presented in this case study. Smart contract developers should strive to build high-quality smart contracts as efficiently as possible. The development method supporting this goal could contribute to the smart contracts' efficiency, performance, and quality.

8.3 Limitations

Many conventional simulators do not offer an easy way to analyze or visualize the results with larger nodes (Xu, Yu, and Sepehrnoori, 2019). To overcome this barrier, additional scripts to the main implementation of export to another tool are needed. Nevertheless, the process of analyzing data is straightforward with Repast Simphony (Garcia and Rodriguez-Paton, 2016). Furthermore, real-time graphs and text sinks are generated (after configuration), which allows for improved analysis. To conclude, the simulator comes with a friendly interface, demonstrating the dynamic communication between nodes, collecting input fields to customize the simulation, and taking control of the variable inputs and outputs illustrating the current statistics.

It is noticeable that the time needed in Repast Simphony to implement a peer to peer like protocol is high. This conclusion is made based on a four-week implementation span after the requirements were set out. The implementation of the algorithm was completed with an aggregate of 490 lines of code. In addition, the visualization and configuration cost another 220 lines of code.

Further limitations of this research are discussed according to the four aspects of validity for case studies: construct validity, internal validity, external validity, and reliability (Wohlin et al., 2012).

Firstly, concerning construct validity, one threat can be identified. Construct validity refers to the degree to which the study measures its claims (Claes Wohlin, Höst, and Henningsson, 2003). This study adheres to DSR guidelines, as seen in Section 2.2, and the methodology's cycles are iterated properly to support the traceability of the design decisions. Additionally, since one researcher performed all analyses, there is no risk of different data interpretation and results. However, one major threat is found due to the lack of expert interviews. This can be explained by the fact that the case study was performed on a prototypical project where no historical data is present. Hence, a truthful discussion about the results of the simulation could not be established.

Secondly, with internal validity, several threats can be identified. Internal validity describes the extent to which a piece of evidence supports a claim about cause and effect within the context of the study (Claes Wohlin, Höst, and Henningsson, 2003). Two important factors play a role in the outcome of the validity of the simulation. Firstly, the sociological aspect of the agents is missing. There are no sociological rules implemented in the current artifact that describe the agents' rules within the system. This means that the adoption rate of agents interacting with the system is rather abstract and based on artificial input parameters. Second, the input parameters do not stem from historical data, which causes an unrealistic output of the simulation. Lastly, data should hold a relation to either financial or digital currency. This could be handled through an API endpoint, which retrieves financial data in time and space. However, this threat is mitigated due to the nature of the case study where an arbitrary data value has been given.

Thirdly, concerning external validity, there are few threats. The model is implemented of an abstract Gnutella-like P2P system in Repast Simphony. The model serves as an educational tool where insights can be obtained by modifying various nodes within a network. The model can be further developed as the system lays a simple foundation that does not restrict the user to a specific scenario. The system is highly generalizable and could serve different companies or institutions. The only aspect which needs modification is the rules and nature of the agent types.

Finally, concerning reliability few threats are discussed. Reliability describes to what extend the data and analysis depend on the specific researcher and the degree of decision traceability (Roberts and Priest, 2006). By following the DSR guidelines, research bias is mitigated to some degree. This is further improved by using Multivocal Literature Review, which captures both academic and practitioner insights. The preparation and the analysis activities are documented in detail, and a protocol is followed. These protocols leave little interpretations and lower the reliability threat.

However, even though the procedure of generating ADICO statements based on institutional grammar is straightforward, the process of translating this into code is not. There is currently not enough evidence to produce an automated compiler that transpiles these statements into valid code. Transpiling is a specific term for taking source code written in one language and transforming it into another language with a similar abstraction level (Nunnari and Heloir, 2018). Therefore, the mapping between ADICO statements and conditional code may vary between researchers. One can hypothesize that automating ADICO statements based on natural language and transpiling such statements to conditional code would mitigate this threat. However, there are not enough facts to produce a conclusive statement, but this was to be expected due to the study's exploratory nature.

8.4 Future Work

A first barrier to overcome is manual effort activities to determine system requirements and functional requirements for the smart contract. This manual effort could potentially be automated through natural language processing by applying the concepts of institutional grammar. Through natural language processing, this process could potentially be automated. **Natural Language Processing** is a subtopic of natural language processing (NLP) in artificial intelligence that deals with machine reading comprehension (Dalpiaz et al., 2018). NLP could assist by automatically translating system requirements into ADICO statements (Tripathy, Agrawal, and Rath, 2014). If this linguistic structure proves applicable between smart contracts and agent-based modeling requirements, then automatically deriving such features is quite straightforward. The effort to transform the statements would be worth it. Finally, deriving ADICO statements from the requirements facilitates traceability since the terms and concepts used in the requirements can be easily linked to the statements.

A second barrier to overcome is lowering the barrier of compiling Java code and Solidity code and vice versa. Automatic code generation would greatly improve traceability, automation, and validity. **Automatic Code Generation.** could lead to a method that automatically converts smart contract code into Java code which can be used to simulate the behaviors of that smart contract. In computer science, the term automatic programming identifies a type of computer programming. Some mechanism generates a computer program to allow human programmers to write the code at a higher abstraction level (Mur, 2006). Automation of the manual process referred to translation of high-level programming languages can be identified as a compiler (Parnas, 1985). To be more concrete, one could explore the area of transpiling. A Transpiler is a type of translator that takes the source code of a program written in a programming language as its input and produces an equivalent source code in the same or a different programming language (Nunnari and Heloir, 2018).

One direction is the generation of validation code based on a more detailed intermediary representation of ADICO rules and advanced Solidity-specific concepts.

The most exciting aspect of future work could be the ability to achieve automatic translation of smart contracts. Is it possible for humans or automated entities to verify a smart contract's contractual semantics and responsibilities? Is it possible to produce ADICO-style institutional statements using just the EVM machine code rather than a high-level language like Solidity? Addressing the aforementioned questions might clarify whether institutional statements could serve as a vehicle to determine coordination mechanisms for collective adaptive systems, regardless of whether smart contracts are encoded by humans or, eventually, by machines.

Chapter 9

Conclusion

This research aims to identify if smart contract design can be supported through agent-based modeling by simulating systems requirements. We found that agent-based models and smart contracts share high-level system properties that arise from the lower-level subsystem interactions through systematic literature reviews. In other words, basic behavior or rules create complicated behaviors or state changes to the whole system level (Bonabeau, 2002). We learned that institutional grammar is used to generate ADICO statements from which conditions can be derived on both an operational and a collective level (Frantz and Nowostawski, 2016).

> This study concludes that ADICO statements can bridge the gap between agent-based modeling and smart contract design by serving as a common source pool from which decision-making policies are derived on both operational and collective levels.

This approach is proven to have potential, which can be seen in the paper "from Institutions to Code: Towards Automated Smart Contract Generation." (Crawford and Ostrom, 1995). Also Smajgl, Izquierdo, and Huigen (2008) proposes a sequence where one could model endogenous institutional rule change for agents. This is done by identifying the structural components of a general rule from a modeling perspective, followed by providing an agent architecture overview. Parts of the proposed sequence are implemented in a NetLogo model. Finally, Ghorbani and Bravo (2016) confirms that institutions significantly contribute to the sustainable management of common-pool resource systems.

This study cannot claim that the output agent-based simulations provide concrete findings to adapt decision-making policies on smart contracts automatically. This research did not generate and deploy smart contracts based on a shared code base with the agent-based simulation. Secondly, a smart contract's influence should be evaluated against historical data by comparing the simulation output with the output of smart contract data. Unfortunately, this data could not be acquired, which forms a threat to the model's validity. We learned that ADICO is not a framework nor an automated method for designing smart contracts. ADICO can not design agents and their rules for an agent-based model, but it can serve as a common source pool from which condition can be derived on operational and collective level (Frantz and Nowostawski, 2016).

> Hence, we conclude that agent-based modeling is a foundational tool for designing smart contracts in new DLT based communities.

The aforementioned conclusions came to stand by following the Design Science Research approach presented by R. J. Wieringa (2014) in which the research objective is answered through a series of sub-questions.

In a first step, this study defines smart contracts and agent-based modeling based on prior literature. A smart contract is defined by the US National Institute of Standards and Technology as a collection of code and data deployed using cryptographically signed transactions on the blockchain network (Yaga et al., 2019). Secondly, an agent-based model (ABM) is a class of computational models for simulating the actions and interactions of agents to assess their effects on the system as a whole (Miller and Page, 2009). Furthermore, an agent-based model is a computerized simulation of many decision-makers (agents) and institutions that interact through prescribed rules.
This study contributes by visualizing how these commonalities relate to each other, as seen in Figure 3.7.

This study analyzes What research into applying agent-based modeling to smart contract development has already been conducted concerning the second research question. Despite a broad range of interesting research in the domain of agent-based modeling and smart contract design, there deems little concrete research that brings these domains together. The research of Kolvart, Poola, and Rull (2016) mentions both the ABM and smart contract domain, but the research emphasizes the sociological interactions between such systems. An interesting study by Boogaard, 2018 makes use of a model-driven approach to develop smart contracts. While Boogaard (2018) mentions ADICO as a possible model-driven approach to design smart contracts, the study in question used a state machine approach instead and did not dig deeper into the topic of agent-based modeling. Further studies which investigate both the domain of ABM and smart contracts such as the work of Frantz and Nowostawski (2016) or Frantz and Nowostawski (2016) do not dedicate enough attention between linking agent-based modeling and smart contracts. Lucassen et al. (2016) argues about a lack of requirements understanding as well as domain information transformation, which is an important factor in agent-based modeling as well as smart contract design. This argument can be confirmed by Pressman (2005) who states that previous work in software development shares the same concerns as smart contract design. That is a lack of understanding of functional requirements and a transformation of domain information.
However, the work described in this study adds value in various ways to the knowledge-based of agent-based modeling and smart contracts. Firstly, this study provides a holistic description of the principles of agent-based modeling and smart contracts, and it provides an overview of the current state of research in these domains. Secondly, the work on ABM and smart contracts is only partially embedded in research. However, for the most part, it is conducted in an online open-source environment without a need for detailed documentation (Pressman, 2005). This study's scientific approach adds value to the current state stat of smart contract design and agent-based modeling.

In response to the third research question, we found that higher-level system properties arise from lower-level subsystem interactions for both agent-based models and smart contracts. Basic behaviors or rules create complicated behaviors. Hence, agent-based modeling and smart contracts development requirements are consolidated, and commonalities found between smart contracts and agent-based modeling are modeled through a concept diagram and visualized in Figure 3.7.

Through the fourth research question, we conclude that various agent-based modeling and simulation programs exist. We concluded that important evaluation criteria are domain, scalability, agent type, programming language, development effort, and licensing.
This study contributes to the agent-based modeling community by creating a repository that

lists this data set of tools and their metadata. Furthermore, as seen in Figure C.1 in Appendix C shows a website that has been deployed to allow for easy searching based on features. Access to the data is granted, free of charge, to any person obtaining a copy of the data without restriction to contribute to the agent-based modeling domain.

Finally, the study encompasses traits to believe that ABM assists in gathering smart contract development. Institutional grammar can bridge the gap between designing agents for an agent-based model and designing smart contracts starting from collecting ADICO statements that correspond to the smart contract requirements. This means that insights are gathered through agent-based simulation by identifying the structural components of a smart contract. Figure B.2 in Appendix B illustrates how a smart contract definition influences the simulation outcome based on the rule-based statements defined through a case study.

> The research presented provides the following contributions. Firstly, through literature review, we find that agent-based modeling concepts and smart contracts concepts heavily depend on a set of rules translated from functional requirements. Secondly, this study shows how different domain concepts regarding agent-based modeling and smart contract design can be mapped, as seen in Figure 3.7. Thirdly, abstraction of requirements through institutional grammar displays how collective ADICO statements are created from functional requirements to be used at the operational level. Furthermore, figure B.2, in Appendix B illustrates a method for gathering insights from agent-based simulations which can guide smart contract development. Additionally, this study generates open source software that simulates a decentralized system as an agent-based model through Repast Symphony. The model gives insight into the sustainability of a P2P-like network. In addition, the model gives insight into the effect of ratios of nodes connecting to the network. The aforementioned insights could help the owner of the network to mitigate imbalance when composing smart contracts. The ABM software serves as a good starting point for evaluating decentralized networks on which can build further. Second, to last, to allow the ABM community to benefit from this research, a repository with ABM available ABM tools is created on Github. Furthermore, as seen in Figure C.1 in Appendix C shows a website that has been deployed to allow for easy searching based on features. Lastly, This study contributes by performing a case study for SearchSECO by gathering insights on the sustainability of the SearchSECO network. A valuable scientifically written overview of the current state of smart contract design is presented through this case study.

The aforementioned contributions display that progress was made in smart contract development by analyzing the possibilities with agent-based modeling. However, not all results are fully conclusive. The activities to determine system requirements and functional requirements for the smart contract are currently manual efforts. Through natural language processing by applying the concepts of institutional grammar, manual labor could be mitigated. Furthermore, the models' validation would benefit from historical data as well as expert interviews. Finally, a method to lower the barrier of compiling Java code and Solidity code and vice versa would greatly improve traceability, automation, and validity.

Acknowledgements

First and foremost, I want to thank Dr. Slinger Jansen and Dr. Frank Dignum for their guidance, feedback, comments and discuss possibilities. I want to thank Dr. Slinger Jansen, who allowed me to do this fascinating project. Secondly, a special thanks to the SecureSECO project. The SecureSECO project collaborates between five companies and five universities with over 10 researchers and tool developers who collaboratively contribute to the vision of a safer and more secure worldwide software ecosystem. Academic research is performed, participation in hackathons, and academic research data providence for other research groups about software. Lastly, I would like to thank my family, particularly Dorna Alvandi, who helped me a lot with the mental burden of dealing with the issues during these COVID-19 times.

Appendix A

Structure of a smart contract

SOURCE CODE A.1: Structure of a Solidity source file which serves to provide a quick overview of State Variables, Functions, Function Modifiers, Events, Struct Types and Enum Types declarations

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.4.0 <0.9.0;

contract SimpleStorage {
    //State Variables
    uint storedData;

    // Function
    function bid() public payable {
        // ...
    }

    // Modifiers
    address public seller;
    modifier onlySeller() { // Modifier
        require(
            msg.sender == seller,
            "Only seller can call this."
        );
        _;
    }
    function abort() public view onlySeller { // Modifier usage
        // ...
    }

    //Events
    event HighestBidIncreased(address bidder, uint amount); // Event
    function bid() public payable {
        // ...
        emit HighestBidIncreased(msg.sender, msg.value); // Triggering
         ↪  event
    }

    // Structs
     struct Voter { // Struct
        uint weight;
        bool voted;
        address delegate;
        uint vote;
    }

    //Enums
    //enum State { Created, Locked, Inactive } // Enum
}
```

SOURCE CODE A.2: Scheduled Method - Connecting, uploading and updating network balance code snippets

```
        @ScheduledMethod(start = 1, interval = 2)
        public void tick() {
                Context<Object> context = ContextUtils.getContext(this);
                Network<Object> network = (Network<Object>)
                 ↪  context.getProjection("searchNetwork");
                if (config.UploadDistribution.getDecision()) {
                        upload(network, context);
                }
                if (config.RequestDistribution.getDecision()) {
                        download(network, context);
                }
        }

    public void upload(Network<Object> network, Context<Object> context) {
                searchAgent = (SearchSECOAgent)
                 ↪  context.getRandomObjects(SearchSECOAgent.class,
                 ↪  1).iterator().next();
                network.addEdge(searchAgent, this, 1);
                updateBalance(true, searchAgent,context);
        }

        public void download(Network<Object> network, Context<Object>
         ↪  context) {
                searchAgent = (SearchSECOAgent)
                 ↪  context.getRandomObjects(SearchSECOAgent.class,
                 ↪  1).iterator().next();
                network.addEdge(this, searchAgent, 2);
                updateBalance(false, searchAgent,context);
        }

        public void updateBalance(boolean upload, SearchSECOAgent
         ↪  searchAgent,Context<Object> context) {
                if(upload) {
                        credits += 1;
                        searchAgent.credits -= 1;
                }
                else {
                        if(credits > 0)
                        {
                                credits -= 1;
                                searchAgent.credits += 1;
                        }
                        else if(credits == 0) {
                                if(money > 0) {
                                        money -= 1;
                                        searchAgent.money +=1;
                                }else {
                                        context.remove(this);
                                }
                        }
                }
        }
```

Appendix B

Process Deliverable Diagrams

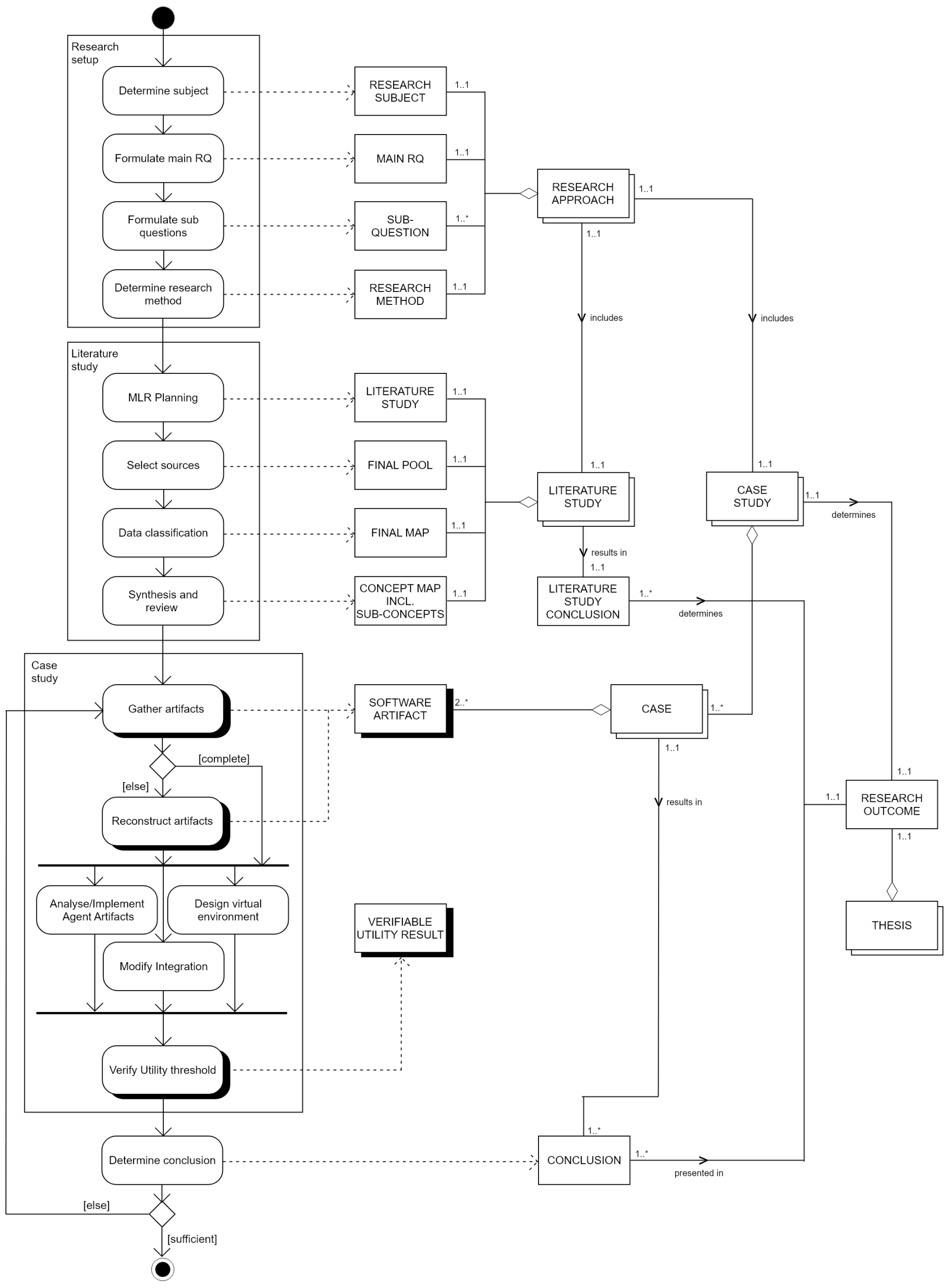


FIGURE B.1: Visualization of the research approach using a PDD. The process view is displayed on the left-hand side based on a UML activity diagram, where the right right-hand side of the diagram displays the deliverables based on a UML class diagram (Weerd and S. Brinkkemper, 2009).

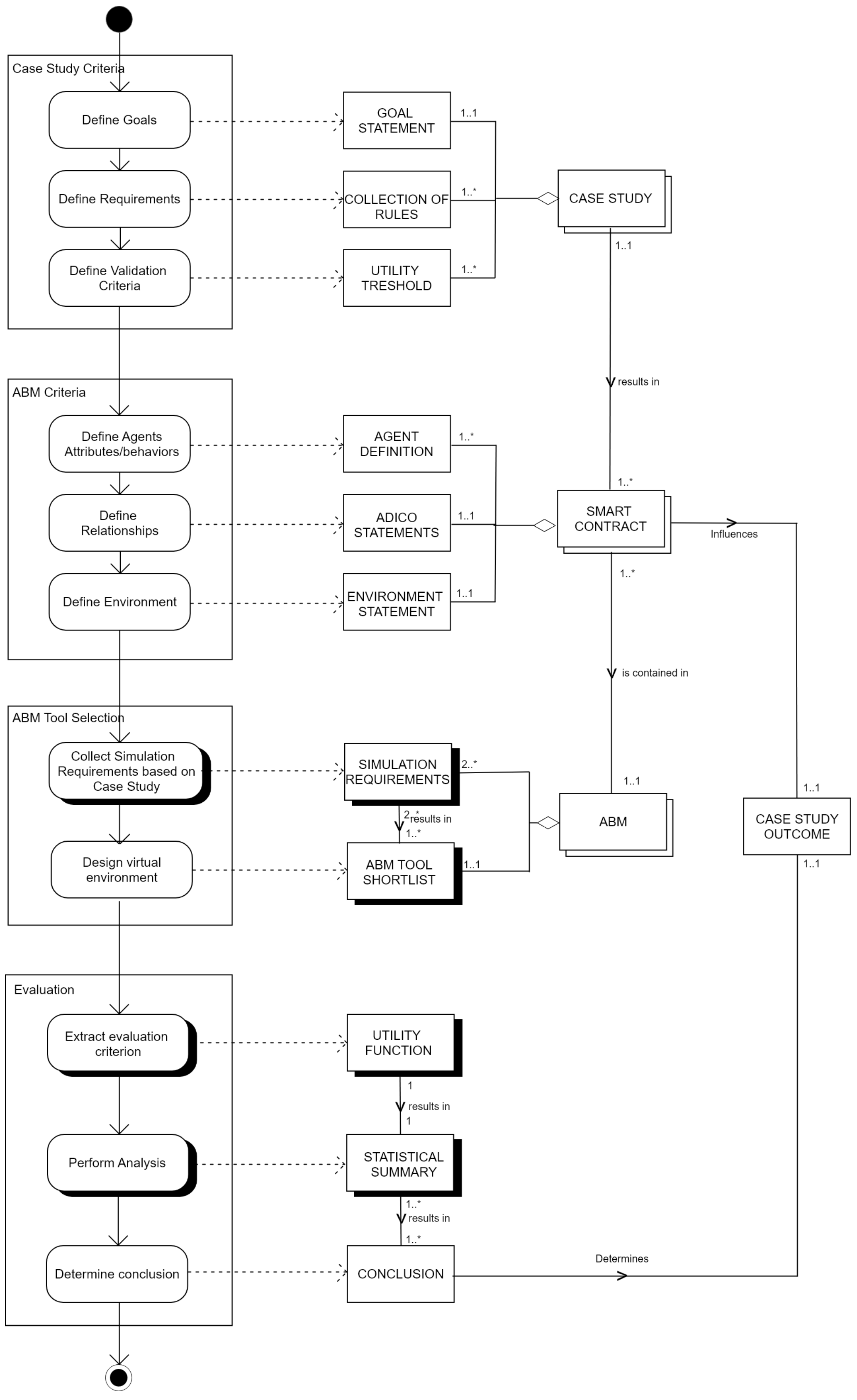
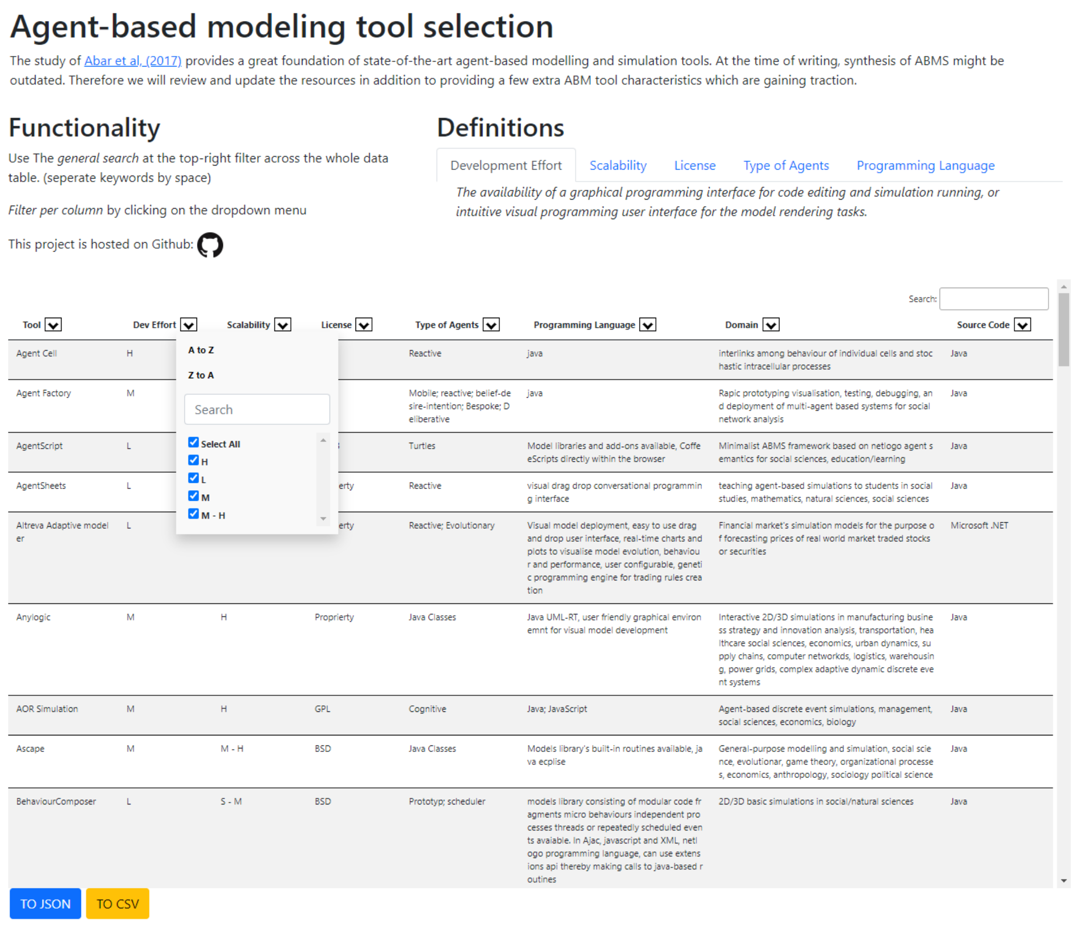


FIGURE B.2: The figure illustrates how the smart contract definition influences the outcome of the simulation based on the rule-based statements defined in the case study.

Appendix C

ABM Tools

For more information about ABM Tool Characteristics and definitions, see Chapter 4.



Agent-based modeling tool selection

The study of Abar et al. (2017) provides a great foundation of state-of-the-art agent-based modelling and simulation tools. At the time of writing, synthesis of ABMS might be outdated. Therefore we will review and update the resources in addition to providing a few extra ABM tool characteristics which are gaining traction.

Functionality

Use The *general search* at the top-right filter across the whole data table. (seperate keywords by space)

Filter per column by clicking on the dropdown menu

This project is hosted on Github:

Definitions

Development Effort | Scalability | License | Type of Agents | Programming Language

The availability of a graphical programming interface for code editing and simulation running, or intuitive visual programming user interface for the model rendering tasks.

Search:

Tool	Dev Effort	Scalability	License	Type of Agents	Programming Language	Domain	Source Code
Agent Cell	H			Reactive	java	interlinks among behaviour of individual cells and stochastic intracellular processes	Java
Agent Factory	M			Mobile; reactive; belief-desire-intention; Bespoke; Deliberative	java	Rapic prototyping visualisation, testing, debugging, and deployment of multi-agent based systems for social network analysis	Java
AgentScript	L			Turtles	Model libraries and add-ons available, CoffeeScripts directly within the browser	Minimalist ABMS framework based on netlogo agent semantics for social sciences, education/learning	Java
AgentSheets	L		erty	Reactive	visual drag drop conversational programming interface	teaching agent-based simulations to students in social studies, mathematics, natural sciences, social sciences	Java
Altreva Adaptive modeler	L		erty	Reactive; Evolutionary	Visual model deployment, easy to use drag and drop user interface, real-time charts and plots to visualise model evolution, behaviour and performance, user configurable, genetic programming engine for trading rules creation	Financial market's simulation models for the purpose of forecasting prices of real world market traded stocks or securities	Microsoft .NET
Anylogic	M	H	Proprierty	Java Classes	Java UML-RT, user friendly graphical environemnt for visual model development	Interactive 2D/3D simulations in manufacturing business strategy and innovation analysis, transportation, healthcare social sciences, economics, urban dynamics, supply chains, computer networkds, logistics, warehousing, power grids, complex adaptive dynamic discrete event systems	Java
AOR Simulation	M	H	GPL	Cognitive	Java; JavaScript	Agent-based discrete event simulations, management, social sciences, economics, biology	Java
Ascape	M	M - H	BSD	Java Classes	Models library's built-in routines available, java ecplise	General-purpose modelling and simulation, social science, evolutionar, game theory, organizational processes, economics, anthropology, sociology political science	Java
BehaviourComposer	L	S - M	BSD	Prototyp; scheduler	models library consisting of modular code fragments micro behaviours independent processes threads or repeatedly scheduled events avaiable. In Ajac, javascript and XML, netlogo programming language, can use extensions api thereby making calls to java-based routines	2D/3D basic simulations in social/natural sciences	Java

FIGURE C.1: https://peirstom.github.io/

TABLE C.1: Question Guideline used to formulate requirements of the case study.

Tool	Dev Effort	Scalability	License	Type of Agents	Programming Language	Domain
Agent Cell	H	H	GPL	Reactive	java	interlinks among behaviour of individual cells and stochastic intracellular processes
Agent Factory	M	M	LGPL	Mobile; reactive; belief-desire-intention; Bespoke; Deliberative	java	Rapic prototyping visualisation, testing, debugging, and deployment of multi-agent based systems for social network analysis
AgentScript	L	S	GPLV3	Turtles	Model libraries and add-ons available, CoffeeScripts directly within the browser	Minimalist ABMS framework based on netlogo agent semantics for social sciences, education/learning
AgentSheets	L	S - M	Proprierty	Reactive	visual drag drop conversational programming interface	teaching agent-based simulations to students in social studies, mathematics, natural sciences, social sciences
Altreva Adaptive modeler	L	H	Proprierty	Reactive; Evolutionary	Visual model deployment, easy to use drag and drop user interface, real-time charts and plots to visualise model evolution, behaviour and performance, user configurable, genetic programming engine for trading rules creation	Financial market's simulation models for the purpose of forecasting prices of real world market traded stocks or securities
Anylogic	M	H	Proprierty	Java Classes	Java UML-RT, user friendly graphical environemnt for visual model development	Interactive 2D/3D simulations in manufacturing business strategy and innovation analysis, transportation, healthcare social sciences, economics, urban dynamics, supply chains, computer networkds, logistics, warehousing, power grids, complex adaptive dynamic discrete event systems
AOR Simulation	M	H	GPL	Cognitive	Java; JavaScript	Agent-based discrete event simulations, management, social sciences, economics, biology
Ascape	M	M - H	BSD	Java Classes	Models library's built-in routines available, java ecplise	General-purpose modelling and simulation, social science, evolutionar, game theory, organizational processes, economics, anthropology, sociology political science
BehaviourComposer	L	S - M	BSD	Prototyp; scheduler	models library consisting of modular code fragments micro behaviours independent processes threads or repeatedly scheduled events avaiable. In Ajac, javascript and XML, netlogo programming language, can use extensions api thereby making calls to java-based routines	2D/3D basic simulations in social/natural sciences
Brahms	H	H	Open Source	Cognitive, Belief-Desire Intention	BDI Language	Multi-agent based system to model and simulate people's activity and situated behaviour (location, artifacts, communication) and collaboration/coordination in organizational processes
Breve	M	M	GPL	Stimulus responsive simple mobile agents	Python	Builds 3D simulations of agent-based systems representing artifical life
Bsim	H	H	OSI-MIT	Reactive Behavioural agents	Java	Simulations in 2D/3D related to: stohastic interactions of bacterial populations and particles in a fluids, multi-cellular computing in synthetic bioloigy
CloudSim	M	H	GNU Lesser GPL	Logic of agent/objects implemented as policies for cloud, provisioning, scheduling, migration	Java; IDE eclipse	Modelling and simulation of cloud computing / virtualised datacentre based infrastructures and services
Cormas	M	M	Open Source	Agents/objects implemented as class constructs	SmallTalk	Simulations of natural renewable resource management, geographic information systems, marketing, ecology
CRAFTY	M	M - H	Open Source	Configurable spatical data structures as objects (cells/regions) with different production functions in thresholds, component role-based agents	R	Simulations of a wide range of land uses and logistics (goods/services)
CybelePro	M	H	Proprierty	Reactive agent/objects implemented as java classes	Java-based cybelePro API	Modelling and simulations of high performance infrastructures and large-scale distributed systems (such as robotics, planning and scheduling, communication network systems and cross-enterprises, data-mining, control of air and ground transportation systems)
D-Omar	H	H	Proprierty	Cognitive agents	SCORE (a procedural language)	Simulation development environment designed to explore and model human multiple task-based behaviours (air-crew air-traffic control and communication)
DigiHive	H	M	Proprierty	Evolutionary	Prolog	Simulations of artificial life, emergent phenomena, self-adaption, self-replication
Echo	H	H	Open Source	Adaptive Evolutionary agents	C	Simulation environments for complex adaptive systems, ecological modelling

Table C.1 – *Continued on next page*

Table C.1 – *Continued from previous page*

Tool	Dev Effort	Scalability	License	Type of Agents	Programming Language	Domain
EcoLab	H	H	Open Source	MPI-based agents implemented as C++ classes	C++	Simulations of complex dynamics of evolution
Envision	M	M	Open Source	Reactive Behavioural agents	C++	Multi-paradigm GIS (Geographic Information System based tool for analysing scenario-based coupled natural/human resource systems and community/regional integrated planning and environmental assessments
Eve	M	S - M	Open Source	Agents/objects implemented as Java classes	Java	General multi-purpose modelling and simulations
ExtendSim	M	S - M	Proprierty	Object oeriented agents or entities interacting via discrete events	ModL	2D/3D simulations in a variety of fields (business, industry, healthcare, meteorology, air defence, and academic)
FLAME	M	H	GNU Lesser GPL	Agents as objects characterized by states functions and sets of variables	GUI	General multi-purpose simulations (cellular automata, economics, biology, medical, traffic, situations)
FLAME GPU	H	H	GPU	Reactive processing agents as communicating X-Machines with inputs and outputs	CUDA	3D simulations for emergent complex behaviours in biology/medical domains (tissue cultures and signaling pathways) with multi massive amounts of agents on GPU
FlexSim	L	S - M	Proprierty	Agents/objects implementd as C++ classes	FlexSim's library of standard customizable objects available	2D/3D simulations for manufacturing, production, distribution of logistics, supply chains, transportation, oil field or mining process, networking data flow, healthcare optimizations with OptQuest plugin
Framsticks	L	S	GPL LGPL	Evolutionary	Java Runtime Environments	2D/3D simulations of evolving agent-based systems and articicla life for research and education
Gama	M	M - H	GNU Gplv2	Reactive Behavioural agents	GAML	2D/3D modelling and development platform for building spatially explicit agent0based simulations (arbitrary complex GIS data as the bases of agents), land-use and land-planning, social instituational, economical ecological or biophysical systems
GALATEA	M	M	Open Source	Logic-based agents or objects (as observe-reason-act-processes)	Java	Modeling and simulatino of discrete -event simulation, continuous, combined and multi agent systems e.g. hardware and software co-design, communication, systes, manufacturing systems, biology, sociology, economics
GridABM	H	H	GPL	Reactice, BDI agents	Java	High-performance agent-based distributed cellular automation models based on repast
GROWLab	M	S - M	Open Source	Agents/objects implemented as Java classes	Java	Social phenomena
HLA_Agent	H	H	Open Source	Reactive, Deliberative, Cognitive agents	C++	Complec adaptive systems, evolutionary computation, social and naturaal sciences, mapping passenger flow, manufacturing military combat scenarios, high performance computing
HLA_RePast	H	H	Open Source	Reactive/BDI object oeriented agents	Java	Cellular automata, complex adaptive ystems, evolutionary computation, social and natural sciences, mapping passenger flow, manufacturing military combat scenarios, high performance computing
IDEA	M	M	Open Source	Reactuve, proactive, deliberative agents	Java IDE	Applied sciences (knowledge based systems, multimedia, micro-grid operation, distributed network management, ubiquitous care-support or assisted services, information retrieval)
InsightMaker	M	S - M	GPLV3	Reactive Behavioural agents	JavaScript	Differntial equation or dynamical systems modelling, such as spatially-aware model of infectious disease spread
JAMEL	L	S	GPL	Reactive	Java	Building agent0based macroeconomic simulations
JAMSIM	M	M	Open Source	Agents composed as a scapes hierarchy or collectino of agents that represent basic building blocks such as an offspring parent, or household nit	Casper model implementation in Java	Dynamic discrete-time policy-oriented microsimulation government projects (taxation/pensions)
Janus	M	M	GPLv3	Recursive holon agent (composed of a set of structures + Various agents as extra-modules)	Groovy, Javascript, Ruby, Python	General purpose platform with organizational and holonic agent0based simulation layers
JAS	L	M	GNU LGPL	Agents/objects implemented as Java classes	Java	Simulations of dynamic social systems, genetic algorithms and neural networks
JASA	M	M	GPLv2	Adaptive trading agents	Java	Simulations of computational economics
JAS-mine	M	M	Open Source	Agents/objects implemented as Java classes	Java	Simulations of data-driven models, discrete-event/continuous simulation, dynamic microsimulations of specific processes (aging, educational choices, labor market events, houshold fomrations), statistical analysis

Table C.1 – *Continued on next page*

Table C.1 – *Continued from previous page*

Tool	Dev Effort	Scalability	License	Type of Agents	Programming Language	Domain
JCASim	L	S	Proprierty	Interactive Reactive agents	Java	Microscopic cellular automata simulated in different lattices 1D/2D/3D
jES	L	S	Open Source	Objects or decision nodes based on independent piece of code or action rules and algorithms to represent agents as avatars of actual people	Java	Simulations in the context of enterprise behaviour and activities
LSD	M	H	GPL	Agents/objects implemented as C++ classes	C++ LSD language in Eclipse	Economic and social science simulations; can generate multiple formats for graphs: time series, cross-section, 2D and 3D scatter plots, frequency histograms
MACSimJX	M	M	Proprierty	JADEs (Java Agent Development framework and FIPA-complian agent classes)	C/C++ or Java	Modelling and simulation environment integrated with Matlab-Simulink for developing dynamic, embedded, decentralised control systems. E.g. aircraft Boeing 747 sensor unit's flight dynamics and kinematics undergoing a complex series of manouveres using JADA (an environment for developing agents)
MASON	H	M-H	Open Source	Agents/objects implemented as Java classes	Java	General multi-purpose 2D/3D simulations (social complexity, physical and abstract modelling, artificial intellgince, robotics and machine learning)
MASS	M	H	Proprierty	Agents/objects implemented as Java classes	Java	General purpose distributed simulations (complex social economic system, traffic situations)
MASyV	H	M	Open Source	Agents/objects inplemented as C class constructs	Client simulations in C language trough GUY representatino of client simulations	2D/3D visualisations of cellular automate
Mathmatica (Wolfram)	M	M	Proprierty	Agents/objects implemented as class constructs	Wolfram multi paradigm programming language and its interface with C/C++/Java	Simulation of social and behavioural sciences, customer movements in a store, complex adaptive social systems or articial societies.
MATSim	H	E	GPL	Agents/objects implemented as Java classes	Java	Simulations of transport mobility systems and Geographical Informatino Systems (GIS) Based evacuation scenarios
MESA	M	S - M	Apache 2.0	Agents as class constructs (having a unique identifier consisting of variable and actions)	Python	General purpose artifical life related simulations (Basically, Meas is a Python 3 based alternative to Netlogo, Repast, or MASON)
Mimosa	M	M	LGPLv2, CIRAD	Agents/objects implemented as java classes	Java	Building conceptual models for running the economic, ecological, social simulations
MIMOSE	M	M	GPL	Agents/objects implemented as java classes	A model description language GUI	Simulation in social sciences, epidemiology, eductation/research
MOBIDYC	L	S	GPL	Agents/objects implemented as class constructs	Smalltalk	Academic simulations in ecology, cellular automata, biology, and environment
Mobility Testbed	M	M	GPL	Agents/objects implemented as class constructs	Java	Simulations for transportation networks, activities, life-cycles and mobility. Multimodal route and journey planning services
Modgen	M	M	Proprierty	Agents/objects implemented as linked actors having specific characteristics as states/events	C++ Modgen Language	Dynamic social science microsimulations for the socio-economic and demographic development of societies
NETLOGO	L	M	GPL	Active objects with simple goals implemented as mobile agents (tutles, patches, links and the observer)	NetLogo Language	2D/3D simulations in social and natural sciences, teaching/research
OBEUS	M	M	Proprierty	Agents/objects implemented as C++ classes	Microsoft.NET languages C#, C++, Visual Basic	Simulations of Geographic Automata Systems (GAS) or urba or regional planning areas.
Pandora	M	H	Proprierty	Agents/objects implemented as C++/Python classes	Python or C++	High-performance computing environments, GIS support, social and environmental phenomena archeaology
PDES-MAS	H	E	Open Source	Situated agents as MPI-based agent Logical Processes ALPs implemented as C++ classes	C++	Complex adaptive systems, social and natural sciences, mapping passenger flow, manufacturing military combat scenarios, high performance computing
PedSim	L	S	GPL	Agents/objects implemented as C++ classes	C++	Simulation system and library for microscopic pedestrian crowd mapping
PS-I	L	S	GPLv2	Agents/objects implemented as class constructs	Models Library available, declarative model specification language; TCL/TK scripts to apply graphic effects	Simulations of political phenomena, cultural, psychological, administrative, geographical, and other social factors
Repast-J	H	H	BSD	Reactive/BDI object-oriented agents	Java; C# managed C++ Lisp Prolog Visual Basic .NET python	Simulations of social networks and integrated support for GIS, genetic algorithms
Repast HPC	M - H	E	BSD	Reactive/BDI object oeriented agents	Standard or Logo-styel C++	Simulations incomputational social sciences, cellular automata, complex adaptive systems.

Table C.1 – *Continued on next page*

Table C.1 – *Continued from previous page*

Tool	Dev Effort	Scalability	License	Type of Agents	Programming Language	Domain
Repast Simphony	H	H	BSD	Reactive/BDI object-oriented agents	Java, Groovy, ReLogo (Repasts's NetLogo-like language)	2D/3D simulations in social sciences, consumer products, supply chains, GIS, cellular automata, complex adaptive systems
Scratch	L	S	GPLv2, CC-BY-SA 2.0 License	Sprites or objects encapsulating state and behaviour	Programming scripts made by snapping graphical chunks/blocks in the form of stacks	Self-directed creative learning software for students and educators to make 2D/3D animated games in social sciences, geography, mathematics, linguistics, arts, computer science
SeaS	M	S	EULA	Agents/objects implemented as C++ classes	Tactical Programming Language TPL	2D/3D simulations of complex adaptive systems, military war-fighting scenarios
SeSam	L	H	LGPL	Agents/objects implemented as java classes	Visual Modelling Language	Simulations in social sciences (logistics, production, traffic, passenger flow, healthcare, biology, urban planning), research/teaching
SimAgent	H	H	MIT/XFREE86	Reactive, Deliberative, Cognitive agents	Robust extensible multi-purpose coding format which support programming in Prolog, Common Lisp or standard ML	Simulations in research/education social sciences (biology, psychology) evolutionary computation
SimBioSys	M	M	ALA	Agents/objects implemented as C++ classes	class libreraries available in C++	Evolutionary simulations in biological and social sciences
SimEvents(Matlab)	M	M - H	Proprierty	Collectino of independent object oeriented agents/entities interacting via discrete events	SimEvents with Matlab Simulink and stateflow functions libraries, toolboxes and add-ons in MATLAB coding syntax	Simulations to optimize supply chain processes for manufacturing and operations, forecasting, capacity, planning aerospace, automotive mission plans.
Simio	M	M - H	Proprierty	Agents as objects characterized by properties states and behaviours	Standard object libraries available built-in GUI	2D/3D simulations in advanced predictive analysis tourist flow, manufacturing, military solutions, production, scheduling, transportation, logistics, supply chain, mining, industry, healthcare, maritime/ports airfreight services, optimizations with OptQuest
SimJr	L	S	BSD	Cognitive agents	JavaScript	Simulations of air and ground forces, military scenarios, different aspects of human behaviours or actions modelling, command and control modules intelligence analysis and informatino visualisation.
SimSketch	L	S	Proprierty	Reactive Behavioural agents	Java Web Start (JavaWS)	Educational relevant scientific phenomena for young learnerns
Simul8	M	M - H	Proprierty	Agents as physical logical or activity-based objects	Visual Logic	2D/3D simulations in education healthcare, manufacturing, logistics, contact centre or client-based services, supply chains, capacity planning, administrative workflows, optimization with OptQuest plugin
SOARS	L	S	Open Source	Role-Based Agents	Javascripts	Simulations of social, business, public health and organizational system, GIS, epidemiology
StarLogo	L	S	Clearthrough Software License V1.0	Procedural agents having a range of potential functions	Template wizzard available for creating different types of active objects or turtles with simple goals	Simulation in social and natural sciences, education, for depicting the behaviour of decentralized models (bird flocking, traffic jamming, and ant colony formations)
StarLogo TNG	L	S	Proprierty	Procedural agents having a range of potential functions	StarLogo TNG block based scripting - GUI	3D simulations of educational models and video games
Sugarscape	L	S	GPL	Agents/objects implemented as java classes	Java	Simulations in social sciences, cellular automata, education
Swarm	H	E	GPL	Collection (swarms) of independent object-oriented agents interacting via discrete events	Objective-C; Swarm code; Java	Biological sciences; supply chain optimization and logistics; consumer behaviour with social network effects; distributed computing workforce/traffic/portfolio management
TerraME	M	M	Open Source	Agents/objects implemented as C++/Lua scripts classes	TerraMe's modelling models libraries and API	Multi-paradigm spatial dynamical systems, cellular automata, GIS, terrestrial systems, land change, hydrologic, species dispersion, climate models
UrbanSim	M	H	GNU	Agents/objects implemented as class constructs of Python-based custom domain-specific programming language "Tekoa"	Model librry available on Opus-based GUI	Support planning, analysis and modelling of urban developmnet/land use/ transportation/ housing affordability/ environmentally sensitive habitats and greenhouse gas emissions
VisualBots	L	S	EULA	Agents/objects implemented as class constructs	GUI in Excel VBA	Simulations in social and political science, economics, emergent behaviours, cellular automata, educations and teaching
VSEit	L	M	Proprierty	Agents/objects implemented as java classes	Java	Social sciences (ecology and economy), object-oriented stochastic event-drive simulations, cellular automate) and education and teaching
Xholon	M	M	LGPL	Reactive Behavioural agents	XML/JAVA support for UML; Turtle geometry using an optional NetLogo-like syntax	2D/3D simulations in cellular automata, biology, biochemistry, statistical modelling, controllers and other embedded systems.

Bibliography

Abar, Sameera, Georgios K Theodoropoulos, Pierre Lemarinier, and Gregory MP O'Hare (2017). "Agent Based Modelling and Simulation tools: A review of the state-of-art software". In: *Computer Science Review* 24, pp. 13–33.

Adams, Jean, Frances C Hillier-Brown, Helen J Moore, Amelia A Lake, Vera Araujo-Soares, Martin White, and Carolyn Summerbell (2016). "Searching and synthesising 'grey literature'and 'grey information'in public health: critical reflections on three case studies". In: *Systematic reviews* 5.1, p. 164.

Adams, Mike, Shannon Bearly, David Bills, Sean Foy, Margaret Li, Tim Rains, Micheal Ray, Dan Rogers, Frank Simorjay, Sian Suthers, and Jason Wescott (2014). "An introduction to designing reliable cloud services". In: *Microsoft Corporation*, pp. 1–14.

Afonso, Margarida, Regis Vogel, and Jose Teixeira (2006). "From code centric to model centric software engineering: practical case study of MDD infusion in a systems integration company". In: *Fourth Workshop on Model-Based Development of Computer-Based Systems and Third International Workshop on Model-Based Methodologies for Pervasive and Embedded Software (MBD-MOMPES'06)*. IEEE, 10–pp.

Alharby, Maher and Aad Van Moorsel (2017). "Blockchain-based smart contracts: A systematic mapping study". In: *arXiv preprint arXiv:1710.06372*.

Arthur, W Brian (2018). *The economy as an evolving complex system II*. CRC Press.

Axtell, Robert (2000). "Why agents?: on the varied motivations for agent computing in the social sciences". In:

Axtell, Robert L, Clinton J Andrews, and Mitchell J Small (2003). "Agent-based models of industrial ecosystems". In: *Rutgers University, October* 6.

Bagni, Raul, Roberto Berchi, and Pasquale Cariello (2002). "A comparison of simulation models applied to epidemics". In: *Journal of Artificial Societies and Social Simulation* 5.3.

Boehm, Barry, Bradford Clark, Ellis Horowitz, Chris Westland, Ray Madachy, and Richard Selby (1995). "Cost models for future software life cycle processes: COCOMO 2.0". In: *Annals of software engineering* 1.1, pp. 57–94.

Bonabeau, Eric (2002). "Agent-based modeling: Methods and techniques for simulating human systems". In: *Proceedings of the national academy of sciences* 99.suppl 3, pp. 7280–7287.

Borshchev, Andrei, Sally Brailsford, Leonid Churilov, and Brian Dangerfield (2014). "Multi-method modelling: AnyLogic". In: *Discrete-event simulation and system dynamics for management decision making*, pp. 248–279.

Braun, Virginia and Victoria Clarke (2012). "Thematic analysis". In: *APA handbook of research methods in psychology, Vol 2: Research designs: Quantitative, qualitative, neuropsychological, and biological.* 2, pp. 57–71.

Brinkkemper, Sjaak (1996). "Method engineering: engineering of information systems development methods and tools". In: *Information and software technology* 38.4, pp. 275–280.

Brown, D and H Geist (2006). "The earth's changing land: An encyclopedia of land-use and land-cover change". In: *CT: Greenwood Publishing Group Westport*.

Buterin, Vitalik et al. (2014). "A next-generation smart contract and decentralized application platform". In: *white paper* 3.37.

Cachin, Christian et al. (2016). "Architecture of the hyperledger blockchain fabric". In: *Workshop on distributed cryptocurrencies and consensus ledgers*. Vol. 310. 4.

Cannarsa, Michel (2018). "Interpretation of Contracts and Smart Contracts: Smart Interpretation or Interpretation of Smart Contracts?" In: *European Review of Private Law* 26.6.

Carley, Kathleen M, Douglas B Fridsma, Elizabeth Casman, Alex Yahja, Neal Altman, Li-Chiou Chen, Boris Kaminsky, and Démian Nave (2006). "BioWar: scalable agent-based model of bioattacks". In: *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans* 36.2, pp. 252–265.

Clack, Christopher D, Vikram A Bakshi, and Lee Braine (2016). "Smart contract templates: foundations, design landscape and research directions". In: *arXiv preprint arXiv:1608.00771*.

Crawford, Sue ES and Elinor Ostrom (1995). "A grammar of institutions". In: *American political science review*, pp. 582–600.

Crooks, Andrew T and Christian JE Castle (2012). "The integration of agent-based modelling and geographical information for geospatial simulation". In: *Agent-based models of geographical systems*. Springer, pp. 219–251.

Cruzes, Daniela S and Tore Dyba (2011). "Recommended steps for thematic synthesis in software engineering". In: *2011 international symposium on empirical software engineering and measurement*. IEEE, pp. 275–284.

Daian, Phil (2016). "Analysis of the DAO exploit". In: *Hacking, Distributed* 6.

Dalpiaz, Fabiano, Alessio Ferrari, Xavier Franch, and Cristina Palomares (2018). "Natural language processing for requirements engineering: The best is yet to come". In: *IEEE software* 35.5, pp. 115–119.

Decker, Christian, Jochen Seidel, and Roger Wattenhofer (2016). "Bitcoin meets strong consistency". In: *Proceedings of the 17th International Conference on Distributed Computing and Networking*, pp. 1–10.

Delmolino, Kevin, Mitchell Arnett, Ahmed Kosba, Andrew Miller, and Elaine Shi (2016). "Step by step towards creating a safe smart contract: Lessons and insights from a cryptocurrency lab". In: *International conference on financial cryptography and data security*. Springer, pp. 79–94.

Dignum, F, J-J Ch Meyer, and R Wieringa (1994). "A dynamic logic for reasoning about sub-ideal states". In: *ECAI Workshop on Artificial Normative Reasoning*. Citeseer, pp. 79–92.

Dignum, Frank and Rosaria Conte (1997). "Intentional agents and goal formation". In: *International Workshop on Agent Theories, Architectures, and Languages*. Springer, pp. 231–243.

Dunham, Jill Bigley (2005). "An agent-based spatially explicit epidemiological model in MASON". In: *Journal of Artificial Societies and Social Simulation* 9.1.

Durieux, Thomas, João F Ferreira, Rui Abreu, and Pedro Cruz (2020). "Empirical review of automated analysis tools on 47,587 Ethereum smart contracts". In: *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, pp. 530–541.

Farshidi, Siamak, Slinger Jansen, Sergio España, and Jacco Verkleij (2020). "Decision support for blockchain platform selection: Three industry case studies". In: *IEEE Transactions on Engineering Management*.

Feist, Josselin, Gustavo Grieco, and Alex Groce (2019). "Slither: a static analysis framework for smart contracts". In: *2019 IEEE/ACM 2nd International Workshop* *on Emerging Trends in Software Engineering for Blockchain (WETSEB)*. IEEE, pp. 8–15.

Filatova, Nataliia (2020). “Smart contracts from the contract law perspective: outlining new regulative strategies”. In: *International Journal of Law and Information Technology* 28.3, pp. 217–242.

Finley, Klint (2016). “A $50 million hack just showed that the DAO was all too human”. In: *Wired https://www. wired. com/2016/06/50-million-hack-just-showed-dao-human/(June 2016)*.

Folcik, Virginia A, Gary C An, and Charles G Orosz (2007). “The Basic Immune Simulator: an agent-based model to study the interactions between innate and adaptive immunity”. In: *Theoretical Biology and Medical Modelling* 4.1, p. 39.

Fox, Geoffrey (2001). “Peer-to-peer networks”. In: *Computing in Science & Engineering* 3.3, pp. 75–77.

Franklin, Stan and Art Graesser (1996). “Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents”. In: *International Workshop on Agent Theories, Architectures, and Languages*. Springer, pp. 21–35.

Frantz, Christopher K and Mariusz Nowostawski (2016). “From institutions to code: Towards automated generation of smart contracts”. In: *2016 IEEE 1st International Workshops on Foundations and Applications of Self* Systems (FAS* W)*. IEEE, pp. 210–215.

Gale, David (1955). “The law of supply and demand”. In: *Mathematica scandinavica*, pp. 155–169.

Garcia, Antonio Prestes and Alfonso Rodriguez-Paton (2016). “Analyzing Repast Symphony models in R with Repast package”. In: *bioRxiv*, p. 047985.

Garousi, Vahid, Michael Felderer, and Mika V Mäntylä (2019). “Guidelines for including grey literature and conducting multivocal literature reviews in software engineering”. In: *Information and Software Technology* 106, pp. 101–121.

Ghorbani, Amineh and Giangiacomo Bravo (2016). “Managing the commons: a simple model of the emergence of institutions through collective action”. In: *International Journal of the Commons* 10.1.

Gilbert, Nigel and Pietro Terna (2000). “How to build and use agent-based models in social science”. In: *Mind & Society* 1.1, pp. 57–72.

Godin, Katelyn, Jackie Stapleton, Sharon I Kirkpatrick, Rhona M Hanning, and Scott T Leatherdale (2015). “Applying systematic review search methods to the grey literature: a case study examining guidelines for school-based breakfast programs in Canada”. In: *Systematic reviews* 4.1, p. 138.

Grimm, Volker, Uta Berger, Finn Bastiansen, Sigrunn Eliassen, Vincent Ginot, Jarl Giske, John Goss-Custard, Tamara Grand, Simone K Heinz, Geir Huse, et al. (2006). “A standard protocol for describing individual-based and agent-based models”. In: *Ecological modelling* 198.1-2, pp. 115–126.

Grishchenko, Ilya, Matteo Maffei, and Clara Schneidewind (2018). “A semantic framework for the security analysis of ethereum smart contracts”. In: *International Conference on Principles of Security and Trust*. Springer, pp. 243–269.

Hall, Andreas and Kirsi Virrantaus (2016). “Visualizing the workings of agent-based models: Diagrams as a tool for communication and knowledge acquisition”. In: *Computers, Environment and Urban Systems* 58, pp. 1–11.

Harmsen, Anton Frank, Jacobus Nicolaas Brinkkemper, and JL Han Oei (1994). *Situational method engineering for information system project approaches*. Citeseer.

Helbing, Dirk (2012). *Social self-organization: Agent-based simulations and experiments to study emergent social behavior*. Springer.

Heppenstall, AJ, AJ Evans, and MH Birkin (2006). "Application of multi-agent systems to modelling a dynamic, locally interacting retail market". In: *Journal of Artificial Societies and Social Simulation* 9.3, p. 2.

Heppenstall, Alison J, Andrew J Evans, and Mark H Birkin (2007). "Genetic algorithm optimisation of an agent-based model for simulating a retail market". In: *Environment and Planning B: Planning and Design* 34.6, pp. 1051–1070.

Hevner, Alan and Samir Chatterjee (2010). *Design Research in Information Systems*. Vol. 28, pp. 63–86. ISBN: 978-1-4419-5652-1.

Hevner, Alan R, Salvatore T March, Jinsoo Park, and Sudha Ram (2004). "Design science in information systems research". In: *MIS quarterly*, pp. 75–105.

Huckle, Steve, Rituparna Bhattacharya, Martin White, and Natalia Beloff (2016). "Internet of things, blockchain and shared economy applications". In: *Procedia computer science* 98, pp. 461–466.

Jansen, Slinger, Siamak Farshidi, Georgios Gousios, Tijs van der Storm, Joost Visser, and Magiel Bruntink (2020). "SearchSECO: A Worldwide Index of the Open Source Software Ecosystem". In:

Jensen, Randall (1983). "An improved macrolevel software development resource estimation model". In: *5th ISPA Conference*, pp. 88–92.

Kennedy, William G (2012). "Modelling human behaviour in agent-based models". In: *Agent-based models of geographical systems*. Springer, pp. 167–179.

Kiczales, Gregor, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier, and John Irwin (1997). "Aspect-oriented programming". In: *European conference on object-oriented programming*. Springer, pp. 220–242.

King, Sunny (2013). "Primecoin: Cryptocurrency with prime number proof-of-work". In: *July 7th* 1.6.

Kitchenham, Barbara and Stuart Charters (2007). "Guidelines for performing systematic literature reviews in software engineering". In:

Kolvart, Merit, Margus Poola, and Addi Rull (2016). "Smart contracts". In: *The Future of Law and etechnologies*. Springer, pp. 133–147.

Korba, Larry and Ronggong Song (2002). "Modeling and Simulating the Scalability of A Multi-agent Application System". In: *National Reseach Council of Canada*.

Kraft, Daniel (2016). "Difficulty control for blockchain-based consensus systems". In: *Peer-to-Peer Networking and Applications* 9.2, pp. 397–413.

Lamport, Leslie, Robert Shostak, and Marshall Pease (2019). "The Byzantine generals problem". In: *Concurrency: the Works of Leslie Lamport*, pp. 203–226.

Levy, Eliezer and Abraham Silberschatz (1990). "Distributed file systems: Concepts and examples". In: *ACM Computing Surveys (CSUR)* 22.4, pp. 321–374.

Liu, Henry H (2011). *Software performance and scalability: a quantitative approach*. Vol. 7. John Wiley & Sons.

Lorig, Fabian, Nils Dammenhayn, David-Johannes Müller, and Ingo J Timm (2015). "Measuring and comparing scalability of agent-based simulation frameworks". In: *German Conference on Multiagent System Technologies*. Springer, pp. 42–60.

Lucassen, Garm, Fabiano Dalpiaz, Jan Martijn EM van der Werf, and Sjaak Brinkkemper (2016). "Improving agile requirements: the quality user story framework and tool". In: *Requirements Engineering* 21.3, pp. 383–403.

Luke, Sean, Gabriel Catalin Balan, Liviu Panait, Claudio Cioffi-Revilla, and Sean Paus (2003). "MASON: A Java multi-agent simulation library". In: *Proceedings of Agent 2003 Conference on Challenges in Social Simulation*. Vol. 9. 9.

Luu, Loi, Duc-Hiep Chu, Hrishi Olickel, Prateek Saxena, and Aquinas Hobor (2016). "Making smart contracts smarter". In: *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pp. 254–269.

Macal, C, D Sallach, and M North (2004). "Emergent structures from trust relationships in supply chains". In: *Proc. Agent 2004: Conf. on Social Dynamics*, pp. 7–9.

Macal, Charles M and Michael J North (2005). "Tutorial on agent-based modeling and simulation". In: *Proceedings of the Winter Simulation Conference, 2005.* IEEE, 14–pp.

Mahood, Quenby, Dwayne Van Eerd, and Emma Irvin (2014). "Searching for grey literature for systematic reviews: challenges and benefits". In: *Research synthesis methods* 5.3, pp. 221–234.

Malleson, NS, AJ Heppenstall, and LM See (2010). "Simulating burglary with an agent-based model". In: *Computers, Environment and Urban Systems* 34.3, pp. 236–250.

March, Salvatore T and Gerald F Smith (1995). "Design and natural science research on information technology". In: *Decision support systems* 15.4, pp. 251–266.

Marsh, William E and Raymond R Hill (2008). "An initial agent behaviour modelling and definition methodology as applied to unmanned aerial vehicle simulations". In: *International Journal of Simulation and Process Modelling* 4.2, pp. 119–129.

Mernik, Marjan, Jan Heering, and Anthony M Sloane (2005). "When and how to develop domain-specific languages". In: *ACM computing surveys (CSUR)* 37.4, pp. 316–344.

Michael, Maged, Jose E Moreira, Doron Shiloach, and Robert W Wisniewski (2007). "Scale-up x scale-out: A case study using nutch/lucene". In: *2007 IEEE International Parallel and Distributed Processing Symposium*. IEEE, pp. 1–8.

Miller, John H and Scott E Page (2009). *Complex adaptive systems: An introduction to computational models of social life*. Princeton university press.

Moon, Young B (2017). "Simulation modelling for sustainability: a review of the literature". In: *International Journal of Sustainable Engineering* 10.1, pp. 2–19.

Mur, Ricardo Aler (2006). "Automatic inductive programming". In: *Proceedings of the 23rd international conference on machine learning, tutorial*.

Neuman, B Cli ord (1994). "Scale in distributed systems". In: *ISI/USC*, p. 68.

North, Michael J, Nicholson T Collier, Jonathan Ozik, Eric R Tatara, Charles M Macal, Mark Bragen, and Pam Sydelko (2013). "Complex adaptive systems modeling with Repast Simphony". In: *Complex adaptive systems modeling* 1.1, p. 3.

North, Michael J and Charles M Macal (2007). *Managing business complexity: discovering strategic solutions with agent-based modeling and simulation*. Oxford University Press.

North, Michael J, Charles M Macal, James St Aubin, Prakash Thimmapuram, Mark Bragen, June Hahn, James Karr, Nancy Brigham, Mark E Lacy, and Delaine Hampton (2010). "Multiscale agent-based consumer market modeling". In: *Complexity* 15.5, pp. 37–47.

Nunnari, Fabrizio and Alexis Heloir (2018). "Write-once, transpile-everywhere: reusing motion controllers of virtual humans across multiple game engines". In: *International Conference on Augmented Reality, Virtual Reality and Computer Graphics*. Springer, pp. 435–446.

Osis, Janis and Erika Asnina (2010). *Model-driven domain analysis and software development: Architectures and functions: Architectures and functions*. IGI Global.

Parizi, Reza M, Ali Dehghantanha, Kim-Kwang Raymond Choo, and Amritraj Singh (2018). "Empirical vulnerability analysis of automated smart contracts security testing on blockchains". In: *arXiv preprint arXiv:1809.02702*.

Parker, Dawn C, Steven M Manson, Marco A Janssen, Matthew J Hoffmann, and Peter Deadman (2003). "Multi-agent systems for the simulation of land-use and land-cover change: a review". In: *Annals of the association of American Geographers* 93.2, pp. 314–337.

Parnas, David Lorge (1985). "Software aspects of strategic defense systems". In: *Communications of the ACM* 28.12, pp. 1326–1335.

Parunak, H Van Dyke, Robert Savit, and Rick L Riolo (1998). "Agent-based modeling vs. equation-based modeling: A case study and users' guide". In: *International Workshop on Multi-Agent Systems and Agent-Based Simulation*. Springer, pp. 10–25.

Perez, Daniel and Benjamin Livshits (2019). "Smart contract vulnerabilities: Does anyone care?" In: *arXiv preprint arXiv:1902.06710*.

Pressman, Roger S (2005). *Software engineering: a practitioner's approach*. Palgrave macmillan.

Prieto, Luis P, Maria Jesus Rodriguez Triana, Marge Kusmin, and Mart Laanpere (2017). "Smart school multimodal dataset and challenges". In: *Joint proceedings of the sixth Multimodal Learning Analytics (MMLA) workshop and the second cross-LAK workshop co-located with 7th international learning analytics and knowledge conference*. Vol. 1828. CONF. CEUR, pp. 53–59.

Putnam, Lawrence H and Ware Myers (1991). *Measures for excellence: reliable software on time, within budget*. Prentice Hall Professional Technical Reference.

Ralyté, Jolita and Colette Rolland (2001). "An assembly process model for method engineering". In: *International Conference on Advanced Information Systems Engineering*. Springer, pp. 267–283.

Roberts, Paula and Helena Priest (2006). "Reliability and validity in research". In: *Nursing standard* 20.44, pp. 41–46.

Röscheisen, Martin, Michelle Baldonado, Kevin Chang, Luis Gravano, Steven Ketchpel, and Andreas Paepcke (1998). "The Stanford InfoBus and its service layers: Augmenting the Internet with higher-level information management protocols". In: *Digital Libraries in Computer Science: The MeDoc Approach*. Springer, pp. 213–230.

Runeson, Per and Martin Höst (2009). "Guidelines for conducting and reporting case study research in software engineering". In: *Empirical software engineering* 14.2, p. 131.

Savelyev, Alexander (2017). "Contract law 2.0:'Smart'contracts as the beginning of the end of classic contract law". In: *Information & Communications Technology Law* 26.2, pp. 116–134.

Schelling, Thomas C (1971). "Dynamic models of segregation". In: *Journal of mathematical sociology* 1.2, pp. 143–186.

Schollmeier, Rüdiger (2001). "A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications". In: *Proceedings First International Conference on Peer-to-Peer Computing*. IEEE, pp. 101–102.

Smajgl, Alex, Luis R Izquierdo, and Marco Huigen (2008). "Modeling endogenous rule changes in an institutional context: The adico sequence". In: *Advances in Complex Systems* 11.02, pp. 199–215.

Stahl, Thomas, Markus Voelter, and Krzysztof Czarnecki (2006). *Model-driven software development: technology, engineering, management*. John Wiley & Sons, Inc.

Terna, Pietro et al. (1998). "Simulation tools for social scientists: Building agent based models with swarm". In: *Journal of artificial societies and social simulation* 1.2, pp. 1–12.

Tikhomirov, Sergei, Ekaterina Voskresenskaya, Ivan Ivanitskiy, Ramil Takhaviev, Evgeny Marchenko, and Yaroslav Alexandrov (2018). "Smartcheck: Static analysis of ethereum smart contracts". In: *Proceedings of the 1st International Workshop on Emerging Trends in Software Engineering for Blockchain*, pp. 9–16.

Tripathy, Abinash, Ankit Agrawal, and Santanu Kumar Rath (2014). "Requirement analysis using natural language processing". In: *Fifth International Conference on Advances in Computer Engineering*, pp. 26–27.

Tsankov, Petar, Andrei Dan, Dana Drachsler-Cohen, Arthur Gervais, Florian Buenzli, and Martin Vechev (2018). "Securify: Practical security analysis of smart contracts". In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pp. 67–82.

Vo, Hoang Tam, Ashish Kundu, and Mukesh K Mohania (2018). "Research Directions in Blockchain Data Management and Analytics." In: *EDBT*, pp. 445–448.

Walls, Joseph G, George R Widmeyer, and Omar A El Sawy (1992). "Building an information system design theory for vigilant EIS". In: *Information systems research* 3.1, pp. 36–59.

Weerd, Inge van de and Sjaak Brinkkemper (2009). "Meta-modeling for situational analysis and design methods". In: *Handbook of research on modern systems analysis and design technologies and applications*. IGI Global, pp. 35–54.

Wieringa, Roel J (2014). *Design science methodology for information systems and software engineering*. Springer.

Wohlin, Claes (2014). "Guidelines for snowballing in systematic literature studies and a replication in software engineering". In: *Proceedings of the 18th international conference on evaluation and assessment in software engineering*, pp. 1–10.

Wohlin, Claes, Martin Höst, and Kennet Henningsson (2003). "Empirical research methods in software engineering". In: *Empirical methods and studies in software engineering*. Springer, pp. 7–23.

Wooldridge, Michael J and Nicholas R Jennings (1995). "Intelligent agents: Theory and practice". In: *The knowledge engineering review* 10.2, pp. 115–152.

Xu, Yifei, Wei Yu, and Kamy Sepehrnoori (2019). "Modeling dynamic behaviors of complex fractures in conventional reservoir simulators". In: *SPE Reservoir Evaluation & Engineering* 22.03, pp. 1110–1130.

Yaga, Dylan, Peter Mell, Nik Roby, and Karen Scarfone (2019). "Blockchain technology overview". In: *arXiv preprint arXiv: 1906.11078*.

Zheng, Zibin, Shaoan Xie, Hongning Dai, Xiangping Chen, and Huaimin Wang (2017). "An overview of blockchain technology: Architecture, consensus, and future trends". In: *2017 IEEE international congress on big data (BigData congress)*. IEEE, pp. 557–564.

Grey Literature

Ballard, Dan (Nov. 2011). URL: http://www.agentbuilder.com/Documentation/whyAgents.html.

Buterin, Vitalik (2013). *Ethereum Whitepaper*. URL: https://ethereum.org/en/whitepaper/.

Holland, J (1996). *Holland, Hidden order: how adaptation builds complexity*.

Nakamoto, Satoshi (2019). *Bitcoin: A peer-to-peer electronic cash system*. Tech. rep. Manubot.

Schulpen, Ruben (2018). *A legal research regarding the use of smart contracts within Dutch contract law and legal framework*. URL: http://arno.uvt.nl/show.cgi?fid=146860.

Wohlin, C, P Runeson, M Höst, M Ohlsson, B Regnell, and A Wesslén (2012). *Experimentation in software engineering: A practical guide*.