The use of Blockchain technology to achieve web scale agent-based simulations

Conor Deegan[1][0000-0001-7835-2696]

School of Computer Science, University College Dublin, Dublin, Ireland
conor.deegan@ucdconnect.ie

Abstract. The paper explores the use of Blockchain technology to achieve web scale agent-based simulations. Agent-based simulations are used in many domains including logistics [16], biology [20], ecology [28], and social science [34]. However, research has shown more complex models, such as modelling real world simulations, cannot be captured by a single model on a single machine [25]. Current research to solve this problem includes the use of Hybrid Simulations (HS) which combines multiple interconnected sub-simulations [17], and the work in [11] which builds upon this further by exploring the use of loosely-coupled microservices to implement HS at scale. There has been a recent increase in popularity of blockchains, driven by networks such as Bitcoin and Ethereum. Blockchains, such as the Ethereum blockchain, come equipped with a built in Turing-complete programming language which can be used to create decentralised applications which are executed publicly on a global network of decentralised nodes. This thesis aims to build a proof-of-concept agent-based simulation which runs on this decentralised global network and achieves a scale similar to production grade web applications. A toy traffic simulation problem is created where agents commute from their home to work across a network consisting of buildings, roads, and junctions. Each entity in the simulation is implemented using Solidity smart contracts. This proof-of-concept simulation demonstrates how blockchain technology can be used to facilitate complex agent-based simulations, which can be integrated with existing centralised simulations as needed.

Keywords: Agent-based modelling · Blockchain · Distributed computing

1 Introduction

Agent-Based Modelling (ABM) is a bottom-up approach to studying the behaviour of complex systems [36]. Systems are modelled as a collection of individual agents where each agent encapsulates private state and can make decisions based on a set of rules [5]. The behaviour of the system emerges through the decisions made by these agents and their interactions with one another. ABM has been applied in many domains including logistics [16], biology [20], ecology [28], and social science [34].

One of the key challenges in ABM is the ability to model systems of increasing complexity [11]. Work has shown that more complex ABM cannot be captured by a single model on a single machine [25]. Current solutions to this scaling problem include the use of Hybrid Simulation (HS) which combines multiple interconnected sub-simulations where each simulation may be implemented using different modelling techniques [17]. The work in [11] builds upon this further by exploring the use of loosely-coupled microservices to implement HS at scale.

In recent years, there has been increasing concerns around centralised web services in areas such as privacy, governance, surveillance, and security. This has triggered the emergence of new distributed technologies such as blockchain technology (BCT) [41]. Blockchains, such as the Ethereum blockchain [44], come equipped with a built in Turing-complete programming language which can be used to create decentralised applications which are executed publicly on a global network of decentralised nodes.

This thesis explores the use of BCT to achieve web scale agent-based simulations. It concludes with an implementation and description of a proof of concept ABM running on BCT.

2 Agent-Based Simulation

Agent Based Modelling is an established simulation technique that has been applied successfully in a variety of domains. A detailed literature review of ABM has been presented separately. This section reviews the findings discussed within the literature review in the context of the work presented in this thesis.

A key challenge in the area of ABM is the ability to model systems of increasing complexity [11]. Although there are a range of approaches to implementing ABMs [1], they have been traditionally viewed as a desktop computer style of exercise where simulations are executed on a single machine [11]. However, work has shown that desktop agent based models do not scale to what is required for extremely large applications in the study of realistic complex systems [2] [25]. Although an in-depth review of the existing literature can be found in the preceding literature review, this section will give a brief overview of the most pertinent literature discussed.

Components of ABMs that contribute to slower execution times and memory issues when scaling to larger simulations include; a larger number of agents, model design, and computational complexity of the agents behaviour [10]. This can lead to model simplifications until the execution time is acceptable on a single machine [40].

ABMs which push the limits of resources due to their large numbers of agents or their complexity may be referred to as 'Massively Multi-agent Systems (MMAS)' or 'Massive Agent-based Systems (MABS)' [24] [23].

Current state of the art solutions for scaling agent-based simulations include the use of Hybrid Simulation [17]. Hybrid Simulation combines multiple interconnected sub-simulations where each simulation may be implemented using different modelling techniques [31]. Within the simulation community, two main approaches have emerged; Distributed Simulation and Cloud Based Simulation.

The Distributed Simulation (DS) approach focuses on the development and deployment on high-performance computing clusters, leading to tools such has RePAST HPC [9]. However, these implementations are typically bespoke and tailored to specific tasks [39]. Thus, a DS approach does not solve issues such as interoperability and model reuse that are required for large-scale ABM.

Another criticism of DS is the cost and availability of computing clusters. This has led to the proposal of Cloud Based Simulation (CBS) [39], which focuses on the deployment of simulations in the cloud using microservices [40].

However, research has shown that there is a lack of suitable tools and frameworks for integrating ABMs with other technologies [36]. Interoperability is a particular issue leading to many existing Hybrid Simulations being built using a single tool [17].

The work in [11] argues that Hybrid Simulations should not be built on monolithic architectures but instead implemented as loosely-coupled microservices in a manner that ensures scalability. They highlight that the use of microservices to achieve greater scale in ABM allows for reusable components, interoperability, polyglot development, and deployment at scale. They propose that each sub-simulation is encapsulated as a microservice that uses REST for communication and can be integrated into larger simulations that are not agent-based. This has been demonstrated using Multi-Agent Microservices (MAMS) in the ASTRA agent programming language [43] [35] [12] [14]

This thesis will build upon the research in [11] and explore the use of smart contracts to achieve greater scale in ABM.

3 Blockchain Technology

At it's core, a blockchain is a distributed peer-to-peer (P2P) network which allows for transactions to take place in a fully trust-less environment. It does this through the use of digital signatures, a digital ledger to maintain the network state, and some form of consensus algorithm used to achieve agreement about this state. Blockchains, such as the Bitcoin network, solve a problem with digital cash known as the double spending problem - a potential flaw in a digital cash scheme in which the same single digital token can be spent more than once [33]. Other blockchains, such as the Ethereum network, extend this further by providing a blockchain with a built in programming language. This can be used to create programs that run across the distributed network. In the Ethereum ecosystem, these are known as smart contracts [44].

Rather than describing the Ethereum network as a distributed ledger, a more suitable description for the network would be a distributed state machine. This state machine consists of a large data structure which holds all of the Ethereum accounts and balances and is also capable of executing compiled smart contract machine code. The rules which dictate how this state changes from block to block are defined by the Ethereum Virtual Machine [44].

At the time of this thesis, the consensus algorithm used by the Ethereum main network is a Proof of Work (POW) algorithm. As a result of this, the Ethereum main network is currently limited to just fifteen transactions per second (TPS). However, upcoming updates to the main network will switch the consensus algorithm to a Proof of Stake (POS) algorithm, resulting in an increase to over 100,000 TPS. This increase in TPS will result in the Ethereum network being able to handle similar throughput as many centralised production systems [29]. Some Ethereum test networks and Layer 2 solutions already use a POS consensus algorithm. The remainder of this thesis will assume that the underlying network uses POS.

3.1 Scale

The Ethereum network is a P2P network in which [27] estimated there to be approximately 300,000 nodes on the network at the time of their research. Each of these nodes acts as a server running the EVM. Combined with the increase in throughput obtained by the POS consensus algorithm, the Ethereum network allows for cost-effective scale without the need for application developers to manage a large cluster of servers. This provides a good argument for leveraging existing open P2P protocols for efficiently running applications at scale.

3.2 Interoperability

Blockchain interoperability is emerging as one of the crucial features of blockchain technology [4]. Given that blockchains, such as the Ethereum network, are public by default, application developers can extend existing smart contracts and build upon current infrastructure provided their smart contracts are constructed appropriately. The Ethereum community recommends that, for the purposes of interoperability, smart contracts follow strict interface guidelines. Examples of the effectiveness of interoperability offered by the blockchain can be seen with the ERC-20 fungible token [6], and ERC-721 non-fungible token [18] standards.

3.3 Traceability

BCT provides an immutable and permanent history of transactions [21]. As such, blockchains provide an accurate historic account of events that cannot be altered. Every function call and smart contract execution is recorded by transaction logs on chain and can be used to learn more about the state of the system at any previous point in time. When applied to ABM, this offers traceability and model recreation at any point in a simulations life cycle.

3.4 Blockchain Technology and Agent-based Simulation

There exists a body of research which explores the benefits of BCT for ABM. However, limited research could be found exploring the use of blockchain technology to achieve scale in agent-based simulations. Most studies were concerned with other affordances offered by BCT such as trust [7] [3] [15] [41], data immutability [15] [3] [41], mitigating single-points of failure [3], cost [3], and security [37] [7]. This thesis will focus on three core elements of BCT and their possible benefits for ABM; scale, interoperability, and traceability.

4 A New Approach

In this thesis, ABM architectures are implemented as loosely-coupled smart contracts in a manner that ensures scalability; given the affordances of the underlying distributed network, traceability; given the immutable nature of BCT, and interoperability; given the network is public by default. Table 1 shows similarities which can be drawn between smart contract programs and microservices [42]. As such, this new ABM architecture can operate solely on BCT or as part of a hybrid system building on the work in [11].

This architecture can also be compared to the idea of a Metaverse. [26] defines the Metaverse as a gigantic, unified, persistent, interconnected web of virtual environments. Each environment node can be realised through different technologies such as Virtual Reality, Augmented Reality, centralised applications or indeed decentralised applications [32]. The Metaverses described thus far consists of human controlled avatars operating in a digital universe. However, another body of Metaverse research is concerned with creating a digital twin of our universe and simulating events to study the effects they might have in our reality [38] [22]. Large scale simulations can be developed to study the spread of diseases, evacuation events, traffic patterns etc. In order to model reality effectively, clear guidelines must be developed which document how different nodes of the environment can be connected, how agents in those environments can interact with their surroundings, and how information can be extracted from these simulations to study the results. Given the public interoperable nature of BCT, once these guidelines are enforced, simulations could become open allowing different environment nodes to be added to the simulation by anyone. This would result in a truly decentralised digital twin of our universe used to simulate events and study their effects.

The associated code for this thesis implements a possible starting point for this model. Following is an outline of each of its components.

Agents Generally, there exists two approaches when applying BCT to ABM and Multi-Agent Systems (MAS). The first approach uses traditional centralised agents running on a centralised machine where the blockchain is used as an immutable ledger and trusted data store. The second approach is more integrated, the agents directly interact with, or are part of, the blockchain technology. [8] describe these two approaches as "agent-vs-blockchain" - where the agent and blockchain run side by side, allowing the agents to exploit blockchain services when needed and "agent-to-blockchain" - where the effort is focused on incorporating agent-oriented models and technologies directly into the blockchain. The approach in this thesis follows the "agent-to-blockchain" approach where all elements of the model run on BCT.

Principle	**Microservices**	**Smart Contracts**
Bounded Context	A microservice represents a single piece of business functionality.	A smart contract typically implements a simple autonomous task with a well defined purpose.
Size	Microservices should be small enough to ensure maintainability and extensibility	Given gas costs, smart contracts are typically small in size. They offer extensibility through inheritance.
Isolated State	Sharing of state information is minimised across services.	Smart contract state is maintained privately and can be constructed in such a way that sharing of state information across services in minimal.
Distribution	Services are spread across multiple nodes.	Services are spread across multiple nodes.
Elasticity	The application is designed to allow addition and removal of required resources at runtime.	Not applicable as smart contract execution happens across the entire distributed network by default.
Automated Management	Management operations like failure handling and scaling are automated.	Not applicable, although failure can be accommodated within the logic of the contract.
Loose Coupling	Systems are decomposed into loosely coupled sets of highly cohesive colocated services.	Systems are decomposed into loosely coupled smart contracts distributed across the entire network

Table 1. Comparing Smart Contracts to Microservices

In this new architecture, agents will be implemented using smart contracts. Agents will have an internal state and will make decisions based on specific rules, internal beliefs, and input from their surroundings. The agents will be encapsulated entities and will be capable of flexible autonomous actions in order to meet their design objectives. Thus, these agents follow the popular definition of an agent proposed by Wooldridge and Jennings in [45]. Given the persistent nature of BCT, agents can be added or removed at any point in time during the simulation. This offers greater freedom where more agents can be added after the simulation begins.

Environment The environment moves from a single centralised environment to a distributed environment whereby different aspects of the environment exist as distributed nodes within the network. Similarly to agents, environment nodes are realised through smart contracts and can be added or removed in realtime. This results in a dynamic environment that is not limited in size.

Service Lookup Given the distributed nature of this architecture, a service must be developed for node discovery. This distributed service lookup (DSL) acts like a service registry whereby nodes; agent or environment nodes, register themselves as they come online. Once registered, the DSL acts like a distributed DNS service whereby the location of nodes; their addresses, can be found by querying the DSL. It's important to note that this service does not act like an API gateway [30], as this would introduce a single point of failure into the architecture. Instead, once node A queries the DSL for the location of node B, node A can interact directly with node B without any intermediaries. This offers a more decentralised and scalable solution. This DSL will be implemented as a smart contract.

Communication A communication protocol is required for agent to environment and agent to agent interaction. This protocol will also be used for interaction with the DSL. This communication can take place on-chain as Solidity allows for inter smart contract communication provided the smart contract address and interface is known.

Oracles Oracles are data feeds that connect Ethereum to off-chain information. They act as on-chain APIs which can be queried to get off-chain information into smart contracts. With BCT, each node in the network must replay all transactions in order to reach consensus. Off-chain data introduces potentially variable data. For example, if a smart contract queried the price of the US dollar from an off-chain data source in order to perform some logic, when other nodes attempt to replay this transaction the price of the dollar may have changed and they may compute different results as a consequence. If this happens, the nodes in the network will not be able to agree on the networks current state, breaking consensus. Oracles solve this problem by storing the initial returned data on-chain. When other nodes replay this transaction they use the on-chain immutable data rather than querying the off-chain data source again [19]. Event based oracles rely on off-chain programs listening for events emitted from a smart contract. Once an event is received, the off-chain oracle can complete its computation and post the data back to the smart contract. This process is asynchronous.

In this architecture, oracles will be used for off-chain computation and data querying where needed. This allows this smart contract centric architecture to be integrated with microservices and centralised systems as needed.

Interfaces Each entity in the system should adhere to standardised interfaces. This promotes interoperability. Provided they follow the standard agent interface, new agents can be added to the simulation as needed. These new agents may have different internal beliefs and follow different rules. However, given they implement the standard agent interface they will be able to interact with one another and their environment accordingly. Similarly, new environment node types can be easily added to the simulation provided they adhere to the environment node interface regardless of their underlying logic.

Traceability All entities should produce logs for traceability. Logs can be gathered retrospectively from on-chain transaction data but also in real time from events. These logs are an immutable reference to the state of the simulation at any point in time.

5 Illustrating the Vision

In order to illustrate this new vision, I have created a toy problem based on a city simulation where an agent must navigate through the city based on goals it is presented with. An example of this distributed network in shown in Figure 1. The city is represented as a graph with nodes representing buildings, roads, and junctions. Agents can move from one node to the next based on their plan. Agents can also interact with a Sat Nav node. This is a specialised node which acts as both an oracle and a distributed service lookup node as discussed in section 4.

This example can be considered a hybrid simulation as it consists of multiple sub-simulations that are connected together to create a distributed simulation. Each of the environment nodes; roads, junctions, and buildings, are customizable. For example, each road is given a specific road length which dictates how long an agent must wait in that node until it can move on. Similarly, each junction has an associated junction queue and is integrated with an external traffic light management simulation.

Figure 2 shows a layered architecture diagram of the system created. Layer 1 consists of a React.JS web application which renders a visualisation of the city network and the state of each of the agents within the simulation in realtime. A screenshot of this visualisation can be seen later in Figure 7. This web application can also be used to call smart contract functions in layer 4, via layers 2 and 3. Layers 2 and 3 consist of an API layer and a communication layer in order to communicate with the Ethereum Blockchain, which inhabits layer 4. Finally, layer 5 consists of the nodes running the Ethereum network. During development and testing this layer is simply a single node running locally.

To illustrate the concept, the example simulation is setup as follows:

- Agents have initial beliefs about their current location in the network.
- At any point in time, an agent can be given a goal location in the distributed environment to navigate to.

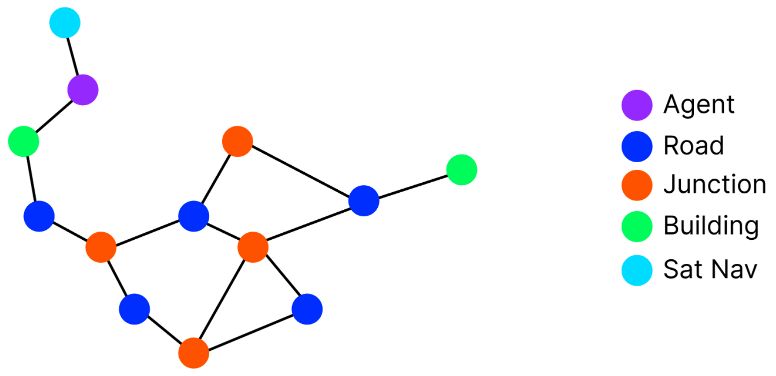


Fig. 1. Representation of the road network being simulated

- Based on this goal and the agent's current location belief, agents interact with a Sat Nav node. This specialised node is an oracle which interacts with an off-chain Node.js application in order to compute the shortest path between the agent's current location and then agent's goal location. The Node.js application implements a breadth-first search in order to compute this. Although in this example the Node.js application is a standalone application, it could be incorporated as a microservice in a larger microservice Hybrid Simulation architecture as described in [11].
- The Sat Nav returns the optimal route to the agent.
- Using this optimal route, the agent creates a plan. This plan is a list of environment nodes that the agent must navigate through in order to arrive at it's goal location.
- The Sat Nav node also acts as the Distributed Service Lookup as each environment node registers itself with the Sat Nav when it comes online.
- Once the agent has a plan, on each epoch, the agent can lookup the necessary environment node from the DSL and interact directly with that node.
- Given each environment node implements a standardised interface, the agent can interact with each node despite the nodes underlying logic; road length, junction queue, etc.
- Once an agent completes their plan and arrives at their goal location, they can now be given a new goal and the process can repeat.

A benefit of this approach is that other simulations that have no direct link to the agents can also be integrated. These external simulations may be realised through smart contracts or could be entirely different such as centralised services or other forms of artificial intelligence. For example, a weather simulation may be integrated into the network which could alter road conditions and thus change navigation patterns.

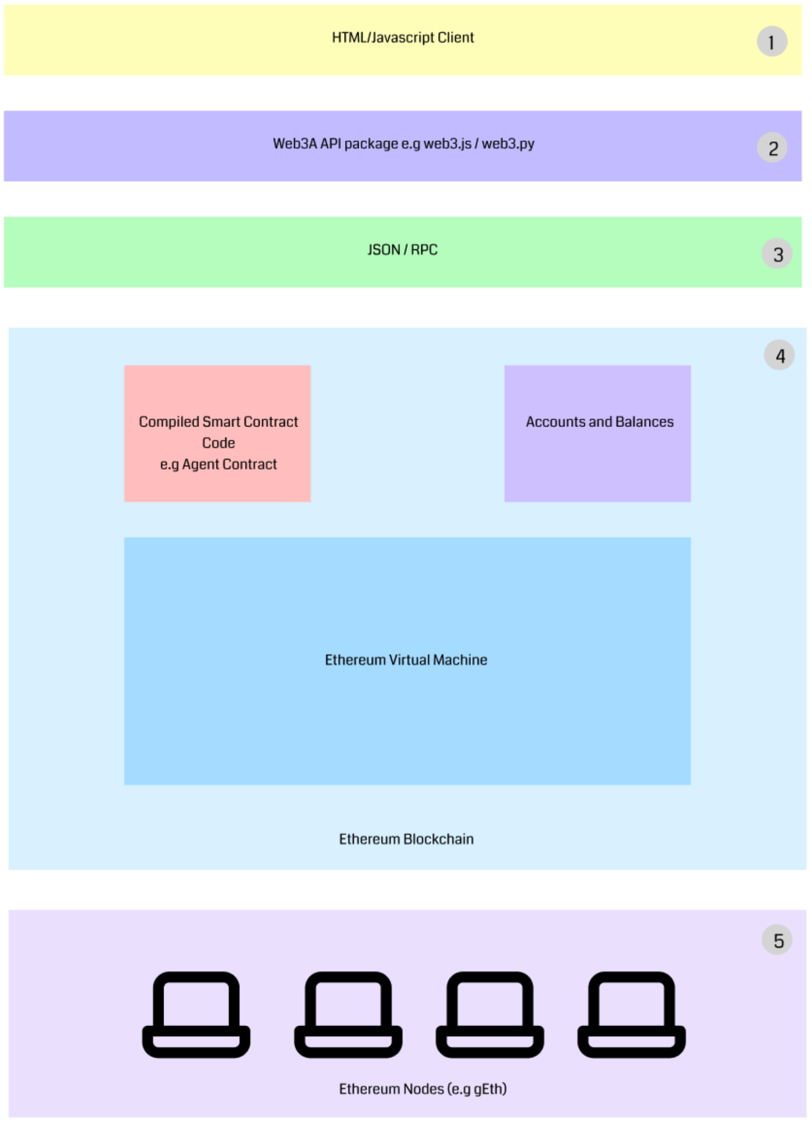
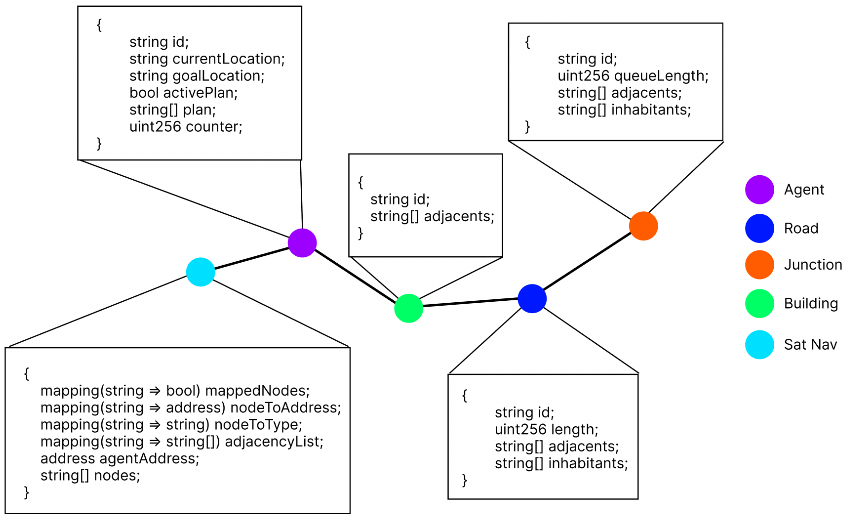

HTML/Javascript Client
1
Web3A API package e.g web3.js / web3.py
2
JSON / RPC
3
4
Compiled Smart Contract Code e.g Agent Contract
Accounts and Balances
Ethereum Virtual Machine
Ethereum Blockchain
5
Ethereum Nodes (e.g gEth)

Fig. 2. Architecture of Ethereum Application



```
{
        string id;
        string currentLocation;
        string goalLocation;
        bool activePlan;
        string[] plan;
        uint256 counter;
}
```

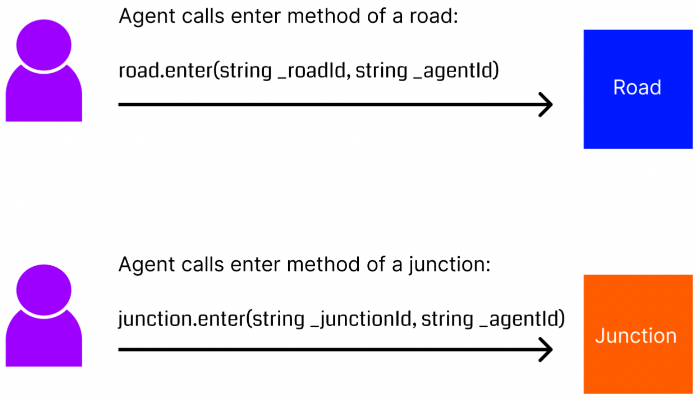
```
{
   string id;
   string[] adjacents;
}
```

```
{
        string id;
        uint256 queueLength;
        string[] adjacents;
        string[] inhabitants;
}
```

```
{
    mapping(string ⇒ bool) mappedNodes;
    mapping(string ⇒ address) nodeToAddress;
    mapping(string ⇒ string) nodeToType;
    mapping(string ⇒ string[]) adjacencyList;
    address agentAddress;
    string[] nodes;
}
```

```
{
        string id;
        uint256 length;
        string[] adjacents;
        string[] inhabitants;
}
```

Fig. 3. Representation of the data structures for each node in Solidity



Agent calls enter method of a road:

```
road.enter(string _roadId, string _agentId)
```

Agent calls enter method of a junction:

```
junction.enter(string _junctionId, string _agentId)
```

Fig. 4. Polymorphism offered by standardised interfaces

5.1 Agents

The code in Listing 1.1 demonstrates how an agent can be created and added to the network.

```
// function used to create an agent
function create(string memory _agentId, string memory _currentLocation)
    public
{
    Agent memory newAgent = Agent({
        id: _agentId,
        currentLocation: _currentLocation,
        goalLocation: "",
        activePlan: false,
        goalLocationStatus: false,
        plan: new string[](0),
        counter: 0,
        totalEpochs: 0
    });
    agents[_agentId] = newAgent;
    agentIds.push(_agentId);
    INode(getNodeAddress(_currentLocation)).enter(
        _currentLocation,
        _agentId
    );
}
```

Listing 1.1. Create an Agent

Each agent must consist of a unique identifier and a starting location in the network. Once an agent is given a goal, the goalLocation is updated along with an activePlan Boolean and a plan. This plan is a route of nodes to traverse in order to reach it's goal destination. On each epoch every agent in the simulation is executed by the epoch controller. This is a Node.JS application which calls the epoch method of each agent. This method can be seen in Listing 1.2. As can be seen at the end of the function, once the agent is created it communicates directly with the environment node it is located within.

5.2 Environment Nodes

As can be seen in Figure 3, each environment node has a different underlying data structure. However, polymorphism is achieved through each environment node adhering to a standardised interface despite the different data structures and underlying node logic. An example of this can be seen in Figure 4. The interface for environment nodes can be seen in Listing 1.3.

```
// function called on each epoch
function epoch(string memory _agentId) public {
    Agent storage agent = agents[_agentId];
    if (!agent.activePlan && agent.goalLocationStatus) {
        creatPlan(agent);
        return;
    } else if (!agent.activePlan) {
        return;
    } else {
        agent.totalEpochs++;
        ISatNav(satNavAddress).getOptimalMove(
            _agentId,
            agent.currentLocation,
            agent.counter,
            agent.plan
        );
        return;
    }
}
```

Listing 1.2. Executing an agent

```
// contracts/INode.sol
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

interface INode {
    function enter(string memory _nodeId, string memory _agentId)
        external;

    function exit(string memory _nodeId, string memory _agentId)
        external;

    function progress(string memory _nodeId, string memory _agentId)
        external;

    function getLength(string memory _nodeId) external returns (uint256);
}
```

Listing 1.3. Environment Node Interface

Listing 1.4 shows the necessary code required in order to add a new road to the network. Each road must have a unique identifier, a length, and a list of environment nodes it links to. Adding a junction or building to the network is very similar and can be found in the attached resource [13].

```
// function used to add a road to the network
function add(
   string memory _roadId,
   uint256 _length,
   string[] memory _adjacents
) public {
   Road memory newRoad = Road({
      id: _roadId,
      length: _length,
      adjacents: _adjacents,
      inhabitants: new string[](0)
   });
   roads[_roadId] = newRoad;
   ISatNav(satNavAddress).addNode(
      _roadId,
      address(this),
      _adjacents,
      "road"
   );
}
```

Listing 1.4. Adding a Road Node

5.3 Communication

Smart contracts can interact by calling functions directly in other smart contracts provided the address is known. As discussed, the discovery of a node's address is facilitated by the Distributed Service Lookup (DSL). If an agent wishes to communicate with one of the road nodes, it must first resolve the road smart contract address using the DSL. After this, the agent can interact directly with the road node. This results in a truly decentralised P2P architecture. An example of this can be seen in listing 1.5 where the agent communicates directly with the Sat Nav node in order to retrieve the shortest path between it's current location and goal location.

```
// function used to create a plan for an agent (public)
function creatPlanById(string memory _agentId) public {
   Agent memory agent = agents[_agentId];
   ISatNav(satNavAddress).shortestPathRequest(
      agent.id,
      agent.currentLocation,
      agent.goalLocation
   );
}
```

Listing 1.5. Agent to Environment Communication

Although not explicitly demonstrated, inter-agent communication can take place the exact same way, given both environment nodes and agents are implemented using smart contracts. The communication protocol with off-chain entities can be seen in Figure 5 and listing 1.6

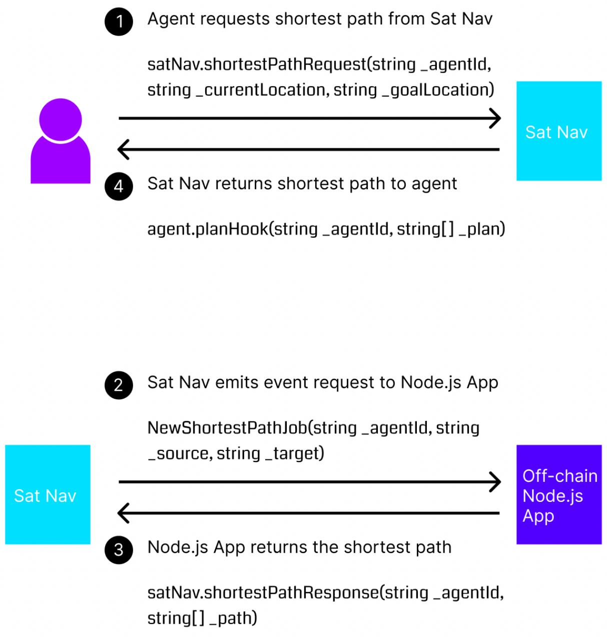

1 Agent requests shortest path from Sat Nav
satNav.shortestPathRequest(string _agentId, string _currentLocation, string _goalLocation)
Sat Nav
4 Sat Nav returns shortest path to agent
agent.planHook(string _agentId, string[] _plan)
2 Sat Nav emits event request to Node.js App
NewShortestPathJob(string _agentId, string _source, string _target)
Sat Nav
Off-chain Node.js App
3 Node.js App returns the shortest path
satNav.shortestPathResponse(string _agentId, string[] _path)

Fig. 5. Oracle communication protocol

5.4 Logs

Nodes emit important log messages which can be persisted for traceability. Given that the Ethereum blockchain is public, anyone can build an application which listens for logs produced by the simulation. These logs are immutable and provide a trustless account of the events of the simulation. These logs could also be consumed by other simulations, both centralised and decentralised, and reacted to accordingly. A sample of these logs can be seen in Figure 6

6 Evaluation

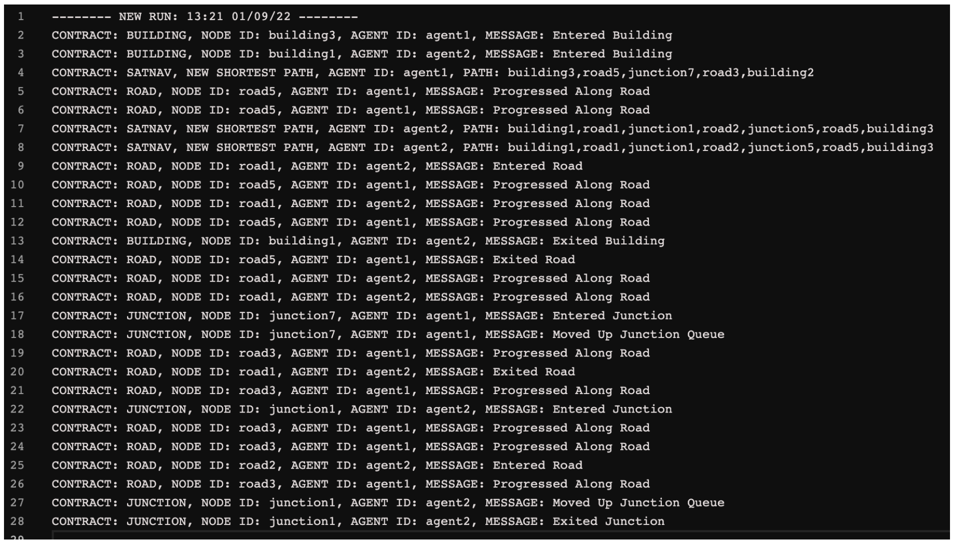
6.1 Experimental Set-Up

In order to demonstrate the effectiveness of this architecture, three different experiments were run. Each experiment was run on a MacBook Pro M1 laptop

```
// function used to initiate a new shortest path computation (on-chain)
function shortestPathRequest(
    string memory _agentId,
    string memory _source,
    string memory _target
) public {
    emit NewShortestPathJob(_agentId, _source, _target);
}

// off-chain listener which responds to the event emitted above
satNav.on("NewShortestPathJob", async (agentId, source, target) => {
    const shortestPath = await getShortestPath(satNav, source, target,
        agentId);
    await satNav.shortestPathResponse(shortestPath.agentId,
        shortestPath.path);
});
```

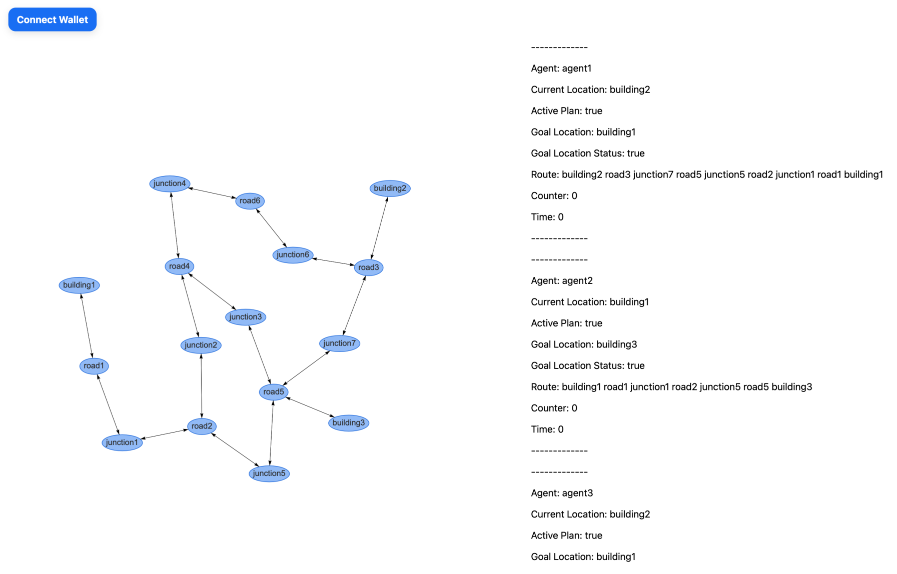
Listing 1.6. Oracle communication protocol



```
-------- NEW RUN: 13:21 01/09/22 --------
CONTRACT: BUILDING, NODE ID: building3, AGENT ID: agent1, MESSAGE: Entered Building
CONTRACT: BUILDING, NODE ID: building1, AGENT ID: agent2, MESSAGE: Entered Building
CONTRACT: SATNAV, NEW SHORTEST PATH, AGENT ID: agent1, PATH: building3,road5,junction7,road3,building2
CONTRACT: ROAD, NODE ID: road5, AGENT ID: agent1, MESSAGE: Progressed Along Road
CONTRACT: ROAD, NODE ID: road5, AGENT ID: agent1, MESSAGE: Progressed Along Road
CONTRACT: SATNAV, NEW SHORTEST PATH, AGENT ID: agent2, PATH: building1,road1,junction1,road2,junction5,road5,building3
CONTRACT: SATNAV, NEW SHORTEST PATH, AGENT ID: agent2, PATH: building1,road1,junction1,road2,junction5,road5,building3
CONTRACT: ROAD, NODE ID: road1, AGENT ID: agent2, MESSAGE: Entered Road
CONTRACT: ROAD, NODE ID: road5, AGENT ID: agent1, MESSAGE: Progressed Along Road
CONTRACT: ROAD, NODE ID: road1, AGENT ID: agent2, MESSAGE: Progressed Along Road
CONTRACT: ROAD, NODE ID: road5, AGENT ID: agent1, MESSAGE: Progressed Along Road
CONTRACT: BUILDING, NODE ID: building1, AGENT ID: agent2, MESSAGE: Exited Building
CONTRACT: ROAD, NODE ID: road5, AGENT ID: agent1, MESSAGE: Exited Road
CONTRACT: ROAD, NODE ID: road1, AGENT ID: agent2, MESSAGE: Progressed Along Road
CONTRACT: ROAD, NODE ID: road1, AGENT ID: agent2, MESSAGE: Progressed Along Road
CONTRACT: JUNCTION, NODE ID: junction7, AGENT ID: agent1, MESSAGE: Entered Junction
CONTRACT: JUNCTION, NODE ID: junction7, AGENT ID: agent1, MESSAGE: Moved Up Junction Queue
CONTRACT: ROAD, NODE ID: road3, AGENT ID: agent1, MESSAGE: Progressed Along Road
CONTRACT: ROAD, NODE ID: road1, AGENT ID: agent2, MESSAGE: Exited Road
CONTRACT: ROAD, NODE ID: road3, AGENT ID: agent1, MESSAGE: Progressed Along Road
CONTRACT: JUNCTION, NODE ID: junction1, AGENT ID: agent2, MESSAGE: Entered Junction
CONTRACT: ROAD, NODE ID: road3, AGENT ID: agent1, MESSAGE: Progressed Along Road
CONTRACT: ROAD, NODE ID: road3, AGENT ID: agent1, MESSAGE: Progressed Along Road
CONTRACT: ROAD, NODE ID: road2, AGENT ID: agent2, MESSAGE: Entered Road
CONTRACT: ROAD, NODE ID: road3, AGENT ID: agent1, MESSAGE: Progressed Along Road
CONTRACT: JUNCTION, NODE ID: junction1, AGENT ID: agent2, MESSAGE: Moved Up Junction Queue
CONTRACT: JUNCTION, NODE ID: junction1, AGENT ID: agent2, MESSAGE: Exited Junction
```

Fig. 6. Sample of output logs

with 16 gigabytes of memory. Hardhat was used to run an Ethereum node locally during development and testing. The visualisation tool discussed in Section 5 was used during each experiment to monitor the state of the network and agent progress in realtime. An example of this visualisation tool can be seen in Figure 7. For standardisation purposes, the network used in each experiment was identical. This is important so that each experiment is both reproducible and comparable.



```
-------------
Agent: agent1
Current Location: building2
Active Plan: true
Goal Location: building1
Goal Location Status: true
Route: building2 road3 junction7 road5 junction5 road2 junction1 road1 building1
Counter: 0
Time: 0
-------------
-------------
Agent: agent2
Current Location: building1
Active Plan: true
Goal Location: building3
Goal Location Status: true
Route: building1 road1 junction1 road2 junction5 road5 building3
Counter: 0
Time: 0
-------------
-------------
Agent: agent3
Current Location: building2
Active Plan: true
Goal Location: building1
```

Fig. 7. React.JS Visualisation Tool of the network

Experiment One The first experiment simply demonstrates the agents ability to navigate through the network from it's initial location, home, towards it's goal location, work. The length of time this commute takes it recorded in epochs. Given there is only one agent in the experiment, there is no build up of traffic at junctions and the only variable dictating how long it takes the agent to reach their destination is whether or not the junctions they arrive at have a green light or red light. The average amount of epochs taken for the agent to complete their goal in experiment 1 was 29 epochs.

Experiment Two The second experiment is similar to the first although 10 agents are introduced to the simulation. Each agent is given a random starting location and a different random goal location. Each agent sets off on their commute to work at the same time. The length of time it takes the agents to reach their destination is recorded in epochs and once all agents have completed their commute, the average amount of epochs taken is calculated. Unlike the first experiment, given there are multiple agents in the network, queues form at junctions with red lights. On top of this, when a light turns green at a junction only a certain number of agents will be let through until the light turns red again. The average amount of epochs taken for all agents to reach their destination when all agents begin their commute at the same time was 38.7 epochs.

Experiment Three For the third experiment, 10 agents are given a random starting location and a different random goal location. However, rather then all agents setting off on their commute at the same time, the agents leave for work in a staggered model. This experiment reflects a flexible working hour model and assumes that when working start times are given as a range, people will leave for work uniformly during that range. The average amount of epochs taken for all agents to reach their goal destination when the agents left for work in a staggered model was 32.5 epochs. Approximately 6 epochs less than the results from experiment two.

Although the focus of this work was not to research traffic patterns, the results of these experiments demonstrate how modelling real world events in a metaverse digital twin can lead to cost effective research which can be applied to real life settings. These experiments could of course be extended by adding in pedestrians, weather simulations, roadwork simulations etc. As mentioned, given the interoperable nature of BCT, each of these sub simulations could be created by different parties and added to the metaverse accordingly.

6.2 Results

Exp No.	**No. Agents**	**No. Epochs**
1	1	29
2	10	38.7
3	10	32.5

Table 2. Experiment Results

The associated code for this thesis [13] implements an architecture which could be used as a starting point to achieve web scale agent-based simulations. This code focuses on creating a digital twin of a city transport network. However, the underlying structure is extensible and can be refactored to accommodate other simulation types. The simulations can be run locally for testing purposes and then deployed on the Ethereum network, or a Layer 2 network, to run simulations at scale. Overall, this novel architecture allows for the creation of both distributed agents and environments as well as agent-to-environment and agent-to-agent communication on blockchain technology. Agents also have the ability to communicate with off-chain services and simulations as needed, this is demonstrated with the traffic light controller. This controller is an off-chain simulation which is integrated into the network. The distributed environment allows for a dynamic size whereby new environment nodes can be added at any point in time given blockchain interoperability. The visualisation tool created allows the state of the network to be monitored as well as the internal state of each of the agents during a simulation.

Although trivial, the results from the three experiments shown in Table 2 demonstrate that distributed agent-based models running on BCT can be used to model real world use cases. In the case of these experiments, commute times were significantly faster when agents could arrive in work over a fixed period of time rather than all agents having to arrive in work at the same time.

It is important to note that Ethereum programming languages such as Solidity are relatively new and as such lack some functionality that may be otherwise achieved with other programming languages. This issue can be mitigated by offloading complex computation to centralised services via Oracles. This is a more cost effective solution given on-chain computation is paid for via gas fees. As these programming languages mature, and network fees decrease, more and more of this computation can take place on-chain.

Overall, this proof of concept framework demonstrates that smart contracts and BCT can be used to achieve scale in agent-based models.

7 Conclusion

The goal of this thesis was to explore the use of blockchain technology to facilitate distributed agent-based simulations. Work has shown that a single machine is insufficient as models increase in complexity [25]. Existing research exploring solutions to this problem has focused on distributed Hybrid Simulations [17] and microservice architectures [11]. There has been a recent increase in popularity of blockchains, driven by networks such as Bitcoin and Ethereum. Given this, large, public, peer-to-peer networks now exist [27] realising, what is known in the Ethereum ecosystem, as a global computer. These networks allow for distributed computation at scale and are being used to create large scale virtual universes known as a Metaverse [26]. These Metaverses can be built to model a decentralised digital twin of our universe and used to simulate events and study their effects. The avatars in these environments can be human controlled or autonomous. Autonomous avatars can be implemented using a range of technologies and frameworks. This thesis implements autonomous avatars, interacting within their distributed Metaverse, as agents. Each agent has it's own internal beliefs and goals, and can interact with both it's environment and other agents in the simulation. The simulation can be extended to interact with centralised services, such as a microservice architecture, as needed.

To demonstrate this new architecture, three separate traffic simulations were created to asses the effect on traffic patterns and commute times when all agents must arrive in work at the same time versus agents having the freedom to arrive in work over an extended period of time. This toy problem demonstrates that this architecture can be used to model distributed agent-based simulations. Given that this architecture is composed of loosely-coupled smart contracts, a simulation can be easily extended to incorporate sub simulations such as weather patterns, pedestrians, or roadworks. Thus more complex simulations can be created by combining multiple sub simulations together. This overcomes the issues discussed in [25] and is inline with the research of Hybrid Simulations in [17] and the use of microservices for ABM in [11].

As updates to the Ethereum network roll out, these simulations will be able to handle scales similar to large web-based platforms such as Visa [29]. On top of this, blockchain technology offers immutable records for tracking simulation progress and analysing results, and offers extensibility through the adoption of standardised interfaces. As such, this paper can be seen as setting the foundations for the use of blockchain technology to achieve web scale agent-based simulations.

References

[1] Sameera Abar et al. “Agent Based Modelling and Simulation tools: A review of the state-of-art software”. In: *Computer Science Review* 24 (2017), pp. 13–33.

[2] Robert John Allan et al. *Survey of agent based modelling and simulation tools*. Science & Technology Facilities Council New York, 2010.

[3] Hany F Atlam et al. “A Review of Blockchain in Internet of Things and AI”. In: *Big Data and Cognitive Computing* 4.4 (2020), p. 28.

[4] Rafael Belchior et al. “A survey on blockchain interoperability: Past, present, and future trends”. In: *ACM Computing Surveys (CSUR)* 54.8 (2021), pp. 1–41.

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

[6] Vitalik Buterin. *EIP-20: Token standard*. Nov. 2015. URL: `https://eips.ethereum.org/EIPS/eip-20`.

[7] Davide Calvaresi et al. “Explainable multi-agent systems through blockchain technology”. In: *International Workshop on Explainable, Transparent Autonomous Agents and Multi-Agent Systems*. Springer. 2019, pp. 41–58.

[8] Giovanni Ciatto et al. “From agents to blockchain: Stairway to integration”. In: *Applied Sciences* 10.21 (2020), p. 7460.

[9] Nicholson Collier and Michael North. “Repast HPC: A platform for large-scale agent-based modeling”. In: *Large-scale computing techniques for complex system simulations* (2011), pp. 81–110.

[10] Nicholson Collier, Jonathan Ozik, and Charles M Macal. "Large-scale agent-based modeling with repast HPC: A case study in parallelizing an agent-based model". In: *European Conference on Parallel Processing*. Springer. 2015, pp. 454–465.

[11] Rem Collier, Seán Russell, and Fatemeh Golpayegani. "Harnessing Hypermedia MAS and Microservices to Deliver Web Scale Agent-based Simulations". In: (2021).

[12] Rem W Collier, Seán Russell, and David Lillis. "Reflecting on agent programming with AgentSpeak (L)". In: *International Conference on Principles and Practice of Multi-Agent Systems*. Springer. 2015, pp. 351–366.

[13] Conor Deegan. *Conor-Deegan/MSC-POC: Proof of Concept*. URL: `https://github.com/conor-deegan/MSc-POC`.

[14] Akshat Dhaon and Rem W Collier. "Multiple inheritance in AgentSpeak (L)-style programming languages". In: *Proceedings of the 4th International Workshop on Programming based on Actors Agents & Decentralized Control*. 2014, pp. 109–120.

[15] André Diogo et al. "A Multi-Agent System Blockchain for a Smart City". In: *The Third International Conference on Cyber-Technologies and Cyber-Systems (CYBER 2018), no. Figure*. Vol. 1. 2018, pp. 68–73.

[16] Juan Du et al. "An ontology and multi-agent based decision support framework for prefabricated component supply chain". In: *Information Systems Frontiers* 22.6 (2020), pp. 1467–1485.

[17] Tillal Eldabi et al. "Hybrid simulation challenges and opportunities: a life-cycle approach". In: *2018 winter simulation conference (WSC)*. IEEE. 2018, pp. 1500–1514.

[18] William Entriken. *EIP-721: Non-Fungible token standard*. Jan. 2018. URL: `https://eips.ethereum.org/EIPS/eip-721`.

[19] Ethereum Foundation. *Oracles*. URL: `https://ethereum.org/en/developers/docs/oracles/`.

[20] Thomas E Gorochowski. "Agent-based modelling in synthetic biology". In: *Essays in biochemistry* 60.4 (2016), pp. 325–336.

[21] Frank Hofmann et al. "The immutability concept of blockchains and benefits of early standardization". In: *2017 ITU Kaleidoscope: Challenges for a Data-Driven Society (ITU K)*. IEEE. 2017, pp. 1–8.

[22] Andrew Hudson-Smith and Moozhan Shakeri. "The Future's Not What It Used To Be: Urban Wormholes, Simulation, Participation, and Planning in the Metaverse". In: *Urban Planning* 7.2 (2022), pp. 214–217.

[23] Toru Ishida, Les Gasser, and Hideyuki Nakashima. *Massively Multi-Agent Systems I: First International Workshop, MMAS 2004, Kyoto, Japan, December 10-11, 2004, Revised Selected and Invited Papers*. Vol. 3446. Springer, 2005.

[24] Nadeem Jamali, Paul Scerri, and Toshiharu Sugawara. *Massively Multi-Agent Technology: AAMAS Workshops, MMAS 2006, LSMAS 2006, and CCMMS 2007 Hakodate, Japan, May 9, 2006 Honolulu, HI, USA, May 15, 2007, Selected and Revised Papers*. Vol. 5043. Springer, 2008.

[25] Olga Victorovna Kitova et al. "Hybrid intelligent system of forecasting of the socio-economic development of the country". In: *International Journal of Applied Business and Economic Research* 14.9 (2016), pp. 5755–5766.

[26] Lik-Hang Lee et al. "All one needs to know about metaverse: A complete survey on technological singularity, virtual ecosystem, and research agenda". In: *arXiv preprint arXiv:2110.05352* (2021).

[27] Soo Hoon Maeng, Meryam Essaid, and Hong Taek Ju. "Analysis of ethereum network properties and behavior of influential nodes". In: *2020 21st Asia-Pacific network operations and management symposium (APNOMS)*. IEEE. 2020, pp. 203–207.

[28] Adam J McLane et al. "The role of agent-based models in wildlife ecology and management". In: *Ecological modelling* 222.8 (2011), pp. 1544–1556.

[29] Daniela Mechkaroska, Vesna Dimitrova, and Aleksandra Popovska-Mitrovikj. "Analysis of the possibilities for improvement of blockchain technology". In: *2018 26th Telecommunications Forum (TELFOR)*. IEEE. 2018, pp. 1–4.

[30] Fabrizio Montesi and Janine Weber. "Circuit breakers, discovery, and API gateways in microservices". In: *arXiv preprint arXiv:1609.05830* (2016).

[31] Navonil Mustafee et al. "Purpose and benefits of hybrid simulation: contributing to the convergence of its definition". In: *2017 Winter Simulation Conference (WSC)*. IEEE. 2017, pp. 1631–1645.

[32] Stylianos Mystakidis. "Metaverse". In: *Encyclopedia* 2.1 (2022), pp. 486–497.

[33] Satoshi Nakamoto. "Bitcoin: A peer-to-peer electronic cash system". In: *Decentralized Business Review* (2008), p. 21260.

[34] Muaz Niazi and Amir Hussain. "Agent-based computing from multi-agent systems to agent-based models: a visual survey". In: *Scientometrics* 89.2 (2011), pp. 479–499.

[35] Eoin O'Neill et al. "Explicit modelling of resources for multi-agent microservices using the cartago framework". In: *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*. 2020, pp. 1957–1959.

[36] J Gareth Polhill et al. "Crossing the chasm: a 'tube-map'for agent-based social simulation of policy scenarios in spatially-distributed systems". In: *GeoInformatica* 23.2 (2019), pp. 169–199.

[37] Anshu Shukla, Swarup Kumar Mohalik, and Ramamurthy Badrinath. "Smart contracts for multiagent plan execution in untrusted cyber-physical systems". In: *2018 IEEE 25th International Conference on High Performance Computing Workshops (HiPCW)*. IEEE. 2018, pp. 86–94.

[38] Youngjun Song and Sunghyuck Hong. "Build a Secure Smart City by using Blockchain and Digital Twin". In: *Int. J. Adv. Sci. Converg* 3 (2021), pp. 9–13.

[39] Simon JE Taylor. "Distributed simulation: State-of-the-art and potential for operational research". In: *European Journal of Operational Research* 273.1 (2019), pp. 1–19.

[40] Simon JE Taylor et al. “Innovations in simulation: experiences with cloud-based simulation experimentation”. In: *2020 Winter Simulation Conference (WSC)*. IEEE. 2020, pp. 3164–3175.

[41] Antonio Tenorio-Fornés, Samer Hassan, and Juan Pavón. “Open peer-to-peer systems over blockchain and ipfs: An agent oriented framework”. In: *Proceedings of the 1st Workshop on Cryptocurrencies and Blockchains for Distributed Systems*. 2018, pp. 19–24.

[42] Roberto Tonelli et al. “Implementing a microservices system with blockchain smart contracts”. In: *2019 IEEE International Workshop on Blockchain Oriented Software Engineering (IWBOSE)*. IEEE. 2019, pp. 22–31.

[43] Rem W. Collier et al. “MAMS: Multi-Agent MicroServices”. In: *Companion Proceedings of The 2019 World Wide Web Conference*. 2019, pp. 655–662.

[44] Gavin Wood et al. “Ethereum: A secure decentralised generalised transaction ledger”. In: *Ethereum project yellow paper* 151.2014 (2014), pp. 1–32.

[45] Michael Wooldridge and Nicholas R Jennings. “Intelligent agents: Theory and practice”. In: *The knowledge engineering review* 10.2 (1995), pp. 115–152.