International Journal of Parallel, Emergent and Distributed Systems

ISSN: (Print) (Online) Journal homepage: https://www.tandfonline.com/loi/gpaa20

Flexible simulation of traffic with microservices, agents & REST

Martynas Jagutis, Seán Russell & Rem Collier

To cite this article: Martynas Jagutis, Seán Russell & Rem Collier (2023): Flexible simulation of traffic with microservices, agents & REST, International Journal of Parallel, Emergent and Distributed Systems, DOI: 10.1080/17445760.2023.2242183

To link to this article: https://doi.org/10.1080/17445760.2023.2242183

Published online: 31 Jul 2023.

INTERNATIONAL JOURNAL OF PARALLEL, EMERGENT AND DISTRIBUTED SYSTEMS
https://doi.org/10.1080/17445760.2023.2242183

Taylor & Francis
Taylor & Francis Group

Flexible simulation of traffic with microservices, agents & REST

Martynas Jagutis, Seán Russell and Rem Collier

University College Dublin, Dublin 4, Ireland

ABSTRACT

Hypermedia Multi-Agent System (MAS) Simulation is a novel approach to building agent-based simulations in which the environment is modelled as a set of linked hypermedia resources that are implemented using microservices. This paper discusses the implementation of an prototypical simulation system based on this concept and the lessons learnt in the process.

ARTICLE HISTORY
Received 28 February 2023
Accepted 25 July 2023

KEYWORDS
Traffic simulation; microservices; REST; multi-agent systems; agent-based modelling

1. Introduction

Agent-Based Modelling (ABM) is an approach to studying the behaviour of complex systems that can be modelled as a population of individuals [1]. These agents are designed to simulate low level behaviours and interact through a shared environment. The overall behaviour of the system is an emergent property of the interactions between the individuals within this environment.

A comprehensive review of the state of the art in Agent Based Modelling tools is presented in [2]. The review highlights that there are really two main approaches to implementing agent-based simulation: desktop simulations, such as NetLogo [3], REPAST [4] and Mason [5]; or distributed simulations, such as RISE [6], REPAST-HPC [7] and Distributed Mason [8]. The distributed simulation community [9] focuses on the development of techniques that can be deployed on high-performance computing clusters. Taylor [10] reviews DS through the lens of Operational Research, highlighting the prevalence of bespoke implementations tailored to specific scenarios, the lack of model reuse and the need for well-designed frameworks. In fact, it is not possible to directly transfer a model from the desktop version to the distributed version of the same tool.

More recently, ABM is being applied to problem domains that are increasingly complex, to the point that it is becoming impossible to capture all the desired properties in a single model [11]. One approach, that is increasingly used in Operational Research [12] and Socio-Environmental Systems [13], is Hybrid Simulation [14]. This approach aims to combine multiple interconnected sub-simulations, potentially implemented using a diverse set of modelling techniques [15]. However, current tools and frameworks are considered inadequate [1] due partially to their lack of support for interoperability [14].

This paper argues that (hybrid-)simulations should not be built on monolithic architectures and homogeneous technology stacks, but should instead be implemented as loosely-coupled collections of reusable components, written using diverse programming languages and frameworks, that are designed to be deployed at scale. In essence, it argues that HS should be implemented using a microservices architecture [16]. This approach is explored from the perspective of Agent Based Modelling, we believe it can be extended to cover the integration of ABM with other forms of simulation.

CONTACT Rem Collier rem.collier@ucd.ie

Hypermedia Multi-Agent System (MAS) Simulation [17] builds upon ABM simulations by modelling the environment as a set of linked hypermedia resources. In this novel approach, the resources within the system function as micro-environments that are connected to other related micro-environments by a set of hyperlinks. The overall design of the system is based on the microservices architecture [16,18] and the interactions between the micro-environments are implemented using REpresentational State Transfer (REST) [19]. This approach can be expanded to include the use of ontologies to provide semantic meaning to the resources and hyperlinks within the system and affordances to allow agents to reason about the actions that can be performed on the resources [20].

Traffic Simulation Systems (TSS) is a mature area of research with the aim of improving the planning, design, and operation of transportation systems [21]. Simulations can vary in both scope and scale. In some situations, the passage of vehicles through the system can be represented as mathematical formulae, similar to throughput in networks, while in others, the position, actions and intentions of individual vehicles are represented at each instant. The trade-off between the level of detail and the computational cost of the simulation is a key consideration in the design of a TSS. Several modelling applications have made use of ABM to simulate the behaviour of individual users of the transport network being simulated [22,23].

This paper expands upon the original proof of concept of the Hypermedia MAS Simulation, which took the form of a TSS, and is described in [24]. The features of the system described in this paper are not novel and are typically included within a much larger feature set that is available in commercial or open source TSS. The novelty of this work is in the way we construct the simulation through the combination of microservices, hypermedia systems, linked data, multi-agent systems, and affordances in order to create a decentralised simulation that can scale while still delivering support for reusability and extensibility.

The Hypermedia MAS Simulation approach is based on the principle that a simulation can be constructed from a set of loosely coupled sub-simulations. Each sub-simulation is implemented as a microservice that can be reused in other simulations. These individual sub-simulations work together to achieve the global aims of the overall simulation. The use of microservices allows for the simulation to be constructed from a diverse set of modelling techniques and languages, rather than being restricted to a single language or framework. Components can be developed independently and integrated into the simulation at a later date as long as they can be accessed through a RESTful interface.

The agent part of the simulation also resides in one (or more) microservices. Conceptually, agents are integrated using the Multi-Agent MicroServices (MAMS) architectural style [25]. There are 2 different implementations of agents in our prototype: one is a pure Java implementation written using Spring Boot[1], and the second is based on a prototype MAMS framework that is based on the ASTRA agent programming language [26,27] and CArtAgO [28]. Details of this framework are presented in [29]. The former approach uses no agent technologies, while the latter uses a dedicated agent programming language that comes with a built in agent run-time. Further details of these implementations are presented in Section 3.3.

Linked Data and its use within the Web of Things (WoT) [30] forms the basis for the approach to interoperability between sub-simulations. WoT developed the Thing Description standard [31] which is a machine readable document that describes the capabilities of the devices that have been deployed. This allows clients to reason about how (or whether) to use them. Analogous descriptions, called *Simulation Descriptions*, could be developed for sub-simulations in order to enable the interoperability and reuse of the sub-simulations. These would describe the inputs and outputs of the sub-simulation as well as the nature of the simulation environment. This description could then be utilised by the system to automatically handle data transformations when required as semantic meaning will be available for the inputs and outputs. The provided semantic meaning could be leveraged by agents to reason about the actions that can be performed on the entities within the simulation environment. The use of a Hypermedia MAS [32] approach would allow agents to navigate the hypermedia space and reason about the information that they perceived.

The embodiment of agents within the environments of the available sub-simulations is a key feature of the this hybrid simulation approach. These agents are expected to control the actions of the entities being represented in the simulation. The need for agents to be capable of interacting with a diverse set of environments presents a significant challenge. In addition to this, the agents may require implementations of greater complexity as a result of the requirement to interact with multiple sub-simulations as they will need to navigate and transition between them.

The 'Keep It Descriptive, Stupid' principle described in [33] argues for the simulation of richer, more realistic models. This principle aligns with the increased complexity of agents that is expected in this approach. Adam and Gaudou [33] argue that the social simulation community needs to adopt BDI style models, tools and programming languages.

Affordances can be used to facilitate agent interactions in diverse simulation environments. Affordances are information perceived from the environment, this information details the possible actions that can be performed. Agents capable of understanding and reasoning about affordances can be designed with a higher level of abstraction. This removes the limitation of hard coded actions or plans for specific circumstances and enables agents to reason about the actions that can be performed in a given situation. Proper documentation of these affordances as hypermedia documents published within the simulation system would allow agents to discover the entities that they are capable of controlling or interacting with regardless of their location.

2. Related work

This paper describes work that combines traffic simulation as well as multi-agent systems research. As such, the related work is presented from both perspectives. First research in the area of traffic simulation is discussed at a high level. This is then followed by a discussion of related work in the area of multi-agent systems.

2.1. Traffic simulation

Traffic simulation systems are an important tool to aid in the planning, design and management of road infrastructure. Decades of increasing urbanisation and levels of car ownership have strained the transportation networks within many cities [34]. These systems help to explore the complexities of the networks being simulated and the impact of different variables on the network. As such research into traffic simulation systems has been ongoing for many years.

Systems that simulate traffic networks are available in large numbers [35]. This includes commercial systems as well as open source solutions. They can often be fundamentally different in terms of efficiency or features, but generally can be categorised based a number of key characteristics, traffic flow behaviour, traffic flow models, and representation.

2.1.1. Traffic flow behaviour

is the way in which vehicles are modelled in the simulation. This is considered both from the perspective of variance of behaviour within the population as well as with respect to the types of vehicles that are modelled. The behaviour of vehicles within most systems is modelled in a homogeneous manner [35]. That is that there are typically a limited number of vehicle types and that the behaviour of each vehicle is predictable. This is in contrast to heterogeneous behaviour where vehicles are modelled with a greater variance in behaviour and types [36]. For behaviour, vehicles may have a greater tendency to reroute based on traffic of to accelerate or decelerate more quickly, for types, the simulator may consider pedestrians, cyclists as well as different classes of vehicles. This approach is made available by only a small number of systems such as Sumo [37].

2.1.2. Traffic flow models

can typically be described as Macroscopic, Microscopic, or Mesoscopic. Macroscopic models are the simplest and most computationally efficient. They model the overall flow of vehicles on the network, rather than the behaviour of individual vehicles [38]. This is typically implemented using a statistical approach. Microscopic models are the most computationally expensive and are based on the simulation of individual vehicles (agents) and include representing the position and velocity of all vehicles on the road network [37,39]. This allows for a greater level of customisation of the behaviour of individual vehicles and the ability to model heterogeneous behaviour. This approach is becoming more relevant as the need to include the simulation of autonomous vehicles becomes more important [40].

Mesoscopic models attempt to benefit from the more individual representation of vehicles in some places while also benefiting from the higher level calculations in others [37,41]. This can be achieved using strategies such as grouping vehicles such that they move together on a road, or by simplifying the dynamics of individual vehicles.

The amount of detail required by a simulation system will have a large influence on the choice of traffic flow model. If a high level of detail is required, then the use of a macroscopic model may not be sufficient. In the case where a high level of detail is not required, then the use of a microscopic model may be wasteful of both resources and of time.

The manner of *Representation* for concepts such as position and velocity can also vary in these systems. One approach is to model the transport network as a collection of cells that the vehicles can inhabit and move between [42,43]. This limits the amount of detail that needs to be modelled, but can simplify the process of calculating the movement of vehicles. This is called a *Discrete* model.

Alternatively, the network can be modelled as a continuous space where vehicles can move freely [37]. This allows for a greater level of detail to be modelled, but requires a greater level of computational power to calculate the movement of vehicles. This is called a *Continuous* model. Discrete models are often used in the case where the network is relatively simple and the level of detail required is low.

2.2. Multi-agent systems

Work on the use of the Belief-Desire-Intention (BDI) architecture [44] in social simulation [45,46] highlights the potential quality improvements that cognitive architectures bring to simulation. Early work in the area was argued to be limited in the level of agency displayed by the software entities in simulations [47]. The argument put forward by Edmonds and Moss that the 'Keep It Descriptive, Stupid' (KIDS) principle is more suitable than the 'Keep It Simple, Stupid' (KISS) principle for the definition of agents within simulation systems reflected similar sentiments [48].

Adam and Gaudou argue that the increased computing resources currently available, along with further technological development, means that the use of richer cognitive agent architectures will facilitate the creation of more nuanced models. Over time, the need to simplify models in an effort to maximise the performance of the simulation will become less and less important [33].

Ciortea et al. proposed the concept of *Hypermedia Aware Agents* [32]. The concept is underpinned by the *Hypermedia MAS*; an approach to building dynamic, open and long-lived MAS [49,50] that are designed to inter-operate seamlessly with the World Wide Web.

The concept was expanded to outline a vision in which the hypermeda fabric of the web is used as an environment into which the agents are integrated [51]. This shared hypermedia environment would be based on the open standards of the Web. Agents, in such an environment, could use hyperlinks and hypermedia controls to discover and interact with entities and other agents regardless of their location. The entities with which the agents interact could be physical services or devices. The hypermedia controls could be published through hypermedia documents that are adapted based on the state of the underlying entities.

A clear understanding of the relationship between *Agents and Microservices Architecture* is critical to the approach advocated in this paper. Recent experience in industry has shown that the use

of microservices can be a powerful approach to building distributed systems that scale. The ability to build and deploy microservices independently make it easier to scale parts of the application as needed. The combination of microservices with agents was first explored in bespoke applications built for the IoT domain [52,53]. An approach to this combination of ideas was presented as an architectural style called *Multi-Agent MicroServices (MAMS)* [25]. The ASTRA agent programming language [26,27] was used to implement and deploy a prototype of the MAMS approach [29]. More recently, the MAMS approach has been used in the development of a semantic web service composition system [54] as well as a group planning system [55].

2.2.1. Affordances

within agent systems are a means of representing the potential actions that an agent can perform in a given environment and context. The concept originates in ecological psychology as a means of representing the relationship between objects in an environment and the potential actions that an agent (human or otherwise) may perform with those objects [56]. Affordances enable the possibility for agents to be implemented at a higher level of abstraction that would otherwise be possible. Agents can potentially have more general plans that can be applied in a variety of situations instead of requiring that actions be hard coded into the plans.

Agent-oriented programming has been undergoing a transition to the point where the environment is now viewed as an explicit part of a multi-agent system [57]. The effect of this change can be seen in the development of systems such as the Environment Interaction System (EIS) [58] and the CArtAgO multi-agent system [28]. These systems provide an abstraction of the environment such that it can be used across agent platforms.

Simulation systems have already been developed that integrate environment abstraction. The JaCaMo-sim platform [59] makes use of environments developed in CArtAgO and as such allows the use of agents within simulations. The Web of Things (WoT) environment [32] is an example of an environment that has been developed to use affordances within a CArtAgO workspace. While both of these systems are based on the use of CArtAgO, the combination of both affordances and simulation has not yet been reported.

Implementation of affordances within existing agent based simulation systems has typically varied. Affordances and effects were paired together in the modelling of human behaviour within an emergency evacuation scenario [60–62]. Affordances used in combination with a scalar value representing fitness have been used in path planning simulations [63]. Others have proposed a representation in which an agent identifies its own affordances and may determine the possible actions within a traffic simulation environment [64]. More recent work has proposed the adoption of a formal affordance schema for use within agent-based modelling systems [65,66].

3. A prototype traffic simulator

The vision originally described in [17] considered simulations to be a composition of two types of interacting microservice. The first type of microservice, *environment microservices*, host pieces of the environment to be simulated. The second type of microservice, *agent-oriented microservices*, implement the intelligent behaviour of the individuals driving the actions of the entities in the simulation.

The implementation of environment microservices can be completed using any technology that can be exposed as a RESTful service. In this regard, we consider the environment microservices to be heterogeneous or at least capable of being implemented using heterogeneous technologies. In this example, all the environment microservices are implemented in Java not because it is a requirement, but simply out of convenience. It is expected that such systems mature, components would be added (or existing ones replaced) with implementations in whichever technology is most appropriate.

The agent-oriented microservices were developed and tested using two alternate implementations. The first implementation is built using the ASTRA [26] platform to define the agents and their behaviour. The second implementation made use of Java to similarly define the agents and their

behaviour within the simulation. Testing using both implementations was completed to ensure that the simulation system was capable of supporting heterogeneous implementations of the agent-oriented microservices.

3.1. Overview of implementation

Figure 1 outlines the architecture and some interactions of the prototype simulation[2]. The primary environment microservices are highlighted in green. The *Traffic Simulator* manages the movement of vehicles on the road and through junctions. The *Home Simulator* and *Work Simulator* operate as sinks and sources of agents, these are destinations that the agents will navigate to and from. Currently, these simulators are very simple, but their inclusion allows for the simulation of agents that can transition between multiple connected micro-environments.

Secondary environment services are shown in blue. The *Clock Service* provides a global discrete time to the environment microservices. This is shown in the figure as a PUT (`/step`) request that is sent to each of the environment microservices informing them of the current time. The *Traffic Lights* service implements the scheduling of the traffic lights at different junctions in the traffic simulator. The addition of nodes to the traffic light service is done by the traffic simulator at the point of initialisation. These nodes then report their status to the traffic simulator at the beginning of each time step.

The remaining microservices are shown in blue. The *Management Service* is used to configure and manage the execution of the simulation. The *Driver Service* is used to decide the actions of the entities within the simulation. Although the name may suggest that the agents are configured only for the traffic simulator, their implementation within this service also governs their actions within the home and work simulators.

3.2. Simulation of traffic

The prototype simulator is initiated in a state where all agents are located within the home simulator. At the appropriate discrete time, the agents will initiate their journey to the work simulator by

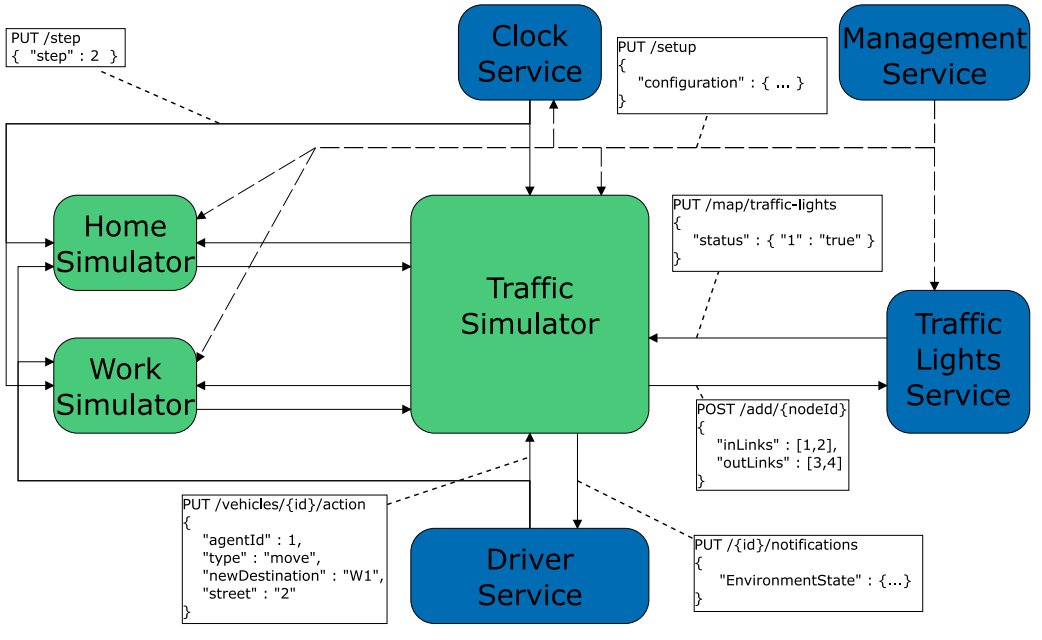

Figure 1. Overview of Simulation Architecture.

transitioning into the traffic simulator. The agent then navigates through the traffic simulator to the appropriate point and transitions into the work simulator. Figure 1 illustrates a sample of the interactions that take place between the microservices. The interactions are in the form of JSON documents which are delivered as the payload of HTTP messages. The interactions are designed based on the REST architectural style.

Entities within the system all have identifiers that are locally unique for their type, but that are not globally unique within the simulator. For example, an agent, street and junction may all share the same identifier, but no two agents will be allocated the same identifier. The separation of environment into multiple services as well as the use of specific URIs for each interaction removes any need for global identifiers.

When joining a simulation environment, agents pass a URI to the micro-environment service which enables information to be passed back to the agent. These webhooks are associated with each individual agent and are used to allow the simulation environment to send updates. These updates concern the current state of the body of the agent in the simulation as well as the relevant perceptions of that body. The state update is sent to the agent as a PUT request at the beginning of each time step. Within this updated state is a list of actions that the agent can perform. This basic implementation of the concept of affordances demonstrates that the approach is viable.

Figure 2 shows an example interaction between an agent and the traffic simulator. The agent receives information about the state of the simulated environment as well as percepts and affordances. The affordances received by the agent represent the possible actions the agent can perform. In response, the agent informs the simulator of the action to be performed by the entity it represents. This is completed by sending a PUT request to the traffic simulator with the action to be performed as well as any required parameters specified within the payload.

The list of all possible affordances are:

- *Accelerate:* increase the current speed of the vehicle;
- *Decelerate:* decrease the current speed of the vehicle;
- *Enter:* enter an intersection with the intention to exit on a specified street;
- *Leave:* leave an intersection via the street specified;
- *Move:* continue travelling along the street at the same speed;
- *Plan:* generate a new route to a given destination; and
- *Wait:* do nothing.

The affordances received by the agent are based on the current state of the simulation. For example, the `Enter` affordance is only received when the vehicle is at the entrance of an intersection, the `Leave` affordance can only be received when the vehicle has already entered an intersection. Some more general affordances such as `Wait` and `Plan` are always available to the agent. The response in

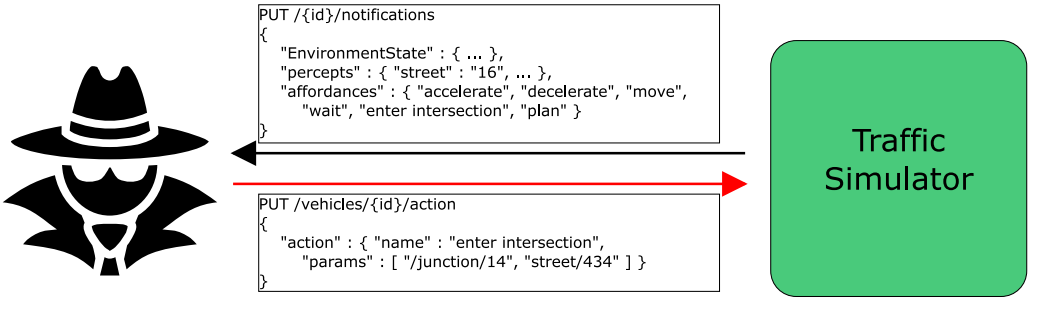

Figure 2. Example interaction between agent and traffic simulator.

Figure 2 details the agents command to enter the vehicle it controls into a junction while specifying that the intended exit point of the junction is street 434.

The traffic simulator uses the cellular automata approach to representation and execution. In this approach the road is discretised into cells of set size, where each cell will be empty or contain a vehicle at any given point in time. Junctions connecting two or more roads are implemented as queues of vehicles. This approach to the simulation of traffic is designed to reduce the complexity and computational requirements of the simulation while involving the more human-oriented aspects of driving [67].

The traffic simulator is responsible for the movement of vehicles through the junctions and along the roads. In order to ensure that the individual sub-simulations remain synchronised, the simulations operate on a discrete time model controlled by the clock service. The home and work services are minimal, these are added to allow the opportunity to allow agents to transition across multiple environments within the simulation. Within these simulations, the only context the agent receives is the current time, and the only affordance they are provided is for the *Plan* action. The agents in these simulations wait for the correct time, then transition from the home simulator to the traffic simulator by making use of the plan action to set the desired destination. The process is then later reversed when the correct time comes for the agent to leave the work simulator.

The home and work simulators are linked to individual junctions in the traffic simulator. When an agent transitions from the home simulator to the traffic simulator, they are placed at the relevant junction. When the agent within the traffic simulator arrives at the correct junction, they are automatically transitioned to the desired simulator.

3.3. Implementing agent drivers

One of the advantages that our approach brings is that it decouples the implementation of the environment from the implementation of the agents. This means that we can write our agent code in any language that is able to implement the protocol outlined in Figure 2. To illustrate this, the prototype system presented in this article includes two examples of how to integrate an agent driver. The first approach is an object-oriented implementation written in Java and the second approach is an Agent-Oriented Programming (AOP) solution that has been built using the Multi-Agent MicroServices (MAMS) architectural style. The two approaches are illustrated in the sections below. This article argues that the latter approach offers a better level of abstraction for defining agent behaviours. This is especially so when linked-data representations that can be transformed directly to knowledge are used. That said, this section demonstrates the flexibility of the approach as the designer is free to choose any implementation strategy they feel is appropriate and the cost of integrating that strategy is minimal.

3.3.1. Java drivers

The procedural approach has been implemented using the well-known Java Spring-Boot framework[3]. The prototype application is quite simple. It defines a single endpoint `/{id}/notifications` that receives the state update from the environment as was shown earlier in Figure 2. It is expected that the consequence of a PUT request being submitted to this endpoint is that the environment description will be passed to an agent implementation that will identify the next action to be performed. Again, as is shown in Figure 2, this action is submitted to the simulator via a second PUT request.

Figure 3 illustrates this through a simplified implementation of the Java method associated with the notification endpoint. Most of the decision making logic is hidden in the `generateAction()` method which takes the state of the environment as an input and returns the selected action.

3.3.2. MAMS drivers

The second approach to implementing drivers uses an AOP language. Specifically, the agents are programmed using the ASTRA programming language, an AOP language that is based on the abstract AgentSpeak(L) language [44]. ASTRA is an event driven language where behaviour is encoded using event-context-plan triples, known as *plan rules*. Example plan rules can be seen in Figure 4. They are the

```
@RestController
public class DriverController {
  @PutMapping("/{id}/notifications")
  public void sendAction(@PathVariable Long id,
                         @RequestBody EnvironmentState state){
    // Make decision
    Driver driver = driverService.getDrivers().get(id);
    Action action = driver.generateAction(state);

    // Prepare PUT request
    RestTemplate restTemplate = new RestTemplate();
    HttpHeaders headers = new HttpHeaders();
    headers.setContentType(MediaType.APPLICATION_JSON);
    HttpEntity<Action> body = new HttpEntity<>(action, headers);

    // Send PUT request
    restTemplate.exchange(state.getWebhook(),
                          HttpMethod.PUT, body, Void.class);
  }
```

Figure 3. Spring Boot Agent Implementation.

statements that are prefixed by the keyword `rule`. Events can model environmental change (known as belief events) or decision points (known as goal events). The latter type of event is used in the example code. Goals have a long history in Multi-Agent Systems and AI but here can be viewed as representing decision points. The idea behind ASTRA (and AgentSpeak(L)) is that an agent behaviour is triggered by an environmental event indicating that something undesirable has occurred in the environment. This event is matched to a plan to that is selected contextually based on the current state (beliefs) of the agent. Because the environment is dynamic the plan parts are written as partial plans and goals identify decision points where the agent must choose how to proceed. Choosing how to proceed involves matching (again contextually) the goal event generated by the adoption of a goal in the current plan to a plan rule and then adopting the plan part as a sub-plan of the overall plan.

You can see an example of this pattern of behaviour in the example program shown in Figure 4. The pattern starts with the handling of the `!updatedObject(...)` goal in the second rule. The plan part of the plan rule defines two sub-goals `!decide(...)` and `!act(...)` which must be achieved in sequence. The last two rules highlight two possible sub-plans for achieving the `!decide(...)` goal. The agent will choose only one of these options based on what its current state is. The selection conditions are expressed by the context part which appears after the colon (:) and before the opening brace () of the plan. This is the equivalent of the `generateAction()` method defined in the Java Drivers example. The second goal is used to send the chosen action to the server. This takes the form of a low level `!put(...)` goal which is implemented as part of the Multi-Agent MicroServices (MAMS) framework.

MAMS is visible in two parts of the example code. The latter place is in the rule associated with the `!act(...)` goal where a `!put(...)` goal is adopted to submit the action to the server. The representation actually sent to the simulation has been simplified for readability. The former place where MAMS is visible is in the rule that handles the `!main(...)` goal. The actions specified in this rule connect the agent to the MAMS infrastructure and create a resource that is exposed on the web under the `/{agent-name}/notification` URL. The simulation service sends the environment state to the agent in the same way; by updating this resource using a PUT request. Upon the processing of a new PUT request, the underlying MAMS infrastructure generates the `updatedObject(...)` goal to trigger a response from the agent.

Finally, the example includes two logical inference rules (denoted by the `inference` keyword). These rules are used to provide abstract (derived) representations of the environment state to improve

```
agent Driver extends mams.PassiveMAMSAgent {
  module ObjectAccess oa;

  inference atIntersection(EnvironmentState state) :-
      oa.isFalse(state, "atIntersection") &
      oa.getInt(state, "vehicleSpeed") > 0;

  inference isStopped(EnvironmentState state) :-
      oa.getInt(state, "vehicleSpeed") > 0;

  rule +!main(list args) {
     MAMSAgent::!init();
     MAMSAgent::!created("base");
     PassiveMAMSAgent::
        !itemResource("notifications", "EnvironmentState");
  }

  rule +!updatedObject(EnvironmentState state) {
    !decide(state, oa.getString(state, "type"), string action);
    !act(state, action);
  }

  rule +!act(EnvironmentState state, string action) {
    !put(oa.valueAsString(state, "webhook"),
        "{ 'action':'"+action"'}", HttpResponse response);
    if (!httpUtils.hasCode(response, 200)) system.fail();
  }

  rule +!decide(EnvironmentState state, "traffic", string action)
        : time(t) & atIntersection(state) {
    action = "move";
  }

  rule +!decide(EnvironmentState state, "traffic", string action)
        : time(t) & isStopped(state) & canAccelerate(state) {
    action = "accelerate";
  }
}
```

Figure 4. ASTRA-MAMS Implementation.

the readability of the plan rule contexts. the contexts environment state into beliefs that are matched against the context of the `!decision(...)` rules. The environment state itself is encapsulated as a Java object that is an instance of the `EnvironmentState` class.

3.4. Walkthrough

Executing the simulation is a process that requires the performance of a set of steps in sequence. If services are started in the wrong order or if the configuration is incorrect, the simulation will not function correctly. In order to execute the prototype, the microservices must be started in the following order: *Clock Service*, *Micro-Environment Service*, *Traffic Simulator*, *Home Simulator*, *Work Simulator*, *Driver Service* and *Management Service*. The management service once started will initiate the simulation.

Figure 5 shows a screenshot taken during the operation of the simulator. The simulation was designed as a small scale test of the underlying system. The network being simulated was made up of 31 streets which intersected at 14 points. Dispersed throughout the network were a number of homes

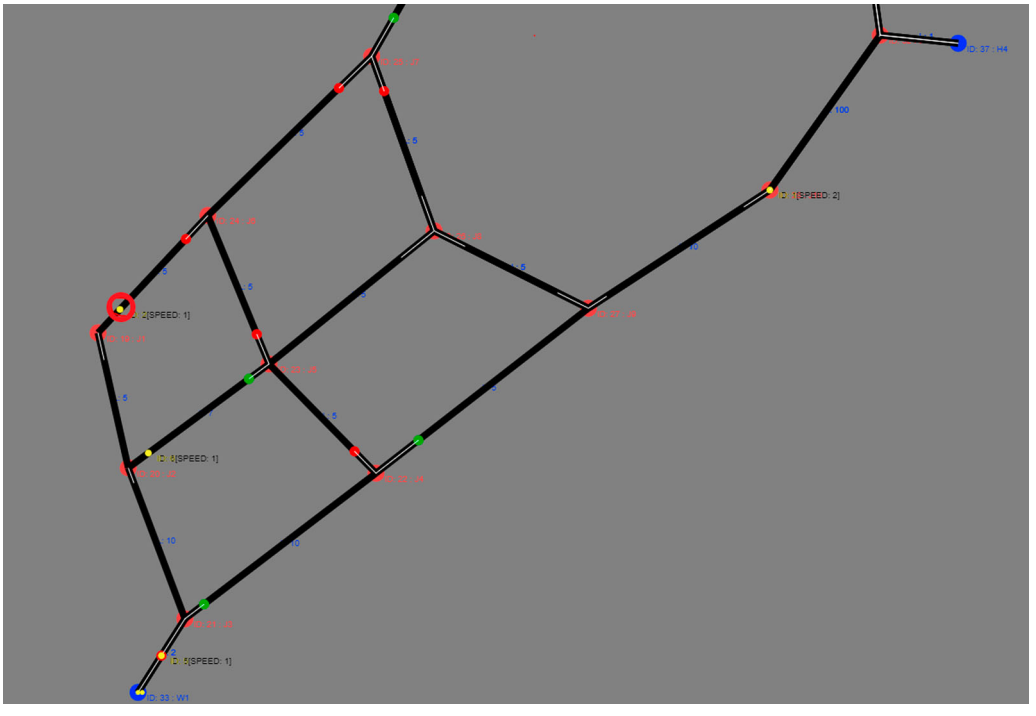

Figure 5. Screenshot of the simulation in progress.

and workplaces that agents would navigate between. Once initialised, agents exist within the home simulation and wait for the correct time to transition into the traffic simulation. Once in the traffic simulation, agents navigate to the relevant junction and transition into the work simulation. The process is reversed when the correct time comes for the agent to leave the work simulator.

The duration that the simulation executes for is specified by the management service. During execution, rudimentary statistics are gathered relating to the travel times of the vehicles. The statistics are not as detailed as those that would be gathered by a traditional traffic simulation system. However, there is no reason that the system could not be extended to gather other statistics that are required for the specific application.

4. Lessons learnt

The development of this prototype was a part of the overall process of developing and evaluating this vision for agent-based modelling. As a consequence of the development and evaluation of the prototype and the natural extension of the ideas and concepts underlying the vision, a number of lessons were learnt. In this section, we describe the most evident lessons learnt.

4.1. The importance of URLs as identifiers

Traditionally, some form of integer is used as a unique identifier to refer to different nodes or edges in a graph. Such identifiers work well in closed systems where all the components are known in advance because it is possible to encode how to use those identifiers across all components. In the case of our prototype, the use of identifiers refers to the construction of URLs needed to interact with specific environment resources defined in the simulation model. For example, when the agent uses a PUT request to submit the next action to be performed or when an agent is transferred from a home environment to a junction in the traffic simulator environment. Here, the knowledge of how to construct URLs that

refer to the specific streets or junctions must be replicated across the home, work, management and agent microservices. While the knowledge is relatively trivial – the base URL that is common to all streets or junctions – the key issue is the replication of the code as this increases the coupling of the system components. The introduction or removal of an environment resource requires that the developer modify all microservices that refer to the new/existing microservice. This task only increases in complexity as the system scales.

The issue of constructing URLs is not new. In fact, it is one of a number of issues raised by Fielding [19] when describing REST. In particular, he describes best practice for the implementation of REST based applications through the concept of Hypermedia As The Application Of Engine State (HATEOAS). Fielding argues that the construction of URLs should be the responsibility by the microservice (he used the word server here) that the URLs refer to and should be provided by that service as part of the resource representation.

In the prototype, we did not follow this design principle, but instead used integers as unique identifiers in the resource representations returned by the three environment microservices. This is evident from the simplified resource representations shown in Figure 1. Our motivation was to keep the design as simple as possible, but it became clear towards the end that the resulting system components had closer coupling that we would like. It was particularly clear in the design of the agents as they interacted with all the other services.

4.2. Semantic descriptions

The lack of semantic descriptions of the environment became a barrier to the addition of new functionality. The underlying layout of the traffic network being simulated was not available to the agents. As a result, it was impossible for agent to plan their route through the network. As a consequence, the route planning functionality was implemented as part of the traffic simulation service. This approach is not ideal because it is not possible to implement alternative route planning strategies.

A semantic representation of the traffic simulation that could expose the current simulation state would be a better approach. Using the appropriate technologies would allow the construction of a shared knowledge graph that agents could explore directly. This would make it possible to implement functionality, such as route planning, within the agent. This allows for heterogeneous route planning strategies, such as the Q-learning approach advocated in [68], while also allowing centralised approaches to be used when they are considered appropriate.

4.3. Decomposition is good

In this prototype, the traffic simulation implements the entire simulation of the traffic network and other functions such as route planning. This seemed like a good idea in the original design, as the traffic simulation was viewed as one of three types of environment that the agents could inhabit.

On reflection this has a number of drawbacks: combining the implementation of junctions and streets into a single service enforces a single (or at least fixed) set of junction and street models be used. In the prototype, we implemented just one junction model and one street model. In practice, it may be necessary to use different models for different parts of the simulation. For example, a multi-lane street models might be designed that include support for overtaking, or a one-way lane models to support one way systems. It became clear that we had not decomposed our traffic simulation sufficiently. A better level of decomposition would be at the street/junction level; allowing the integration of multiple street/junction models by implementing a microservice for each model. An obvious issue in this is the increased complexity described in Subsection 4.1. This approach would also not fit well with the use of a graph database such as Neo4J.

One solution is to use URLs to implement the graph structure. In such an approach, junctions would be implemented as individual environment resources and would include URLs to the street environment resources that connect to it. Similarly street environment resources would include URLs to the

associated junction environment resources. The result is a decentralised graph implementation based on the URLs themselves.

4.4. The potential of knowledge graphs

While reflecting on the lessons learnt in the previous three sections, another idea emerged. The combination of URLs and semantic representations are the building blocks of knowledge graphs [69]. When this is combined with the decomposition of the traffic simulation into junctions and streets, an interesting possibility emerges: the creation of a distributed knowledge graph of the simulation environment. Such a knowledge graph could be used by the simulation agents to explore the environment. For example, in our prototype, route finding is implemented using a shortest path algorithm provided by Neo4J. In a knowledge graph centric environment, the agent could perform route finding by simply exploring the knowledge graph following the URLs connecting streets to junctions and vice versa. An example of such an approach can be found in [68] where agents are implemented that use Reinforcement Learning to explore a semantically defined maze environment.

Knowledge graphs can not only enable discovery of the structure of the environment, but also can be used to provide additional knowledge necessary for the agents to operate effectively in the environment. For example, a more complex work environment could include descriptions of the main tasks associated with each role. This offers the potential for an agent with no prior knowledge of that particular workplace learning how to perform a given job. Here we use the term 'learn' in its most general sense which can be realised through some form of plan sharing, the application of reinforcement learning from first principles or even some form of transfer learning.

4.5. Other lessons

Other lessons have been learnt that are more specific to the prototype. For example, the use of the *Plan* action to trigger transition from a home/work simulation into the traffic simulator only works if there is a traffic simulator. In the context of the avoiding monoliths lesson, perhaps a generic *Leave* action would be more appropriate. Also, the use of the *Move* action does not make much sense in the context of the *Accelerate* and *Decelerate* actions. A better model would be to accelerate to the correct speed and then to *Wait* if there is nothing to do. Similarly, the need to perform an *Enter* action at the end of a street does not make sense, it is surely more appropriate to expect that the car will automatically transition into the intersection when it reaches the end of the street.

5. Conclusions

This paper presents an overview of a prototype agent-based traffic simulator based on microservices and REST. The development of the prototype demonstrates the potential of the approach to building simulations based on Agent-Based Modelling.

One particularly nice feature of the approach is the flexibility afforded in choosing how to implement the agent part of the simulation. By adopting HTTP as an interface between the agent and its environment, it is possible to use any programming language or agent framework that is able to interact with services via HTTP. This is illustrated through two prototype agent implementations. Section 3.3.1 introduces a simple Java based agent implementation that uses the well-known Spring-Boot framework to provide the HTTP integration. The second approach, described in Section 3.3.2 uses the less well known Multi-Agent MicroServices (MAMS) framework which is built on top of the ASTRA agent programming language.

The goal of this work is to explore the potential of the approach. We have not done any significant performance analysis. We do expect that the decomposition of the simulation into a set of microservice will increase network overheads and could have an impact at scale. One interesting avenue of future

research would be to explore possible techniques for managing this increased load. Potential strategies could include the use of mobile agents techniques to balance the network load or the use of the duplex mode offered in HTTP/2. Even with such optimisations, we do not expect that any Hypermedia MAS Simulation would be able to compete with a bespoke simulation developed for and deployed on a cluster. This was never our goal for this work.

The experience of building and testing the prototype has influenced our original vision for agent-based modelling. The importance of semantic meaning and hypermedia links was made evident. The adoption of Semantic Web and Linked Data technologies has three potential benefits:

- The use of linked microservices would allow the decomposition of complex environments into a set of simpler environments implemented as a decentralised suite of microservices that form a *decentralised linked environment*. This has the potential to *enhance the configurability and reusability of the environment components*. For example, a new street model can be introduced by deploying it as a microservice, creating necessary environment resources and linking them to the relevant junction service endpoints.
- The use of linked data technologies should further simplify the interface between the environment microservices and agent frameworks that operate with knowledge/beliefs (such as ASTRA) allowing the *direct ingestion of knowledge from the environment* removing the need to provide custom abstractions such as the inference rules defined in Section 3.3.2. This will simplify further the use of agent frameworks making them a more viable option for agent-based simulation. It also opens up the possibility of hybrid agent simulations where different agents within the simulation are implemented using different technologies, but are able to inter-operate through the same shared environment.
- The combination of linked data and microservices leads to the potential to create a decentralised knowledge graph of the simulation environment. By embedding links to the environment descriptions inside the environment state representations, it becomes possible for agents to access their local knowledge graph. This offers the potential for implementing agents that are able to *combine the traditional local state with access to global knowledge of their environment*.

Future work will focus on the development of the decentralised linked environment approach, the integration of semantics and the creation of a larger and more complex prototype that will allow us to better evaluate the potential of the linked data approach and to enable its evaluation through larger and more complex scenarios.

Notes

1. http://spring.io
2. Source code: https://gitlab.com/mams-ucd/examples/microservice_traffic_simulator
3. https://spring.io

Disclosure statement

No potential conflict of interest was reported by the author(s).

References

[1] Polhill JG, Ge J, Hare MP, et al. Crossing the chasm: a 'tube-map' for agent-based social simulation of policy scenarios in spatially-distributed systems. GeoInformatica. 2019;23(2):169–199. doi: 10.1007/s10707-018-00340-z
[2] Abar S, Theodoropoulos GK, Lemarinier P, et al. Agent based modelling and simulation tools: a review of the state-of-art software. Computer Sci Rev. 2017;24:13–33. doi: 10.1016/j.cosrev.2017.03.001
[3] Tisue S, Wilensky U. Netlogo: a simple environment for modeling complexity. In: International Conference on Complex Systems; Boston, May 16–21; Vol. 21. Citeseer; 2004. p. 16–21.
[4] North MJ, Collier NT, Ozik J, et al. Complex adaptive systems modeling with repast simphony. Complex Adaptive Syst Modeling. 2013;1:1–26. doi: 10.1186/2194-3206-1-3

[5] Luke S, Cioffi-Revilla C, Panait L, et al. Mason: a multiagent simulation environment. Simulation. 2005;81(7):517–527. doi: 10.1177/0037549705058073
[6] Al-Zoubi K, Wainer G. Rise: a general simulation interoperability middleware container. J Parallel Distrib Comput. 2013;73(5):580–594. doi: 10.1016/j.jpdc.2013.01.014
[7] Collier N, North M. Repast HPC: a platform for large-scale agent-based modeling. Large-Scale Comput. 2012;10:81–109. doi: 10.1002/9781118130506.ch5
[8] Cordasco G, Scarano V, Spagnuolo C. Distributed mason: a scalable distributed multi-agent simulation environment. Simul Model Pract Theory. 2018;89:15–34. doi: 10.1016/j.simpat.2018.09.002
[9] Rashid ZN, Zebari SR, Sharif KH, et al. Distributed cloud computing and distributed parallel computing: a review. In: 2018 International Conference on Advanced Science and Engineering (ICOASE). IEEE; 2018. p. 167–172.
[10] Taylor SJ. Distributed simulation: state-of-the-art and potential for operational research. Eur J Oper Res. 2019;273(1):1–19. doi: 10.1016/j.ejor.2018.04.032
[11] Kitova OV, Kolmakov IB, Dyakonova LP, et al. Hybrid intelligent system of forecasting of the socio-economic development of the country. Int J Appl Business Economic Res. 2016;14(9):5755–5766.
[12] Brailsford SC, Eldabi T, Kunc M, et al. Hybrid simulation modelling in operational research: a state-of-the-art review. Eur J Oper Res. 2019;278(3):721–737. doi: 10.1016/j.ejor.2018.10.025
[13] Turner II B, Esler KJ, Bridgewater P, et al. Socio-environmental systems (ses) research: what have we learned and how can we use this information in future research programs. Curr Opin Environ Sustain. 2016;19:160–168. doi: 10.1016/j.cosust.2016.04.001
[14] Eldabi T, Brailsford S, Djanatliev A, et al. Hybrid simulation challenges and opportunities: a life-cycle approach. In: 2018 Winter Simulation Conference (WSC); Gothenburg, Sweden. IEEE; 2018. p. 1500–1514.
[15] Mustafee N, Brailsford S, Djanatliev A, et al. Purpose and benefits of hybrid simulation: contributing to the convergence of its definition. In: 2017 Winter Simulation Conference (WSC); Las Vegas, USA. IEEE; 2017. p. 1631–1645.
[16] Fowler M, Lewis J. Microservices: a definition of this new architectural term; 2014. Available at https://martinfowler.com/articles/microservices.html.
[17] Collier R, Russell S, Golpayegani F. Harnessing hypermedia MAS and microservices to deliver web scale agent-based simulations. In: Proceedings of the 17th International Conference on Web Information Systems and Technologies – WEBIST; INSTICC. SciTePress; 2021. p. 404–411. [online only]
[18] Zimmermann O. Microservices tenets. Computer Sci-Res Develop. 2017;32(3-4):301–310. doi: 10.1007/s00450-016-0337-0
[19] Fielding RT. Architectural styles and the design of network-based software architectures [dissertation]. University of California, Irvine; 2000.
[20] Collier R, Russell S, Ghanadbashi S, et al. Towards the use of hypermedia mas and microservices for web scale agent-based simulation. SN Computer Sci. 2022;3(6):510. doi: 10.1007/s42979-022-01424-2
[21] Pursula M. Simulation of traffic systems-an overview. J Geographic Inform Decision Anal. 1999;3(1):1–8.
[22] Espié S, Auberlet JM. ARCHISIM: a behavioral multi-actors traffic simulation model for the study of a traffic system including ITS aspects. Int J ITS Res. 2007;5(1):7–16.
[23] Horni A, Nagel K, Axhausen K. Multi-agent transport simulation MATSim. London: Ubiquity Press; 2016.
[24] Jagutis M, Russell S, Collier R. Simulating traffic with agents, microservices & rest. In: Proceedings of the 15th International Symposium on Intelligent Distributed Computing; Bremen, Germany. Springer; 2022.
[25] Collier RW, O'Neill E, Lillis D, et al. MAMS: multi-agent microServices. In: Companion Proceedings of The 2019 World Wide Web Conference, WWW '19; San Francisco, USA. New York, NY: Association for Computing Machinery; 2019. p. 655–662.
[26] Collier RW, Russell S, Lillis D. Reflecting on agent programming with agentspeak (l). In: International Conference on Principles and Practice of Multi-Agent Systems; Bertinoro, Italy. Springer; 2015.
[27] Dhaon A, Collier RW. Multiple inheritance in agentSpeak (L)-style programming languages. In: Proceedings of the 4th International Workshop on Programming based on Actors Agents & Decentralized Control; Portland, USA. ACM; 2014. p. 109–120.
[28] Ricci A, Viroli M, Omicini A. Cartago: a framework for prototyping artifact-based environments in mas. In: Weyns D, Parunak HVD, Michel F, editors. Environments for multi-agent systems III. Berlin, Heidelberg: Springer Berlin Heidelberg; 2007. p. 67–86.
[29] O'Neill E, Lillis D, O'Hare GMP, 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; Auckland, NZ; 2020.
[30] Guinard DD, Trifa VM. Building the web of things. Vol. 3. Manning Publications Shelter Island; 2016. ISBN 9781617292682.
[31] Charpenay V, Käbisch S. On modeling the physical world as a collection of things: the w3c thing description ontology. In: European Semantic Web Conference. Springer; 2020. [online only]
[32] Ciortea A, Boissier O, Ricci A. Engineering world-wide multi-agent systems with hypermedia. In: International Workshop on Engineering Multi-Agent Systems; Stockholm, Sweden. Springer; 2018. p. 285–301.

[33] Adam C, Gaudou B. BDI agents in social simulations: a survey. Knowl Eng Rev. 2016;31(3):207–238. doi: 10.1017/S0269888916000096
[34] United Nations Department of Economic and Social Affairs. World Urbanization Prospects: The 2018 Revision. UN; 2019.
[35] Ullah M, Khattak K, Khan Z, et al. Vehicular traffic simulation software: a systematic comparative analysis. Pakistan J Eng Technol. 2021;4(1):66–78. doi: 10.51846/vol4iss1pp66-78.
[36] Noroozian A, Hindriks K, Jonker C. Towards simulating heterogeneous drivers with cognitive agents. In: ICAART 2014 – Proceedings of the 6th International Conference on Agents and Artificial Intelligence; Vol. 2, 6th International Conference on Agents and Artificial Intelligence, ICAART 2014; Conference date: 06-03-2014 Through 08-03-2014; Angers, France. SciTePress; 2014. p. 147–155.
[37] Krajzewicz D, Hertkorn G, Rössel C, et al. SUMO (Simulation of Urban MObility) – an open-source traffic simulation. In: Al-Akaidi A, editor. Proceedings of the 4th Middle East Symposium on Simulation and Modelling (MESM20002); Dubai; 2002. p. 183–187.
[38] Prevedouros PD, Li H. Comparison of freeway simulation with INTEGRATION, KRONOS, and KWaves. In: Fourth International Symposium on Highway Capacity. Maui, Hawaii; 2000. p. 96–107. ISSN 0097-8515.
[39] Axhausen KW, Horni A, Nagel K. The multi-agent transport simulation MATSim. London (UK): Ubiquity Press; 2016.
[40] Fang X, Tettamanti T. Change in microscopic traffic simulation practice with respect to the emerging automated driving technology. Periodica Polytechnica Civil Eng. 2022;66(1):86–95. doi: 10.3311/PPci.17411.
[41] Auld J, Hope M, Ley H, et al. POLARIS: agent-based modeling framework development and implementation for integrated travel demand and network and operations simulations. Transp Res Part C: Emerging Technologies. 2016;64:101–116. doi: 10.1016/j.trc.2015.07.017
[42] Nagel K, Schreckenberg M. Traffic jam dynamics in stochastic cellular automata. Stuttgart (Germany): Los Alamos National Laboratory; 1995.
[43] Shang XC, Li XG, Xie DF, et al. A data-driven two-lane traffic flow model based on cellular automata. Phys A: Statistical Mechanics and Its Applications. 2022;588:126531. doi: 10.1016/j.physa.2021.126531
[44] Rao AS, Georgeff MP. Bdi agents: from theory to practice. In: ICMAS; San Francisco, USA; Vol. 95; 1995.
[45] Wai SY, Cheah WS, Wai SK, et al. Towards software engineering perspective for BDI agent. In: 2021 4th International Symposium on Agents, Multi-Agent Systems and Robotics (ISAMSR); Malaysia; 2021. p. 106–110.
[46] Bulumulla C, Singh D, Padgham L, et al. Multi-level simulation of the physical, cognitive and social. Comput Environ Urban Syst. 2022;93:101756. doi: 10.1016/j.compenvurbsys.2021.101756
[47] Drogoul A, Vanbergue D, Meurisse T. Multi-agent based simulation: where are the agents? In: Simão Sichman J, Bousquet F, Davidsson P, editors. Multi-agent-based simulation II. Berlin, Heidelberg: Springer Berlin Heidelberg; 2003. p. 1–15.
[48] Edmonds B, Moss S. From kiss to kids – an 'anti-simplistic' modelling approach. In: Davidsson P, Logan B, Takadama K, editors. Multi-agent and multi-agent-based simulation. Berlin, Heidelberg: Springer Berlin Heidelberg; 2005. p. 130–144.
[49] Vachtsevanou D, Junker P, Ciortea A, et al. Long-lived agents on the web: Continuous acquisition of behaviors in hypermedia environments. In: Companion Proceedings of the Web Conference; Taipei, Taiwan. 2020. p. 185–189.
[50] Ciortea A, Mayer S, Gandon F, et al. A decade in hindsight: the missing bridge between multi-agent systems and the world wide web. In: Proceedings of the International Conference on Autonomous Agents and Multiagent Systems; Montreal, Canada; 2019.
[51] Ciortea A, Mayer S, Boissier O, et al. Exploiting interaction affordances: on engineering autonomous systems for the web of things. In: Second W3C Workshop on the Web of Things The Open Web to Challenge IoT Fragmentation. Munich, Germany; 2019 Jun. [Online]
[52] Kravari K, Bassiliades N. A rule-based eCommerce methodology for the IoT using trustworthy intelligent agents and microservices. In: International Joint Conference on Rules and Reasoning; Luxemborg. 2018.
[53] Krivic P, Skocir P, Kusek M, et al. Microservices as agents in IoT systems. In: Smart Innovation, Systems and Technologies; 2018. ISSN: 2190-3018.
[54] Zouad S, Boufaida M. Using multi-agent microservices for a better dynamic composition of semantic web services. In: Proceedings of the 4th International Conference on Advances in Artificial Intelligence, ICAAI '20. New York, NY: Association for Computing Machinery; 2021. p. 47–52.
[55] Alves P, Gomes D, Rodrigues C, et al. Grouplanner: a group recommender system for tourism with multi-agent microservices. In: Dignum F, Mathieu P, Corchado JM, De La Prieta F, editors. Advances in practical applications of agents, multi-agent systems, and complex systems simulation. The PAAMS Collection. Cham: Springer International Publishing; 2022. p. 454–460.
[56] Gibson JJ. The ecological approach to visual perception. New York (USA): Houghton-Mifflin; 1979.
[57] Weyns D, Omicini A, Odell J. Environment as a first class abstraction in multiagent systems. Auton Agent Multi Agent Syst. 2007;14(1):5–30. doi: 10.1007/s10458-006-0012-0
[58] Behrens T, Hindriks KV, Bordini RH, et al. An interface for agent-environment interaction. In: Collier R, Dix J, Novák P, editors. Programming multi-agent systems. Berlin, Heidelberg: Springer Berlin Heidelberg; 2012. p. 139–158.

[59] Ricci A, Croatti A, Bordini R, et al. Exploiting simulation for MAS programming and engineering-the JaCaMo-sim platform. In: 8th International Workshop on Engineering Multi-Agent Systems (EMAS 2020); Auckland, New Zealand; 2020 May. Cham: Springer. p. 42–60. (Lecture Notes in Computer Science; vol. 12589).
[60] Joo J, Kim N, Wysk RA, et al. Agent-based simulation of affordance-based human behaviors in emergency evacuation. Simul Model Pract Theory. 2013;32:99–115. doi: 10.1016/j.simpat.2012.12.007
[61] Busogi M, Shin D, Ryu H, et al. Weighted affordance-based agent modeling and simulation in emergency evacuation. Saf Sci. 2017;96:209–227. doi: 10.1016/j.ssci.2017.04.005
[62] Hassanpour S, Rassafi AA. Agent-based simulation for pedestrian evacuation behaviour using the affordance concept. KSCE J Civil Eng. 2021;25(4):1433–1445. doi: 10.1007/s12205-021-0206-7
[63] Kapadia M, Singh S, Hewlett W, et al. Egocentric affordance fields in pedestrian steering. In: Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games, I3D '09. New York, NY: Association for Computing Machinery; 2009. p. 215–223.
[64] Ksontini F, Mandiau R, Guessoum Z, et al. Affordance-based agent model for road traffic simulation. Auton Agent Multi Agent Syst. 2015;29(5):821–849. doi: 10.1007/s10458-014-9269-x
[65] Klügl F, Timpf S. Approaching interactions in agent-based modelling with an affordance perspective. In: Sukthankar G, Rodriguez-Aguilar JA, editors. Autonomous agents and multiagent systems. Cham: Springer International Publishing; 2017. p. 222–238.
[66] Klügl F, Timpf S. Towards more explicit interaction modelling in agent-based simulation using affordance schemata. In: Edelkamp S, Möller R, Rueckert E, editors. KI 2021: Advances in Artificial Intelligence. Cham: Springer International Publishing; 2021. p. 324–337.
[67] Maerivoet S, De Moor B. Cellular automata models of road traffic. Phys Rep. 2005;419(1):1–64. doi: 10.1016/j.physrep.2005.08.005
[68] Beaumont K, O'Neill E, Bermeo N, et al. Collaborative route finding in semantic mazes. In: Proceedings of the All the Agents Challenge (ATAC 2021); 20th International Semantic Web Conference (online); 2021.
[69] Hogan A, Blomqvist E, Cochez M, et al. Knowledge graphs. Springer Nature Switerland; 2021. doi: 10.1007/978-3-031-01918-0. (Synthesis Lectures on Data, Semantics, and Knowledge; 22).