The Current State of Agent-Based Modeling

Research Internship Report

Tim Leys
tim.leys@student.uantwerpen.be

Department of Mathematics and Computer Science
University of Antwerp
Belgium
August 21, 2019

Abstract

The field of agent-based modeling (ABM) has become a very popular field of study. This relatively new paradigm allows modelers to model individual behaviour of each entity in a system to create virtual "micro-worlds". The idea for such an organizational structure strongly relates to multi-agent systems. Multi-agent systems aim to solve complex problems by dividing it into smaller problems and assigning them to agents.

The purpose of this study is twofold: (1) We want to analyze the relation between MAS and ABM by reviewing the current literature of these topics. (2) We want to identify whether ABM tools implement the key features of agents and ABM, as well as determine whether repeatability can be achieved across platforms. We will do this by implementing an example model in a selection of tools and analyze the tools' features and output traces. The selected tools are DEVS, NetLogo, Repast, and SARL.

Keywords: Agent-Based Modeling, Multi-Agent Systems, ABM Tools, Agents.

1 Introduction

The field of agent-based modeling (ABM) has become a very popular field of study [54]. This relatively new paradigm allows modelers to model individual behaviour of each entity in a system, rather than abstracting away individual entities and describing global system behaviour. Agent based models can be considered as "micro-worlds" that are a small versions of their real world counterparts [59]. Because of this, agent-based modeling and simulation are very suitable for modeling systems where living entities interact with each other. Therefore, agent-based modeling and simulation has become a very popular technique in the fields of sociology, anthropology, biology, and market analysis [55].

An agent-based model comprises multiple autonomous agents that interact with each other and their environment. The idea for such an organizational structure comes from the field of multi-agent systems (MAS) [59]. Multi-agent systems aim to solve complex problems by dividing it into smaller problems. Each agent is then tasked to solve one of the sub-problems with a limited capabilities and limited information. The agents then need to collaborate to achieve their goal [37]. The MAS paradigm was originally proposed for distributed artificial intelligence, but quickly became a popular as an organizational structure for distributed systems [46].

Though there is a clear link between ABM and MAS, the specifics of their relation remain unclear. Some call ABM an application of MAS [32], while others state that their is no clear distinction between the two concepts[54]. We, however, believe that their relation is not so easily defined. Even though both approaches use similar concepts, they are developed for different purposes.

We want to analyze the relation between MAS and ABM by reviewing the current literature of these topics. We want to broaden our knowledge of the paradigms by analyzing different definitions definitions and by identifying the different concepts that they use. Then we will look into studies that mention both paradigms and identify their relation.

With the advent of ABM, many tools have been developed to support this modeling paradigm. However, a universally accepted formal semantics remains absent. This situation results in a proliferation of tools that were developed according to different interpretations of the paradigm [9]. Moreover, the lack of formal semantics imply that the semantics of a model are determined by its implementation in a specific tool, making them platform dependent. Platform dependency, in turn, affects the reproducibility of simulation across platforms. This is a very undesirable property for modeling and simulation platforms.

The second purpose of this study is to identify whether ABM tools implement the key features of agents and ABM, as well as determine whether repeatability can be achieved across platforms.

To determine this, we created a feature diagram that shows different features that can be expected from an agent-based modeling tool. Next, an example model will be implemented in each of the tools under study and instantiated

the diagram for each of the tools. To analyze their repeatability, a set of scenarios will be determined and run on each of the implementations. Subsequently, we can determine to which precision the scenarios are repeatable.

To evaluate our results, we established a set of research questions we want to answer in this document:

1. Is there a universally agreed upon meaning for the concepts: agent, multi-agent system, and agent-based modeling?
2. Do agent-based modeling and multi-agent systems agree on the meaning of some concepts.
3. What is the relationship between multi-agent systems and agent-based modeling?
4. How well do contemporary agent-based tools implement the paradigm?

This document is structured as follows. In Section 2, we will discuss the results from our literature review. In Section 3, we will discuss our methodology to review the tools and the results of the experiments. In Section **??**, we will discuss our results from both studies and address some of the problems we identified. In Section 4, we will discuss related work to our study. We will conclude this document in section 5.

2 Literature Review of ABM and MAS Concepts

The first purpose of this document is to analyze the concepts of agents, multi-agent systems, agent-based modeling. Currently, there are no universal agreement on what each of the concepts means [55]. Also, many definitions are proposed that are focused on specific research areas [89]. To give an overview, we will provide a review of some definitions of each concept as well as describe their history and associated background knowledge.

2.1 Methodology

To perform an objective literature study and increase the confidence of our research, we applied a systematic approach based on the technique of Bárbara Kitchenham [50].

The first step of the systematic approach is to identify keywords that can be used for the search queries. The set of keywords was determined by inspecting the research questions. Table 1 shows a list of the keywords that were retrieved with their synonyms. To obtain the synonyms and antonyms, we used thesaurus and looked at different denotations for concepts in the literature.

- (((Multi-agent system) OR MAS OR (self-organized system))
  OR ((agent-based modeling) OR AMB OR (individual-based model)))
  AND (definition OR explanation OR description)

- ((Multi-agent system) OR MAS OR (self-organized system))
  AND ((agent-based modeling) OR AMB OR (individual-based model))
  AND (similarity OR similar OR likeness OR Correspondence OR comparison OR sameness OR relationship OR correlation)
  OR (dissimilarity OR dissimilar OR contrast OR unlikeness OR distinction OR dissimilitude OR variation)

- (((Multi-agent system) OR MAS OR (self-organized system))
  OR ((agent-based modeling) OR AMB OR (individual-based model)))
  AND (semantics OR meaning)

In addition, we gathered a set of important researchers in the domain of Multi-Agent Systems and Agent-Based Modeling, as well as important journals and conferences. Figure 1 shows a diagram with all important authors. We also asked colleagues who are more familiar in the field to recommend interesting work on the topics.

We selected literature based on how well the title of the work relates to the topic. This search resulted in a set of 86 papers. An overview of each paper can be downloaded via this link.

In the next phase of the process, we read the abstracts of each of the papers and ranked them into three categories: *must reads*, *interesting work*, *least related*

Table 1: Table of the used keywords and their synonyms

Keyword	**Synonyms**
Multi Agent System	MAS self-organized system
Agent Based Modeling	ABM individual-based model
relation(ship)	likeness correspondence comparison sameness similarity correlation
dissimilarity	contrast difference unlikeness distinction dissimilitude variation
definition	explanation description
semantics	meaning

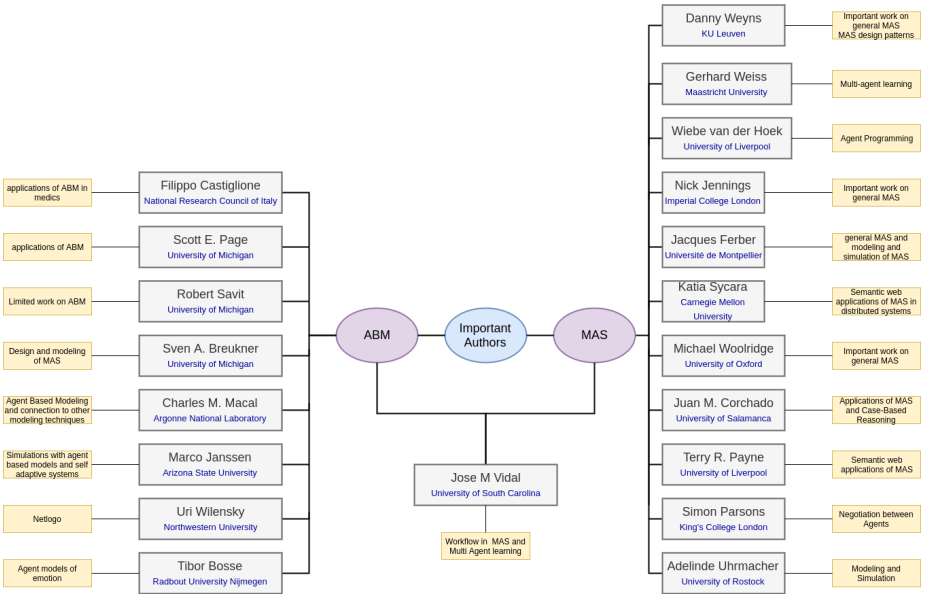

Figure 1: Mind map of important authors

work. Due to the scope of the project, we selected the 30 most important papers as starting point for the research.

During our research, we came across additional questions, which lead to the inclusion of more papers.

2.2 Multi-Agent Systems

The first paradigm we discuss that introduced the concept of agents is multi-agent systems [73]. Currently there are two big research fields interested in this approach. The first one is artificial intelligence [73], which use communicating agents to achieve distributed artificial intelligence. The other field is engineering (especially robotics and control theory) [32, 73], who use software and hardware agents to eliminate the need for a central control unit.

In this section, we will elaborate on the concept by briefly discussing its history and reviewing some definitions found in literature.

2.2.1 History of Multi-Agent Systems

The first steps into multi-agent research began in 1980, when a group of AI researchers organized the first workshop for distributed artificial intelligence at MIT. This new field of study concerned itself with issues of how intelligent problem solvers could coordinate effectively to solve problems [46]. Initially the field was divided into two camps[19]: Distributed Problem Solving (dps) and

Multi-Agent Systems (mas). However, recently the term multi-agent systems has become a more general term, encompassing all distributed problem solvers.

The first model for distributed problem solvers were actors [12, 10]. Actors are self-contained components in a distributed system that can make autonomous decisions. The primitives of an actor are:

- Create an actor
- Send a message
- Change state

Actors are blind to their environment. They only received messages from each other and decided how to respond to these messages. Though agents and actors are two distinct concepts, they share technical and historical connection [48] and actors have been defined as "computational agents" in the past.

Actors were the first implementation format for multi-agent systems. Now many architectures and methodologies have been defined that are specifically developed for multi-agent systems. The PASSI methodology [30] provides a requirements-to-code methodology for developing agent systems from both the OOP perspective as well as the AI perspective. The ADELFE methodology [16, 68] was specifically developed for designing adaptive multi-agent systems [25]. In these systems agents learn there topology through cooperative and non-cooperative states. Next to design-methodologies, specific architectures have been developed for implementing distributed agents systems. Some examples are the open agent architecture [27], and Cougaar [44]. Also, dedicated tools were developed that allowed users to develop multi-agent systems more easily by providing domain specific abstractions, such as JADE [15].

Some early applications of MAS include *air traffic control* [24], the distributed vehicle monitoring task [34], and blackboards [29]. Currently, MAS has a wide variety of applications ranging from cloud computing to robotics and even city and built environments [32].

2.3 Definition review

In this section we will review some of the definitions of multi-agent systems we found in the literature.

The first definition comes from a survey by P. Stone. [77].

Definition 1 *A multi-agent system is a loosely coupled network of problem-solving entities (agents) that work together to find answers to problems that are beyond the individual capabilities or knowledge of each entity (agent).*

Stone focuses on the distributed and collaborative aspect of multi-agent systems. The global goal of the system is divided into smaller parts and given to agents. To achieve the global goal agents need to collaborate.

The second definition we reviewed, we derived from *An Introduction to Multi-Agent Systems* [37].

Definition 2 *A Multi-Agent System (MAS) is an extension of the agent technology where a group of loosely connected autonomous agents act in an environment to achieve a common goal. This is done either by cooperating or competing, sharing or not sharing knowledge with each other.*

Ferber et al. note that agents in a multi-agent system are not necessarily collaborative. In some cases to achieve a certain goal it is necessary for agents to compete against each other.

In the book *Multi-agent systems: a modern approach to distributed artificial intelligence* [88] stated the major characteristics of a multi-agent system.

Definition 3 *The major characteristics of a multi-agent system are:*

- *Each agent has just incomplete information and is restricted in its capabilities.*
- *System control is distributed.*
- *Data is decentralized.*
- *Computation is asynchronous.*

This definition again highlights the distributed nature of multi-agent systems. It also discusses the more technical details of multi-agent systems.

In Table 2, we created a table with important keywords and their occurrence in the reviewed definitions. We clearly see that all definitions are very similar. We can conclude that essential features of a multi-agent system are:

- A MAS is a distributed system with agents as central entities
- Agents are limited in their knowledge and capabilities
- Agents need to collaborate to achieve their goals

2.3.1 Single-Agent Systems

In addition to distributed multi-agent systems, there are centralized single-agent systems [77]. These systems, which were popularized by the book *Logical foundations of Artificial Intelligence* [39], consist of a single agent in an environment. These systems obviously omit the need for communication. In this contexts, the term agent refers to a distinguishable entity that is able to sense and act on its environment deliberately, Figure 2.

According to [77], a single agent system can consist out of multiple entities. However, these entities act as actuators for a central agent (or central process) and the agent will identify each of these separate components as part of itself.

In [77], another type of single-agent systems is described. In these systems, multiple agents are present. However, an agent has no representation

Table 2: Table of occurences of concepts in the reviewed definitions

	P. Stone [77]	Ferber et al. [37]	G. Weiss [88]
Agents	✓	✓	✓
Limited knowledge	✓		✓
Limited capabilities	✓		✓
Distributed system	✓	✓	✓
Common goal		✓	
Collaboration	✓	✓	
Competition		✓	
Asynchronous computation			✓
Decentralized data			✓

of the other agents present. The other agents are considered to be part of the environment and can not be contacted through specialized agent-to-agent communication protocols.

2.4 Agent-Based Modeling

Agent-based modeling is a relative new paradigm for modeling and simulation [55]. The paradigm advocates to model each entity in a system individually as an autonomous agent. ABM exploits a bottom-up approach when modeling a system, instead of a top-down approach used in for example system dynamics [31] and causal block diagrams [42]. In this section, we will discuss the history of this novel approach, as well as discuss definitions found in the literature. We will also discuss the relation between ABM and MAS.

2.4.1 History of Agent-Based Modeling

In [59], Fabien Michel clearly describes the history of behind multi-agent systems. In this section we will give a summary and add some extra information we found in literature.

"Modeling of complex systems has always been a motivation for researchers" [59]. A historical example of such a model is the predator prey model, originally proposed by Volterra [86]. This continuous deterministic model captured the population dynamics of multiple species living in the same habitat with differential equations. This model produced intuitively sound results and therefor became very popular. In a predator prey situation, it displayed a oscillation in both populations. When the prey population is high, the predator population would grow, resulting in a decrease in the prey population. Contrariwise, when the the prey population is low, the predator population would decrease, resulting in a growth of the prey population.

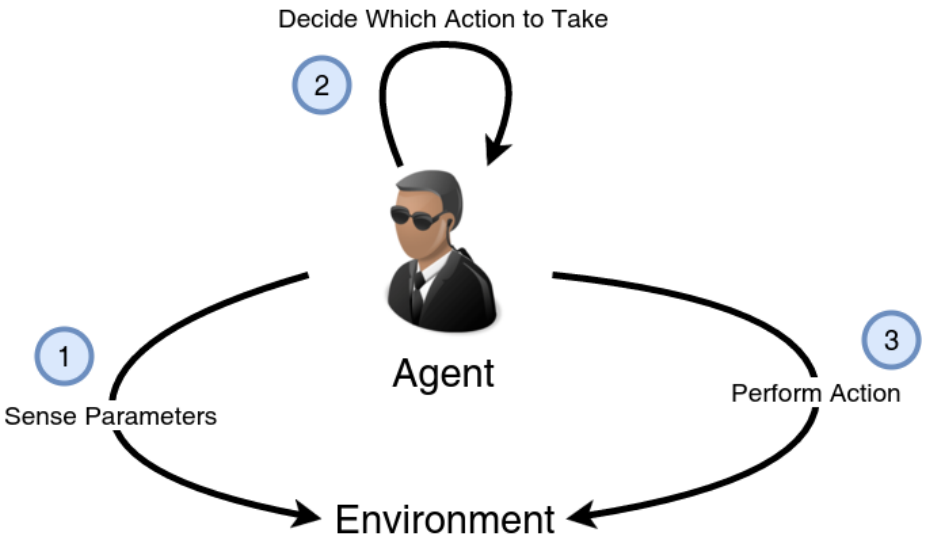

Figure 2: Typical work flow of an agent

Results from experiments on real test predator/prey habitats, however, showed non-oscillating behaviour. This caused the need for more complex models.

"With the advent of computer science, the possibility of simulating stochastic models efficiently presented itself" [59, 14]. Stochastic models allowed for incorporating the inherent non-deterministic behaviour of real life interaction situations. However, these models exhibit much more complex behaviour and a high variability even used with the same parameters.

All of these approaches still face a set of problems [37]:

- Only a global perspective is possible
- Equation parameters hardly take into account the complexity of micro-level interactions
- The modeling of individual agents is impossible
- Integrating qualitative aspects is hard

The first approach to deal with these problems is called micro-simulation and is originally suggested by Orcutt [65]. The basic principle of micro-simulation is to model the micro-level behaviour of a system under study with rules that apply to attributes of micro-level entities, resulting in a change of state or behaviour of those entities. The incorporation of the micro-level into the development of models, allowed the generation of better and more informative results for social system dynamics. *"Though this modeling paradigm originated in the field of social sciences, it can be seen as a forerunner of individual based modeling."* [59]

Agent-based modeling takes the integration of the micro-level a step further. Apart from modeling entities and their behaviour, ABM suggests to model their interactions with each other and actions on the environment explicitly. The popularity of ABM can be credited to a number of factors. First, ABM is very flexible. Models can easily be updated by adding new types of agents or changing the populations of agents. ABMs also allow fine-grained control over the simulation. In contrast to other modeling techniques, ABM does not abstract individual behaviour and interactions. Being societies by themselves, and being built on the same basis as any complex systems, MAS prove to be "artificial micro-worlds", of which it is possible to control all characteristics and reproduce series of experiments as in a laboratory [59].

2.4.2 Definition Review

In the current literature, we found several definition for agent-based models. We will now list the most prominent and discuss them.

Due to the wide variety of applications of ABM, it is not trivial to give a uniform definition. Charles M. Macal therefor proposed a collection of four definitions for different types of agent based models [54]. The categories identified by Macal are *Individual Agent Based Models, Autonomous Agent Based Models, Interactive Agent Based Models,* and *Adaptive Agent Based Models.*

The first category are defined as follows:

Definition 4 *An individual ABMS is one in which the agents in the model are represented individually and have diverse characteristics.*

In these models, agents are hardly autonomous. Each agent does not perceive any information of its environment and thus can not take any decisions based on those percepts. Their behaviour is described in a script and no individual state is necessary. Note that the agents behaviour in these models is even more simplistic than the simple reflex agent. Though the agents in these models do not meet the requirements of most definitions in Section 2.5.2, these models do fit in the paradigm of agent-based modeling. These models aim to reflect the individual behaviour of each agent and the model is constructed in a "bottom-up" approach that is characteristic for agent based modeling.

The second category of agent-based models is defined as:

Definition 5 *An autonomous ABMS is one in which the individual agents have internal behaviors that allow them to be autonomous, able to sense whatever condition occurs within the model at any time, and to act on the appropriate behavior in response.*

Autonomy is widely considered a fundamental property of agents. This definition also points out that agents are situated in an environment. This property is strongly connected with autonomy, since an agent needs to perceive some external data to decide which action to take. The environment provides this data.

Interactive agent-based models are defined as:

Definition 6 *An interactive ABMS is one in which autonomous agents interact with other agents and with the environment.*

This category introduces an important feature of multi-agent systems, namely interaction between agents. Note that the interaction in these models is either direct, indirect, or both. The agents in the previous category are only able to perceive their environment, but are unable to act upon it. This prohibits indirect communication.

The last category is defined as:

Definition 7 *An adaptive ABMS is one in which the interacting, autonomous agents change their behaviors during the simulation, as agents learn, encounter novel situations, or as populations adjust their composition to include larger proportions of agents who have successfully adapted.*

Here agents are able to change their behaviour during the simulation.

A completely different categorization is given by E. Bonabeau [18]. Bonabeau does not categorize agent based models on the capabilities of the agents, but rather the application of the models. He presents four categories: Flow Models, Market Models, Organization Models, and Diffusion Models.

Flow models are characterized by featuring a large amount of mobile agents in a particular situation. The interest of the modelers is how the agents move through their environment and create certain movement flows. Flow models can be further specialized into evacuation models and flow management models. Evacuation models aim to analyze situations were agents are trying to exit a room with limited exits, while the agents exhibit irrational herding behavior and obsessive personal interests. Flow management models model environments with mobile decisive agents.

Market models aim to model the dynamics of stock markets and auctions. Agent-based models are preferred approach over spreadsheet models or system dynamics, because they do not achieve the same deep insights. The behaviour of the market emerges out of the interactions of the players, who change their behaviour when the market changes.

Organization models are used to analyze emergent collective behaviour of an organization. An example of this is operational risk. Operational risk factors are often largely internal to the organization and clear mathematical or statistical link and the size and frequency of operational loss does not exist.

Diffusion models are applied to cases where people are influenced by their social contexts. Agents-based models improve on classic models in that they can model locality. In these models, agents only communicate with their neighbours.

There is no universal agreement on the definition for ABM [55]. This is apparent from the several definitions for different types of agent-based model by C. Macal. He provides four definitions for agent-based models based on the internal complexity of the agents of the model. There is big lack of unification

in the field, however, we take the view that it is possible to provide a clear all encompassing definition for ABM. Therefor we did not focus on the complexity of a model, but rather on what makes ABM stand out as a modeling paradigm.

Definition 8 *An agent based model is a model in which:*

- *Each individual of a system and its behaviour is modeled (bottom-up), rather than the global behaviour of the system (top-down).*
- *Each of the entities is modeled as an agent, where the interactions between agents and actions on the environment are modeled explicitly.*
- *The environment in which the agents are situated is modeled explicitly.*

2.4.3 Comparison with Other Modeling Paradigms

Agent-based modeling has become a very popular tool for modeling and simulation [54, 59]. In this section, we will discuss what sets this paradigm apart from other modeling paradigms.

In agent-based models, individual entities are not abstracted away. Rather, they are the key elements in the models. This is in stark contrast with equation-based modeling (EBM), which abstracts away individual behaviour and aims to capture global system behaviour with a set of differential equations. Examples of EBM approaches are causal block diagrams [42], and system dynamics [31].

In essence, equation-based models provides an approach to *"numerically characterize the evolution of a system from its parameters"* [59], while ABM allows the modeler to create a habitat of agents which can interact with each other and their environment, and explicitly model their individual behaviour. This creates some kind of 'virtual micro-world', which can be analyzed as a small version of the real system.

Though ABM is usually compared to EBM, it also serves as an alternative to stochastic modeling [69]. In stochastic modeling, a system is described with probability distributions and information about the system is inferred through probabilistic reasoning. This modeling paradigm aims to model variability and randomness that is often found in real-world systems.

In agent-based modeling, stochasticity is modeled in the behaviour. Instead of inferring information through probabilistic reasoning, the system is simulated and information can be retrieved by analyzing the agents interactions.

2.4.4 Simulation Time

In modeling and simulation, time is virtualized (simulation time) to allow for simulating large time intervals in a very short time [41]. This sets ABM apart from MAS, in which the agent systems are deployed in the real world and act

in real-time, as can be seen in the actors paradigm [12] or in the tool SARL [71].

2.5 Agents

The agent is the central concept in both multi-agent systems as well as agent-based modeling [32, 54]. We will therefore first elaborate on this concept by defining what the essential features of agent-hood. We will do this by studying definitions found in the literature. We will then elaborate on concepts that are related to agents.

2.5.1 History of Agents

There are three fields of research that contributed to the concept of agent [46]: Artificial intelligence [73], object-oriented programming [20] and concurrent object based systems [12, 11], and human computer interface design [56].

Artificial intelligence is undoubtedly the main contributor to the field. Ultimately, AI is all about building intelligent artifacts, and if these artifacts sense and act in some environment, then they can be considered agents. This makes agents the central study in artificial intelligence [73]. However, up until the 1980's, little effort was put in the research of intelligent agents. In this time period, research was focused more on the individual components of an agents, such as question-answering systems, theorem-provers, vision systems, etc. In 1987, Genesereth and Nilsson published an influential paper [39] which caused the concept of the whole-agent to be widely accepted in the field of artificial intelligence.

Agents that exhibit simple reactive behaviour were the primary model adapted by psychological behaviourists such as Skinner [76]. However, most AI researchers deem these agents to be too simple to provide much leverage. Rosenshein [72] and Brooks [22] question this assumption. Currently a lot of research is focused on finding efficient algorithms for keeping track of complex systems. An impressive example is the Remote Agent Program that controlled the Deep Space One spacecraft [61].

A more complex behavioural model was developed that introduced goals. Goal-based agents did not simply react to perceptions in their environment, rather they decided on an action to perform that optimized conditions to reach a certain desired end-state or goal. With goal-based agents, the application of agents in robotics was explored. The first robotic implementation of a logical, goal-based agent was Shakey the Robot [62]. The agent approach also gained a lot of attention in the field of software engineering [90]. Shoham [75] developed a new programming paradigm based on agents, agent-oriented programming. The goal-based view of agents also dominates the field of distributed artificial intelligence, where a problem is divided over multiple solvers, the agents, in a multi-agent system [46].

In 1987, the goal-based model was further specialized into belief-desire-intention agents [21]. Goals were further specified into desires (general goals) and intentions (currently pursued goals). BDI agents also have beliefs of their environment. This allows agents to remember information from their environment that they can't perceive.

Research has also been devoted to adding learning capabilities to agents [23, 60]. Learning agents can be divided into two groups. Agents that can adapt their behaviour by studying the result of their actions and agents that can adapt there social interactions to become more competent.

In recent years, the interest in agents and agent design has increased rapidly. This can be partly credited to the growth of the internet, which provides a massive collection of data that can be used as environment for agents [35, 51]. Also the rise of cyber-physical systems is a growing field that utilizes the concept of agents for automation of development in industry 4.0 settings [52, 87, 53].

2.5.2 Definition Review

In this section, we will review a set of definitions for the concept of agent. Since an agent is a concept in MAS as well as ABM, we reviewed definitions in both fields to see whether they have different conceptions about the term. We focused on surveys to filter out definitions that are biased to a specific field of research.

The first definition is derived from a survey of Nick Jennings [6] and Michael Woolridge [4] from 1996 about software agents in multi-agent systems [47].

Definition 9 *Agents should have following key hallmarks:*

- *Autonomy: Agents should be able to perform the majority of their problem solving tasks without the direct intervention of humans or other agents, and they should have a degree of control over their own actions and their own internal state.*
- *Social ability: Agents should be able to interact, when they deem appropriate, with other software agents and humans in order to complete their own problem solving and to help others with their activities where appropriate.*
- *Responsiveness: Agents should perceive their environment (which may be the physical world, a user, a collection of agents, the INTERNET, etc.) and respond in a timely fashion to changes which occur in it.*
- *Pro-activeness: Agents should not simply act in response to their environment, they should be able to exhibit opportunistic, goal-directed behaviour and take the initiative where appropriate.*

Jennings et al. identify agents to be autonomous communicative entities. They hold a degree of autonomy and should be able to solve a majority of

problems by themselves. They should be able to communicate with each other on their own initiative.

Jennings et al. also introduce some terms, such as environment and goals. An agent's environment provides the conditions under which an entity (agent or object) exists [64]. Agents perceive information from their environment and can act upon it, which can in turn cause a change in the environment. Agents also have a local goal. This is a set of conditions that the agent tries to satisfy as best as possible. We will discuss these terms in more detail in Section 2.5.3.

The second definition is from Charles M. Macal [1] and Michael J. North. They provide a definition for agents from the perspective of agent-based modeling in *A Tutorial on Agent-Based Modeling*[55].

Definition 10 *An agents has the following essential features:*

- *An agent is a self-contained, modular, and uniquely identifiable individual.*
- *An agent is autonomous and self-directed*
- *An agent has a state that varies over time*
- *An agent is social having dynamic interactions with other agents that influence its behaviour.*

Both definitions are from a different perspective, however, some features recur in both definitions. Both definition identify that agents should by autonomous and communicative. Macal and North's definition, does not mention that agents are situated in an environment or have goals.

A paper that was included in *Multi-Agent systems: Simulation and applications* [81] by Fabien Michel et al. also provides a definition from the modeling and simulation point of view [59].

Definition 11 *An agent is a software or hardware entity (a process) situated in a virtual or a real environment:*

1. *Which is capable of acting in an environments*
2. *Which is driven by a set of tendencies (individual objectives, goals, drives, satisfaction/survival function)*
3. *Which possesses resources of its own*
4. *Which has only a partial representation of this environment*

5. *Which can directly or indirectly communicate with other agents*
6. *Which may be able to reproduce itself*
7. *Whose autonomous behavior is the consequence of its perceptions, representations and interactions with the world and other agents*

This definition mentions that agents can have a partial view of its environment. This refers to locality of an agent. An agent can only perceive information in its surroundings. E.g. a person can only see what is happening in the room he is in, not a neighbouring room. He can however remember what the room looked like when he left. When an agent only has a partial perception of his environment, the internal representation of an agent's environment is often referred to as the agents belief [40].

In a recent survey from Ali Dorri et al. [32] about MAS, the authors proposed a general application-independent definition for an agent.

Definition 12 *An agent is an **entity** which is placed in an **environment** and **senses** different parameters that are used to make a **decision** based on the **goal** of the entity. The entity performs the necessary **action** on the environment based on this decision.*

Though not mentioned explicitly, the definition states that agents should be autonomous. By sensing the environment and choosing an action accordingly, the environment can not control the agents action directly.

To compare these definitions, we constructed Table 3, in which we listed important keywords from the definitions and put a check mark if the keyword is discussed in the definition.

We identified autonomy as the most important feature of agent-hood. Every definition that is reviewed mentions this feature explicitly or implicitly. Communication, environment, and goals are all mentioned in 3 out of the 4 reviewed definitions, making them also essential features for agent-hood. Pro-activeness is only mentioned explicitly in the definition of Jennings and Woolridge, however, pro-activeness is tightly coupled to goals. If an agent does not have the notion of goals, it can not exhibit opportunistic behaviour.

We also noticed that the definitions of Jennings and Woolridge and the definitions of Michel et al. are very extensive, while the definitions of Macal et al. and Dorri et al. are rather limited.

In the definitions of agents we did not notice any bias towards MAS or ABM. This makes agents a generic concept.

Table 3: Table of occurences of concepts in the reviewed definitions

	Jennings et al. [47]	Macal et al. [55]	Michel et al. [59]	Dorri et al. [32]
Autonomy	✓	✓	✓	✓
Communication	✓	✓	✓	
Environment	✓		✓	✓
Pro-Activeness	✓			
Beliefs			✓	
Goals	✓		✓	✓
Statefull		✓		
Self Reproducibility			✓	
Resources			✓	

2.5.3 Related Terms

In this section, we provide a list of terms and concepts that are often used to describe certain aspects of agents.

Environment As seen in Section 2.5.2, many definitions state that agents are autonomous entities. However, agents are not completely free of external dependencies. *"Agents are situated in an environment that provides the conditions under which an entity (agent or object) exists "*[64]. An agent senses its environment and makes decisions based on the sensed data [85]. An environment has multiple properties that have an effect on the complexity of the system [32]:

- Accessibility: Accessibility refers to the accuracy with which agents can sense an environment. The more accessible an environment is, the more accurate and up-to-date sensed data is.
- Determinism: This feature refers to the predictability of the outcome of actions. In a deterministic environment, the effect of actions is predictable. In a non-deterministic environment, multiple factors can influence the outcome of an action and the outcome is not entirely predictable. The Influence-Reaction model [57] aims to represents these uncertainties. We will discuss this model in more detail later in this section.
- Dynamism: An environment can either be static or dynamic. In a static environment, the changes in the environment only occur as a consequence of the actions of an agent. In a dynamic environment, changes in the environment can happen independent of agent actions.
- Continuity: Continuity refers to the continuity or discreteness of an environment. A continuous environment influences the state of the agents in a continuous function. In a discrete environment, agents have a discrete set of predetermined states.

When mobile agents are involved, the environment will also provide the spatial information. This information can be provided by a two or three dimensional environment or the information can be more detailed and provide a rich set of geographical information, like a GIS [55].

To describe the interactions between agents and their environment better in a non-deterministic environment, Ferber and Müller have developed the influence/reaction theory [36, 57]. In this theory agents produce influences on its environment and the environment reacts to the complete set of concurrent influences.

Behaviour Next to sensing parameters from their environment, agents are capable of performing actions which result into changes in their environment. Agents decide autonomously which action to perform in a situation, Figure 2. This is described in the agents behaviour [55, 85].

While an action is an atomic event, an agents behaviour can span a longer period of time. Agents can also have multiple behaviours. Its the agents task to decide which behaviour will optimize its goals and if he can change to that behaviour at a particular time frame [85].

Often, behaviour is categorized according to its complexity. A popular categorization divides behaviour into two categories: Reactive behaviour and Cognitive behaviour [85, 33]. Reactive behaviour defines if-then rules, such that an agent can react to the current percepts considering his own beliefs. Cognitive behaviour is more complex in that agents have a set of goals as well as a set of plans that to achieve their goals. A popular model for describing cognitive behaviour is the Belief-Desire-Intention model [40]. This model divides behaviour into three parts:

- *Beliefs* The collection of internal representations an agent maintains about itself and its environment
- *Desires* A set of conditions that describe a desired end-state
- *Intentions* A set of plans to achieve the desired end-state

Russel and Norvig [73] identified more fine-grained categories: *Simple Reflex Agents, Model-Based Reflex Agents, Goal-Based Agents, Utility-Based Reflex Agents,and Learning Agents.* They are defined as follows.

Simple reflex agents select actions on the basis of the current percept, ignoring the rest of the percept history. Simple reflex agents have the simplest behaviour. Its behaviour consists only out of simple condition–action rules. A condition-action rule checks whether the condition is met based on the current perceptions and, if it is met, performs the action.

A model-based agent can handle partially observable environments. Its current state is stored inside the agent maintaining some kind of structure which describes the part of the world which cannot be seen. This knowledge about "how the world works" is called a model of the world, hence the name "model-based agent".

Goal-based agents further expand on the capabilities of the model-based agents, by using "goal" information. Goal information describes situations that are desirable. This allows the agent a way to choose among multiple possibilities, selecting the one which reaches a goal state.

A rational utility-based agent chooses the action that maximizes the expected utility of the action outcomes - that is, what the agent expects to derive, on average, given the probabilities and utilities of each outcome. A utility-based agent has to model and keep track of its environment, tasks that have involved a great deal of research on perception, representation, reasoning, and learning.

Learning agents are competent to initially operate in unknown environments and to become more competent than its initial knowledge alone might allow.

Emergent Behaviour *"Emergent behavior is commonly defined as macroscopic coherent regularities, such as identifiable distributions, coherent patterns (spatial, temporal, or behavioral), equilibria, and so forth"* [26]. In multi-agent systems, emergent behaviour is often considered to be global system behaviour that is not specified in the behaviour of the individual entities. E.g. in traffic models, we model cars such that they try to avoid traffic jams. However, due to their partial knowledge about the system, traffic jams can still occur [67].

Topology In a system that comprises multiple agents that communicate with each other, the agents are organized according to a topology. A topology describes the locations and relations of agents and shows who can send messages to whom [32, 55]. There are many different ways of representing these topologies, examples can of these can be seen in Figure 3. In some multi-agent systems, the agents interact according to multiple topologies. For example, a topology can describes the physical positions of each agent and its connectedness with the other agents, while another topology describes the social structure within the agents.

Topologies can be either dynamic or static. In a static topology, the relations and positions of each agent remains the same. In a dynamic topology, relations between agents can vary over time, as well as their positions.

A typical example of a dynamic topology is the connectivity between moving agents. Each agent has a certain range for sending messages. If two agents are withing each others range, they are able to communicate and we can say that they are connected. If the Two agents move such that they are out of each others range, they are no longer connected, thus the topology of the system changes.

An agents neighbourhood is the set of agents that are connected directly to the agent according to a topology. For example, in a spatial environment, the neighbours of an agent can be all agents that he can communicate with directly. In a social topology, the neighbours can be the agents peers and direct supervisors.

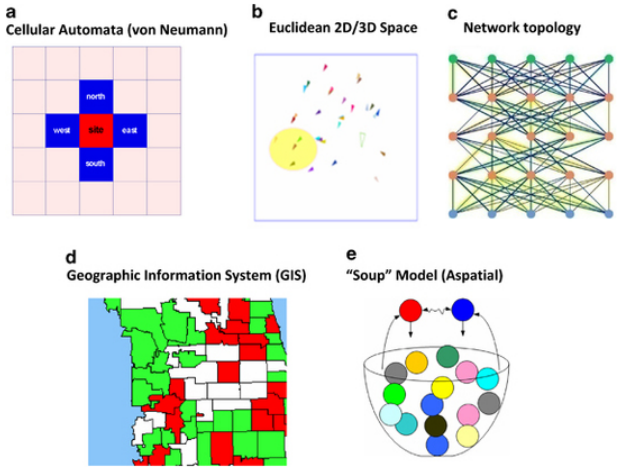

Figure 3: Different types of topologies [55]

2.6 Relation Between Agent-Based Modeling and Multi-Agent Systems

In this section, we will discuss how agent-based modeling relates to multi-agent systems. First we discuss how they agree or disagree on certain concepts. In the second section we discuss how agent-based modeling is used in multi-agent systems.

2.6.1 Agreement on Concepts

In the previous sections, we saw that both agent-based modeling and multi-agent systems agree on certain concepts from a high-level point of view. Both paradigms represent entities in the system by agents which are situated in an environment and can perform actions in the environment. Both fields mention that agents should be autonomous and communicative.

Though they agree on general concepts, both fields are very different. The ultimate goal for multi-agent systems is to design complex distributed systems [46]. The goal of agent-based modeling and simulation is to study and analyze systems [54]. If we look at the definitions by C. M. Macal we see that the first definition does not require agents to be autonomous. In the first two definitions, agents can not communicate with each other. We think the reason that these properties are less important in ABM is because the agents are not necessarily tasked with solving a complex problem that is beyond an individual's capability and an agent-based model isn't necessarily distributed. ABM tools such as NetLogo [79] and Repast [28] simulate agent based models as a single thread process as default. Since autonomy is especially needed to deal with uncertain situations in a distributed environment, it is less relevant on a single threaded simulation tool. Since agents also don't need to collaborate (e.g. in the traffic

model [67] cars do not communicate with each other to decide the best action), the need for communication is less relevant. This is in line with the description of agent based modeling in [59], which only mentions that an agent based model explicitly models individuals, their behaviour, their interactions, and their actions on the environment. That the behaviour should be autonomous is never mentioned.

2.6.2 Agent-Based Modeling for Multi-Agent Systems

The multi-agent systems approach often used to develop large and complex systems. Agents provide a great way to deal with uncertainty and allow developers to divide a large problem into smaller sub-problems [45].

A methodology for facilitating the design of complex systems is model driven engineering [74]. In model driven engineering, multiple models of the system are created. Each model depicts a certain aspects of the system. Then simulations are run to see whether the system performs correctly before implementing the actual system.

Since MAS and ABM agree on the concept of agents, it seems logical that agent-based modeling can be used for simulating MAS. However, MAS development tools like MACE [49] and its successor MACE3J [38] feature their own simulation tools for developing multi-agent systems. Our interpretation is that while developing multi-agent systems, a more complex model is necessary than provided by ABM tools such as NetLogo and Repast. Multi-agent systems need take the problems of distributed systems into account, such as fault tolerance.

3 Tool Comparison

With the advent of agent-based modeling (ABM), many tools have been developed to support this modeling paradigm. However, a universally accepted formal semantics remains absent. This situation results in a proliferation of tools that were developed according to different interpretations of the paradigm [9]. Moreover, the lack of formal semantics imply that the semantics of a model are determined by its implementation in a specific tool, making them platform dependent. Platform dependency, in turn, affects the reproducibility of simulation across platforms. This is a very undesireable property for modeling and simulation platforms.

The aim of this study is to identify whether ABM tools implement the key features of agents and ABM, as well as determine whether reproducibility can be achieved across platforms.

To determine this, an example model will be implemented in each of the tools under study. A set of scenarios will be determined and run on each of the implementations. Subsequently, we can determine to which precision the scenarios are repeatable and reproducible among tools.

3.1 Case Study

The aim of the experiment is to determine which features, that are often related to the ABM paradigm, are included in each of the studied tools. To do this, we created a feature model that represents the features that are related to the ABM paradigm. For each tool, we instantiated the feature model.

3.1.1 Tools Under Study

For the scope of this project we only reviewed a selection of tools that can be used for ABM. The tools we will review in this study are: DEVS (resource centric), DEVS (entity centric), NetLogo, Repast, and SARL.

DEVS [83] have been used as an implementation tool for ABM with the aim of developing a formal semantics for ABM [80, 17]. However, the DEVS formalism is developed to support the discrete event modeling paradigm. We want to analyze how well we can express agent-based models in this formalism. For this we will study two common approaches to DEVS modeling: Resource centric and Entity centric DEVS.

Railsback et al. [70] analyzed a set of the most popular tools for agent-based modeling. They concluded that Repast [28] is the most complete framework in java. They also mentioned NetLogo [79] to be a very popular tool for educational purposes. NetLogo features an easy to use interface and an extensive set of agent-oriented capabilities. Since both tools were often mentioned in literature regarding ABM, we chose these tools to represent specific ABM tools.

Last, we reviewed SARL [71], which is an agent-oriented programming language. This language however has no built-in support for simulating agent system, however a lot of concepts of MAS and ABM are first-class abstractions.

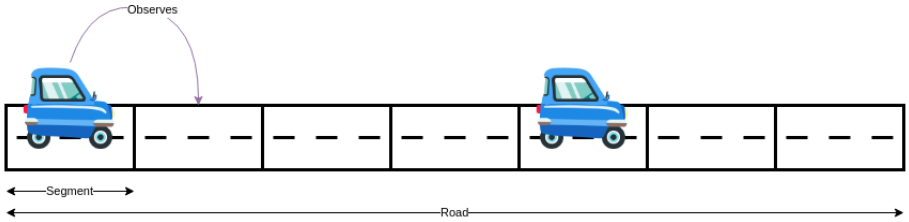

Figure 4: A simple diagram depicting the example model

Since SARL seems very promising for implementing agents, we wanted to see if we can use this tool for simulation purposes in the future.

3.1.2 Example Model

To get a better understanding of the tools, we implemented an example model in each of them. The example is chosen to be complex enough such that it shows most of the platforms capabilities. As example, we modeled a traffic situation where cars travel over a road. This road is divided into segments and when a car enters a segment, he can observe the next segment and adjusts his speed accordingly. The model was inspired by an assignment from *Modeling of Software Intensive Systems* course at the University of Antwerp. In this section, we will give a detailed explanation of the model.

In the model a road of length l is divided into n segments, each of length l/n. Each segment has a *max_speed* which is the maximum allowed speed on the segment. This maximum speed depends on the speed-limit of the road and the condition of the road-segment (e.g. potholes can decrease the maximum allowed speed).

A car has following attributes:

- *carID*: A unique identifier
- *preferred velocity*: The preferred velocity with which the car advances
- *dv_pos_max*: This value represents the maximal difference in speed in one section, when the car is speeding up
- *dv_neg_max*: This value represents the maximal difference in speed in one section, when the car is slowing down

Cars arrive at the first section with random intervals, sampled between the minimum inter arrival time and the maximum inter arrival time. Then, they will run over each segment consecutively, starting at the first segment. At each segment, the car will observe the next segment once. When the road is clear, the car will try to drive at the maximum speed that he can attain (note that this is the minimum between its preferred velocity and the speed limit of the road segment). When their is another car present in the next segment, the car will

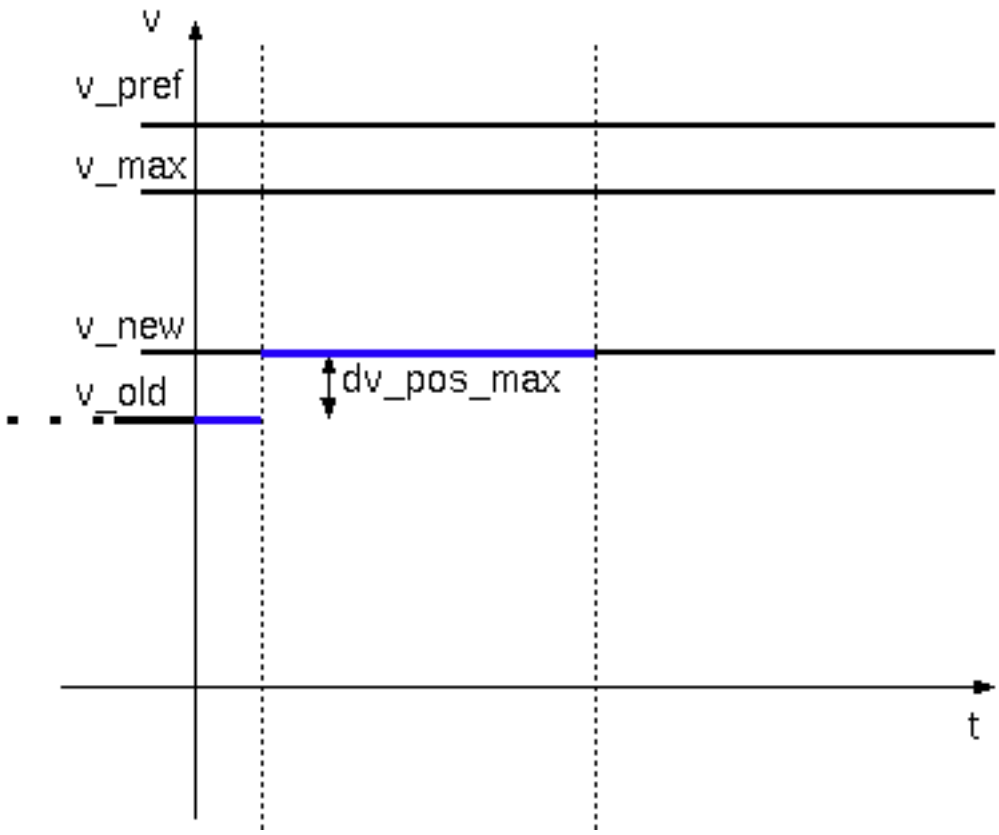

Figure 5: Graph of change in velocity

adapt its speed to avoid collision. Each car computes or remembers at which point in time it will advance to the next section, according to his current speed. A car can thus adjust its speed according to the time-until-departure of the car in the next section.

In Figure 5, we see how a car changes velocity on a segment. First the car determines the target speed. This is the minimum of the preferred speed, the maximum speed, and the speed that prevents collision. Then the car will add or subtract at most dv_post_max or dv_neg_max from his speed.

When two cars are in the same segment, they collide. When this happens, both cars set their current speed to zero and their time_until_departure to infinity. When cars collide they do not try to restart and remain in the same segment for the rest of the simulation.

3.1.3 Scenarios

Since we are also interested in the reproducibility and the repeatability [**?**], we will analyze 4 scenarios in each of the models that were implemented. Here follows a list of the scenarios that were analyzed:

- **1 car, standard configuration** The first scenario tries to show the correct behaviour of a single in a basic situation. 1 car will go through all the segments with a random preferred velocity in the range of [10, 15], and a maximum velocity of 13 m/s.
- **Car stream, standard configuration** The second scenario introduces a variable arrival time of cars. New cars will arrive at the start of the road with an inter-arrival time uniformly sampled from [10-20] seconds. This scenario aims to show proper behaviour when cars will almost never interfere with each other.
- **Car stream, strong variability on IAT** In the third scenario, we introduce a strong variability on the inter-arrival time of cars. The IATs will be uniformly sampled from a range [1-20], now cars have a higher chance of interacting with each other, however, since their speed will not vary a lot, we expect not much congestion or collisions.
- **Car stream, strong variability on preferred velocity** In the final experiment, the preferred velocity will vary a lot as well. This will greatly increase the chances of collision or congestion. The preferred velocity will be sampled from the range [1-15]

We set the maximum speed on a segment to 13m/s, the dv_pos_max to 2.35 and the dv_neg_max to 2.5. The road itself is 200 meter long and is divided into 50 segments. Figure 6 shows the average travel time on each of the tools for seeds [0,1,2,3,4] for scenario 2. The output traces found in the download links were all run with seed 0.

3.1.4 Feature Model for ABM Tools

In Section 2, we analyzed the key features of agents and agent-based modeling. To analyze how well the tools represent the paradigm of agent-based modeling, we created a feature diagram for ABM. Later, we will instantiate a concrete feature model for each of the reviewed tools.

3.2 DEVS - Resource Centric

DEVS [83] is a modeling formalism that was developed to model systems according to the discrete event paradigm.

DEVS is a very generic formalism, which is aimes to be a common implementation platform for all discrete event models [84]. This entails that often there are many ways to implement the same model. In this study, we analyzed two popular approaches: Resource centric, and Entity centric.

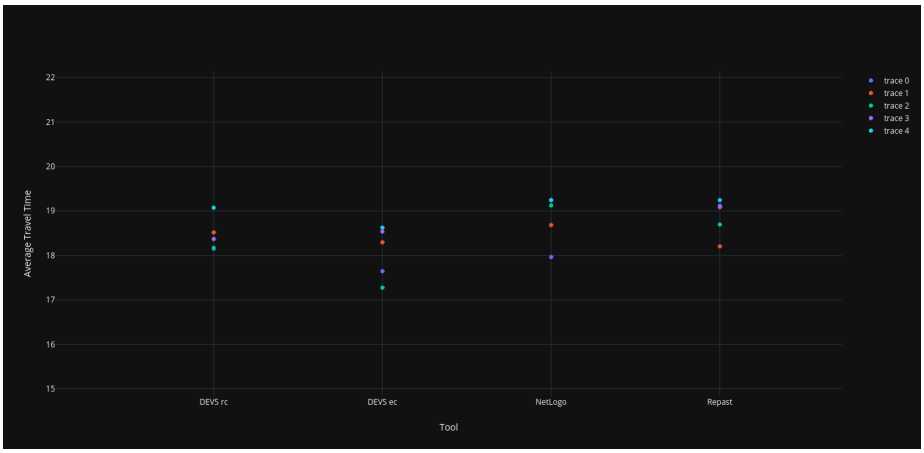

Figure 6: A plot of the average travel time of cars in scenario 3.

In resource centric DEVS, the behaviour of resources is modeled explicitely. The entities, which act upon the resources, are passive objects that are passed through the resources.

To implement DEVS models, we used the tool Python PDEVS [82]. We used version 2.4.1.

3.2.1 The DEVS Formalism

Classic DEVS features two types of models, atomic DEVS models and coupled DEVS models. Atomic models are *the indivisible building blocks of a model* [83] and model a single entity of the system under study. Atomic models can then be connected in a coupled model.

An atomic DEVS model is a tuple $< S, ta, \delta_{int}, \delta_{ext}, Y, \lambda, I, O >$, where:

- S is a set of internal states of the model.
- I is the set of input ports of the model.
- O is the set of output ports of the model.
- Y is the set of allowed messages.
- ta is a function $ta : S \mapsto \mathbb{R}$, which maps the internal state of the model to the time to the next internal transition.
- δ_{int} or the internal transition function maps each internal state to the subsequent internal state.
- δ_{ext} or the external transition is a function $\delta_{ext} : (Y \times I) \mapsto S$, which given a message retrieved at an input port returns the next state.

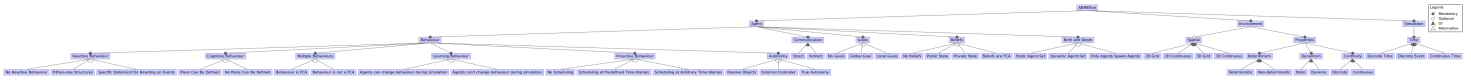

Figure 7: The feature model

- λ or the output function is a function $\lambda : S \mapsto (Y \times O)$ which maps each internal state to an output message which is send via a certain output port.

A coupled DEVS model consists of a set of atomic or coupled DEVS. Within the coupled model, models can be coupled by connecting their ports. Above that a coupled DEVS model also has a set of input and output ports. This allows the creation of hierarchical models.

3.2.2 The Example Traffic Model

To implement the example model in DEVS, we started by creating the messages that can be send by the models. There are two different types of messages in the model: queries, and acknowledgements, Figure 8. Whenever a car arrives at a road segment, the segment will send a query to the next segment. A query does not contain any information. The next segment will answer a query with a query acknowledgement (QueryAck), which contains the time until departure of that segment. If no car is present, the time until departure is 0. If a car is present the time until departure is computed with the current speed of the car and remaining distance.

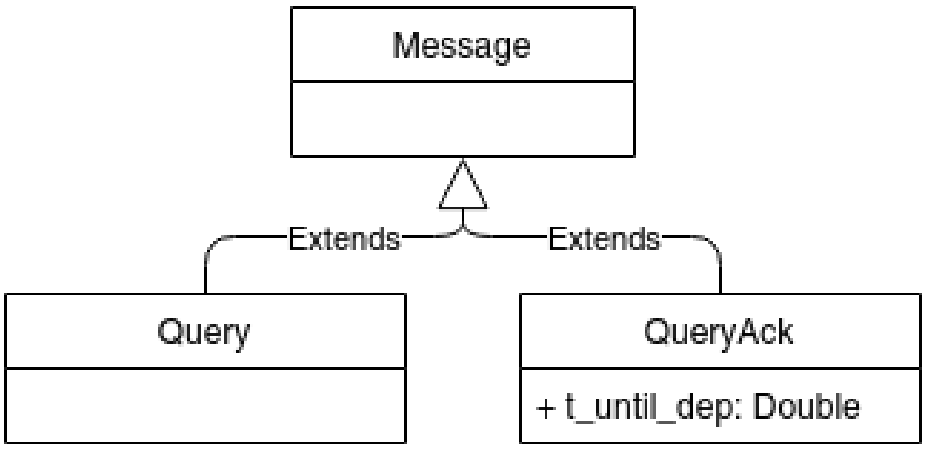

Figure 8: Message types in the model

Since the model we implemented is resource centric, cars are passive objects that are passed around by the DEVS models. Cars can therefor also be send as a message. A car has following attributes:

- car_id, a unique identifier
- v_pref, the preferred velocity of a car
- dv_pos_max, the maximum increase in velocity a car can undergo in one segment

- dv_neg_max,the maximum increase in velocity a car can undergo in one segment
- departureTime, the current simulation time when the car departs at the start of the road
- arrivalTime, the current simulation time when the car arrives at the end of the road
- velocity, the current velocity of the car

To model the road, road segments are modeled by atomic DEVS models. Each model has three states: poll, final, and empty. In the empty state, there is no car present on the segment. When a car arrives at the segment, the segment reaches the poll state. In this state the car waits to send his query to simulate the observe delay. This is the time it takes for the car to observe the next section and react to it. When a query acknowledgement is received or there is no time to react the segment reaches the final state. In the final state, the car proceeds to the end of the segment and is passed on to the next segment. Furthermore, a segment has an input and output port for queries and for acknowledgements. Segments also have an input and output port for passing cars. Figure 9 shows how multiple segments can be connected.

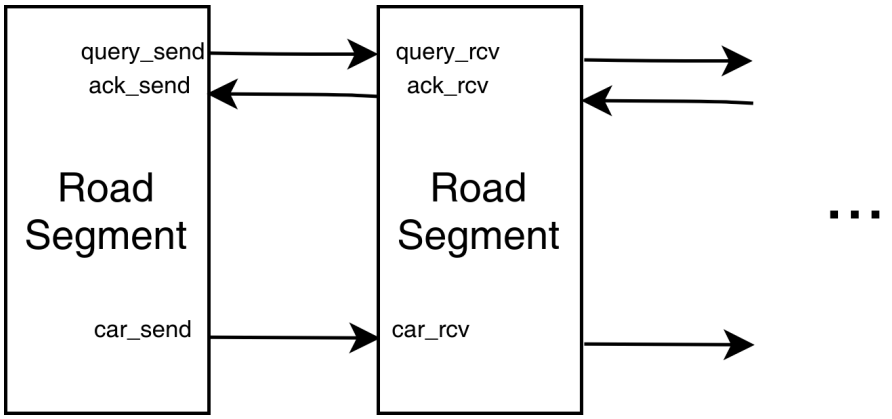

Figure 9: Connection of multiple segments

Cars need to be generated and collected. Therefor we created the generator model and the collector model. The car generator will create a car at each internal transition and send it to the first section. As time-advance, the generator samples a random number from [IAT_min, IAT_max].

3.2.3 Running the model

To run the model, we used python version 2.7 and python PDEVS version 2.4.1. Installation instructions for the tool can be found at the documentation site.

The model can be downloaded at this link. To run the model, run the simulation.py as python file. Parameters for the simulation can be set in the simulator.py file.

The outputs of the scenarios can also be found at this link.

3.2.4 Feature Overview

To see how well this approach is suited for agent-based modeling and simulation, we instantiated the feature model in 3.1.4 for resource centric DEVS. The resulting feature diagram can be seen in Figure 10.

In the model, the agents are passive objects that passed passed around as messages. Agents thus have no behaviour of themselves. It is thus impossible to define autonomous behaviour, since the agents actions are completely controlled by the environment (the road segments). Cars can have a representation about their environment, so it is possible to add beliefs to agents. However, there is no concept of explicitly modeled goals in this formalism.

DEVS is a deterministic formalism [83]. This means that the internal as well as external transition functions will always return the same state given the same parameters. It is thus impossible to a non-deterministic environment. However, the tool we used to implement the model offers a way to bypass this. Because Python PDEVS is a python library, we can use the python random number generator to randomize the outcome of the transition functions.

This approach does not allow users to create static environments. In static environments, changes in the environment only happen as reaction to actions by the agents. In resource centric DEVS however, it is the environment that acts on the agents.

DEVS is a formalism to model discrete event models. This is apparent from the time advance function, which allows models to schedule events at arbitrary time frames. We can limit the choice of time frames to predefined equi distant time slots to create a discrete time model. However it is impossible to create a continuous model.

3.2.5 Repeatibility of DEVS

Here we provide the results of the repeatability of the scenarios discussed in 3.1. For this experiment we simulated each scenario 10 times and compared their output traces. The tool we used for DEVS, python PDEVS, provides an informative output trace of each event during the simulation. We used this output trace for comparison. To check for differences in the output trace, we used the diff tool on linux.

DEVS is a deterministic formalism. If we use the same random number generator with the same seed, we expect the output traces to be identical.

After running each of the scenarios, we saw that experiments in this tool are repeatable on trace level.

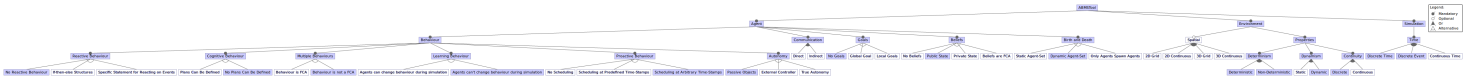

Figure 10: The feature model for resource centric DEVS modeling for ABM

3.3 DEVS - Entity Centric

Another approach in modeling the given system in the DEVS formalism is an entity centric DEVS model. In entity centric DEVS models, we model the entities in the system rather than the resources. In the example of the traffic model, this means that the behaviour of the cars is modeled in a specific Car model rather than the road segment models.

A problem now is that we need to add new cars during the simulation of the model. The DEVS formalism does not support dynamic structures. There is, however, an extension to DEVS that allows for dynamic structures [83].

Since Python PDEVS also feature dynamic structured DEVS [83, 13], we used the same tool for this model.

3.3.1 Dynamic Structured Devs

In this section, we will discuss the extension that is needed for standard DEVS models to include dynamic structures.

In dynamic structure DEVS, *the model configuration is seen as part of the state of the model* [83]. This allows the configuration to change during the simulation. Apart from an internal transition, models now have a model transition, which is triggered at every state change. In the model transition function changes to the model configuration are performed. Coupled DEVS models don't have a state in standard DEVS, however in dynamic structure DEVS, they do. Since coupled models don't have an internal or external transition function, their model transition function needs to be triggered in a different way. In Python PDEVS, this is done by making the model transition function return true or false. When true is returned, the model transition function of the first encapsulating coupled model is invoked. When false is returned, the execution of the model transition function in models higher in the hierarchy is prohibited.

Possible changes are:

- Add Model: Adds a new model to the configuration
- Remove model: Removes a model from the configuration
- Connect ports: Connects an input port of a model with an output port of another model
- Disconnect ports: Breaks the connection of two connected ports

3.3.2 The Example Traffic Model

Here we will describe how we implemented the traffic model in dynamic structured DEVS.

The central entity in the model is the car. Each car in the system is represented by a dedicated atomic DEVS model. Next to the car models, we also created a model that represents the environment of the cars, the road model.

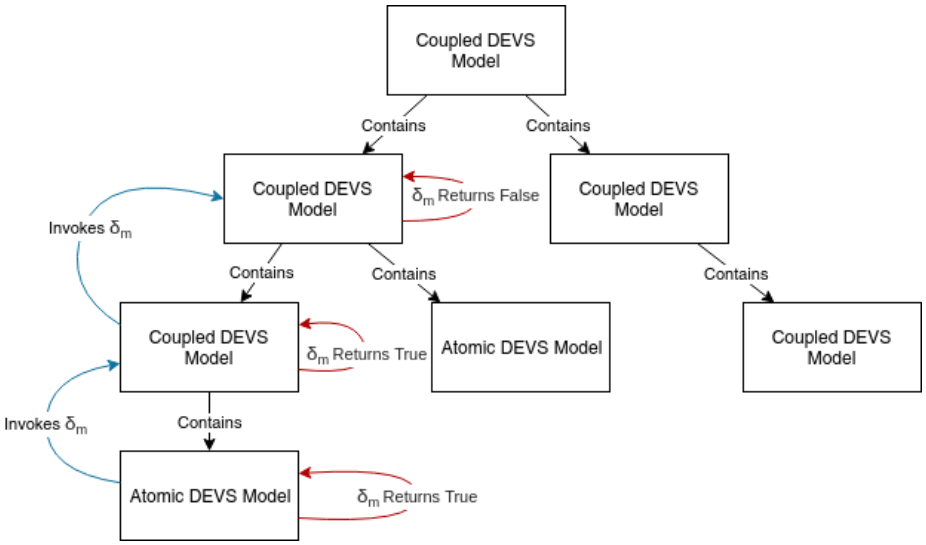

Figure 11: Invocation of the δ_m function in an atomic model

(a) If segment_length/velocity > obs_delay (b) If segment_length/velocity ≤ obs_delay

Figure 12: Flow chart depicting a cars behaviour

A car has the same attributes as in the resource centric DEVS model. When a car enters a segment, it is in the arrival state. It then waits the observe delay before querying the next road segment. Each car is connected to the road with a dedicated input and output port. To observe the next road segment, the car sends a query to the road model, which responds with a queryAck containing the time until departure. Based on the response, the car adjusts its speed and based on his new speed and the remaining distance he has to travel, he waits before departing to the next section. To advance, the car emits an advance message to the road model. If the car has advanced successfully, the road sends an advanced message to the car. Whenever a car enters a segment or adjusts his speed, it sends a message to the road with the updated time until departure. A flowchart of the car behaviour can be seen in Figure 12. If cars collide, the road will send a collide message to all cars involved.

The road model keeps a dictionary from each segment to the list of cars in that segment. The road model also has an input and output port for each of the cars in the system. The road model keeps track of the positions of each of the cars, and the time until departure of each of segments. It answers to queries from the cars. This resembles agent behaviour really well. The cars sense the environment by sending queries and perform actions on the environment by

sending event messages, such as the advance message.

We also created a generator model, which only purpose is to trigger the model transition function of the encapsulating traffic model. This model then creates a new car and connects it to the road appropriately.

3.3.3 Running the model

To run the model, we used python version 2.7 and python PDEVS version 2.4.1. Installation instructions for the tool can be found at the documentation site.

The model can be downloaded at this link. To run the model, run the simulation.py as python file. Parameters for the simulation can be set in the class TrafficModel.py file.

The outputs of the scenarios can also be found at this link.

3.3.4 Feature Overview

We will now discuss which features, defined in 7, can be implemented in entity centric DEVS.

First of all, we noticed that, since DEVS is not developed specifically for ABM, that agent related concepts are no first-class abstractions in this formalism. However, in entity centric DEVS models, structures can be defined that closely resemble agent-based entities. The resulting diagram can be seen in Figure 13.

Atomic models can be seen as autonomous entities. There is no way for other models to directly trigger a state transition in the atomic model, except from sending a message. Since the external transition, which is triggered when a message is received, considers both the message as well as the current state of the atomic DEVS model, it deliberates to which state it will transition. This is clearly autonomous reactive behaviour. On top of this, it is possible to define pro-active behaviour. For pro-active behaviour, the entity needs to be able to schedule certain actions by himself. This allows it to exploit opportunities even when no events happen. Defining pro-active behaviour can be done with the internal transition function. During an internal state transition, the model can schedule the next internal transition, which will be triggered when no external transition happens.

It is possible to include direct as well as indirect communication. Direct communication can be achieved by giving models that represent an agent dedicated ports to send messages to other agent models. Indirect communication can be achieved by creating ports that are connected to models that represent the environment.

To represent the environment, we advise to create a dedicated model. Agents can then be connected through dedicated ports. To simulate the perception of the environment by the agent, the environment can send partial information to the agent periodically, or the agent can query its environment. As opposed to resource centric DEVS, we can now define static behaviour. Agents can act on

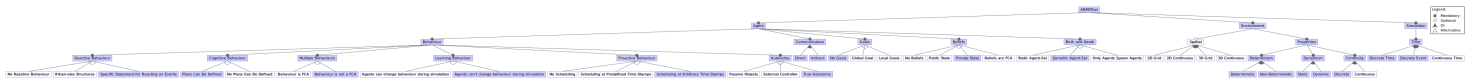

Figure 13: The feature model for entity centric DEVS modeling for ABM

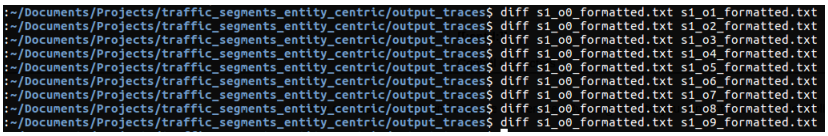

Figure 14: Output of the diff algorithm on the first scenario

their environment by sending event messages, such as the advance message in our example.

3.3.5 Repeatibility of DEVS

Here we provide the results of the repeatability of the scenarios discussed in 3.1. For this experiment we simulated each scenario 10 times and compared their output traces. The tool we used for DEVS, python PDEVS, provides an informative output trace of each event during the simulation. We used this output trace for comparison. To check for differences in the output trace, we used the diff tool on linux.

In Figure 14, we included the output of the diff algorithm of the first scenario. The tool did not find any differences in the tool, however we had to format the output first. The output trace of python PDEVS included the physical addresses of the objects, since they are randomly determined by the operating system, they varied among simulations. To obtain our output, physical addresses need to be abstracted first.

As for the other scenarios, no significant differences were found, apart from different inter-leavings of events that happened on the same time-stamp. This was due to the dynamic structure. The select function ordered the events on their physical address. Since models are created at run-time, their physical address were not deterministic. After resolving this problem by ordering events on car ID's, we achieved identical output traces. We did had to create a custom random number generator, such that each car samples from the same distribution with the same seed.

We can conclude that entity centric resources is deterministic and experiments are repeatable on trace level.

3.4 NetLogo

NetLogo is the first tool in this section that is considered an ABM tool [70]. Its main focus lies on mobile agents in a 2 dimensional environment [79, 78]. They extended the functionality to incorporate 3 dimensional environments.

NetLogo is mainly used for research and educational purposes [70]. NetLogo features a domain specific language and a GUI to facilitate development. Due to its easy interface and complete packaging, NetLogo is the most popular tool for ABM [70].

To implement our model we used NetLogo version 6.0.4 which can be downloaded here.

3.4.1 The Netlogo Tool

In this section we will discuss how NetLogo works, which abstraction it made, and how it resembles the ABM paradigm. We will first discuss how netlogo incorporates agents and environments, then we will talk about its interface, since it is one of the key features of the tools. Finally we will discuss the domain specific language that NetLogo provides to define agent-based models.

Agents NetLogo is built on the concept that "the world is made up out of agents" [5]. This means that not only the individual entities are agents, but also the links between agents are in turn agents, as well the environment and the observing entities.

NetLogo divides agents into four categories: turtles, links, patches, and observers. Turtles are agents that represent 'living' entities in the space. They have a position and can freely move around through the environment.

Links are agents that connect two other agents. It does not have a position, but is represented by a line between the two agents that are connected by the link.

A patch represents a cell in the grid that makes up the 2D environment. Patches have an x and a y coordinate, but cannot move.

An observer is an agent that is not present in the world, but rather observes all other agents. The observer is also responsible for giving directions to all other agents.

Environment As mentioned above, NetLogo features a spatial environment for turtles to roam in. The standard version of NetLogo features a 2D environment, but when installing NetLogo, it also install NetLogo3D. NetLogo 3D features a 3-dimensional environment and graphical support.

The environment's only responsibility is to keep track of the positions of each of the agents. Agents have an absolute position in the environment, provided by their x and y coordinate, as well as a grid based position, given by the patch on which the agent is positioned.

Interface What makes NetLogo such a popular tool, is its easy to use interface [70]. Figure 15 shows the initial screen, when starting up NetLogo. The interface features three tabs: the interface tab, the info tab, and the code tab.

In the interface tab, developers can specify the interface for future users of the model. The big black box that is provided for each new model renders the agent system. When turtles are created they will appear in the box at their appropriate position. Developers can add components to the interface.

The first components are buttons. When adding a button, you can assign a method to it that is invoked whenever the button is pushed.

The second type of components are input components. Examples of input components are sliders, choosers, and input prompts. These components can be used to ask for user input. Each of these components defines a global variable that can be accessed in the code.

The last type of components are output components. These components output results from the model to the user. Examples of these components are plots and monitors.

Each of these components can be placed arbitrarily on the window, as well as resized.

The info tab is used to describe the model, as well as give direction on how to use the model. The last tab is used to implement the model using the NetLogo domain specific language.

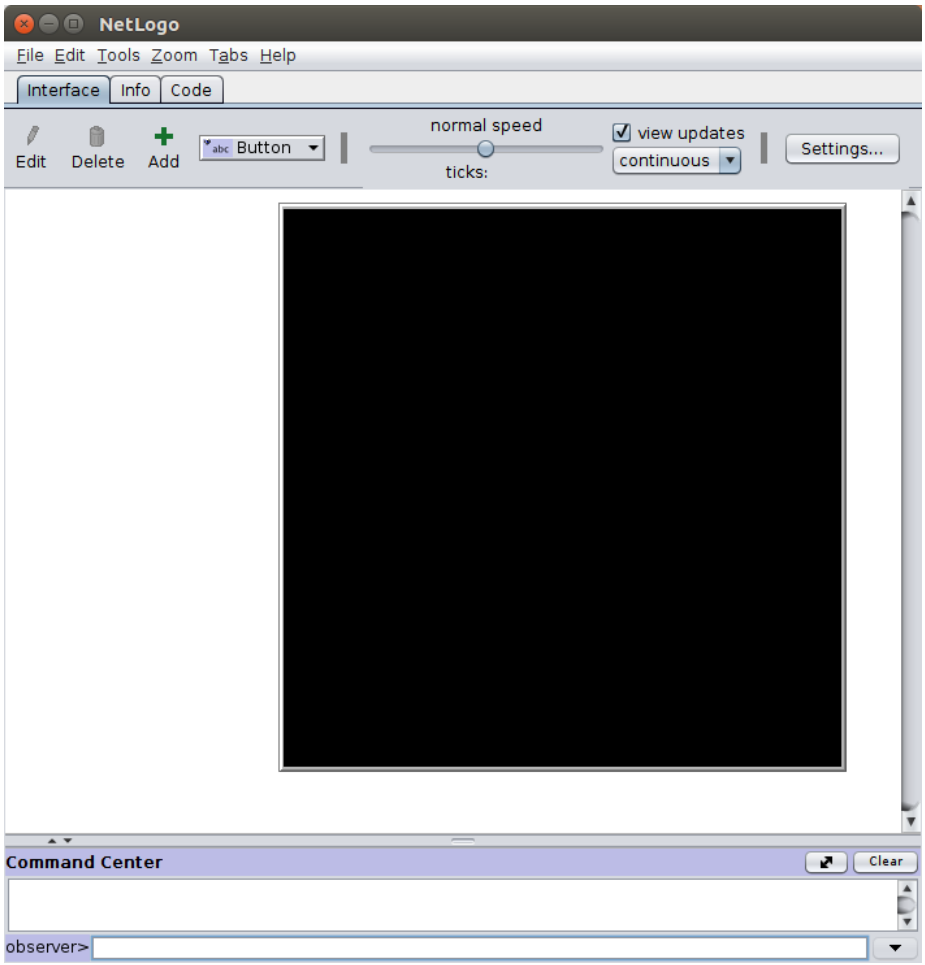

Figure 15: Starting screen of NetLogo

Domain Specific Language NetLogo features its own domain specific language for implementing models. Here, we will give a short overview of its

features, based on the documentation [5].

NetLogo allows different types of turtles called breeds. To create a breed, you can use the breed statement:

breed [cars car] : defines a breed called cars

Each breed has a set of variables that represent that entities state. These variables can be defined with a breed-owns statement:

cars-own [motor, velocity, petrol-level]: defines motor, velocity, and petrol-level as breed-variables for the cars breed

NetLogo provides two types of methods. The first type of method is confined within a *to [params] ... end* statement. These methods do not return a value and thus are only a means to altering the state of the model. The second type is confined within a *to-report [params] ... end* statement. Report is the NetLogo equivalent of return. These functions thus return a value with a report statement, but, on top of that, can also have side effects.

NetLogo does not support breed specific methods, but you can execute a method in the context of an agent with the ask statement.

ask cars [move-forward] will execute the move-forwad function for each car present in the model. If, for example, the method uses the motor variable, it will use the value of the car for which the method is called.

NetLogo features specific built-in methods for turtles, patches, and links. An example is the forward method for turtles. Forward x advances the turtle in a forward direction over distance x.

NetLogo implements the discrete time paradigm. Time is divided into equidistant intervals called ticks. At each tick, an advance method can be called. The method then simulates everything that happened during the last time interval and then advances the tick count by calling the tick method. When the tick count needs to be updated with a value other than 1, you can call the tick-advance method.

3.4.2 The Example Traffic Model

For the traffic model, we created 3 breeds: a Car breed, a Divider breed, and a Road breed. The cars specific variables can be seen in Figure 16. Dividers remember the time until-departure of the next segment. Dividers can be interpreted as flags at the end of each segment. Road agents are a link that links two subsequent dividers.

NetLogo has no obligated method for implementing the run-time behaviour of the model, however a standard approach seems to be accepted by the NetLogo community. This method is used in the models library provided with the tool. A setup button is added to the model which invokes one or more functions that initialize all parameters of the simulation. A go button is added which invokes

```
cars-own [
  id

  t-till-observe

  velocity
  v_pref

  dv-pos-max
  dv-neg-max

  perform-action
  next-divider
  x-rm

  departure-time
]
```

Figure 16: The internal variables of a car

repeatedly a method that simulates a single time-frame and advances the tick-count with 1. We needed to have a smaller tick size, so we used the tick-advance function instead.

The setup function will initialize $n + 1$ segment dividers and connects them two by two with roads. The setup function also initializes all global variables that are not defined by the user.

NetLogo only support discrete time and no discrete events. To spawn cars at a random time interval, we introduced a global variable ticks-till-next-car. In the go procedure, which simulates one time frame and advances the ticks with tick-size, tick-size is subtracted from this variable. If ticks-till-next-car becomes negative a car is spawned and ticks-till-next-car is reset to a new random value. We used the same technique to implement the observer delay of a car.

Since cars can only perform one action on each segment, we created the perform-action field for cars. When a car observes the next segment and adjust his speed, it sets this field to false. This way the remembers that he already performed his action.

Cars also remember what the next their next divider. At each time-step, he will look at the closest divider in front of him. If this one is different from the current next-divider, the car knows it moved to the next segment and resets the appropriate values.

In Figure 17 we show how the go procedures simulates a single time-frame.

3.4.3 Running the model

To run the model, we used NetLogo version 6.0.4 Installation instructions for the tool can be found at the documentation site.

The model can be downloaded at this link. To run the model, load the .nlogo file in NetLogo. Then press setup to initialize the model and go to run it. Parameters for the simulation can be set in the NetLogo interface.

The outputs of the scenarios can also be found at this link.

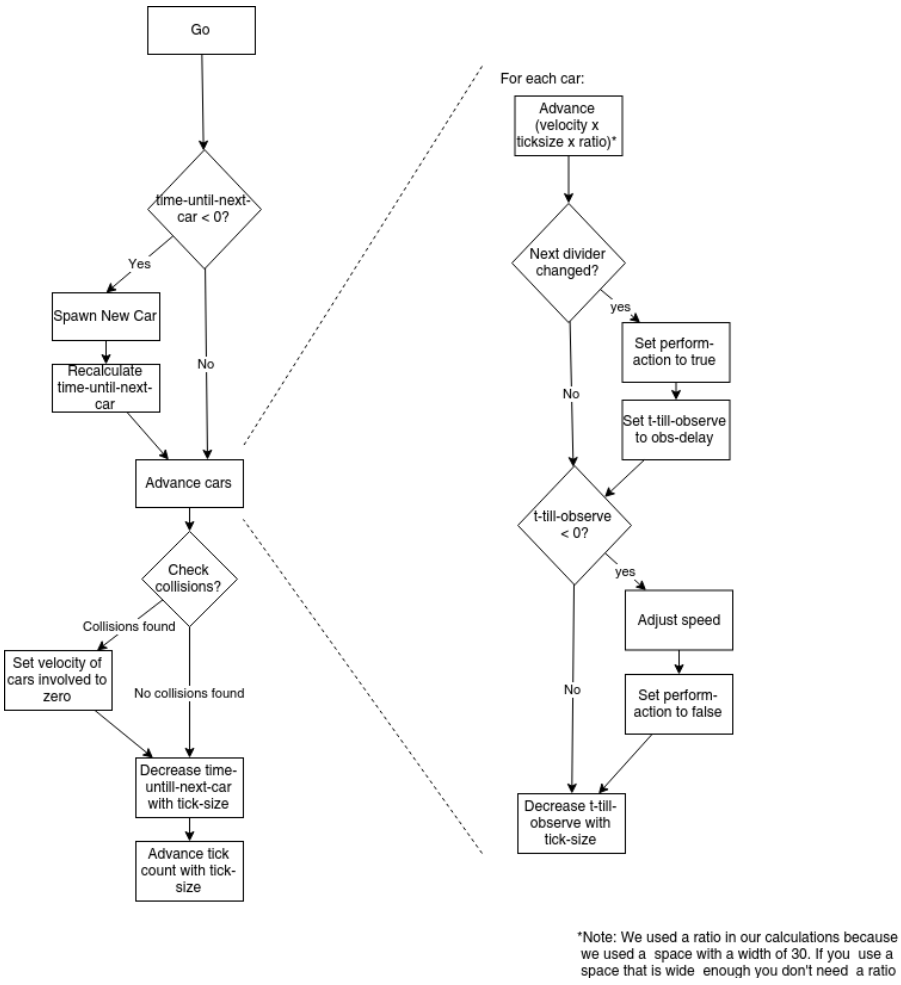

Figure 17: Flowchart depicting the functionality of the go procedure.

3.4.4 Feature Overview

In this section we will discuss the ABM features in NetLogo. The diagram can be seen in Figure 18.

We found no way to create autonomous behaviour in NetLogo. To achieve autonomous behaviour, we expect that only an agent itself can decide which action to take, based on incoming events or messages. In NetLogo, agent behaviour is defined by a central script. This script is then executed by the observer agents which directly directs the agents with the ask statement. While the ask statement suggests that agents can choose not to perform the actions, there is no way to do this. A statement such as *ask car1 [advance]* is equivalent to a remote method invocation in object oriented programming. This is not autonomous, since the object is controlled directly from an external source [85].

In NetLogo, we found no way to set up a dedicated channel between two agents. Direct communication is thus not possible. NetLogo also does not provide a way to create messages. Indirect communication can be achieved by creating dedicated message breeds.

As for the environment, we can only use static environments, since the environment only consists out of the positions of each agents. Since the environment can not change these position itself, we can not define a dynamic environment. For the same reason we can not defin non-deterministic environments.

NetLogo only allows discrete time modeling, since time is divided into ticks.

3.4.5 Repeatibility of Netlogo

To achieve repeatability, models implemented in NetLogo need to be deterministic. According to the documentation, NetLogo is deterministic when a random seed is set at setup with the *set-random-seed* method. However, there are some exceptions:

- The use of *every* and *wait*, such that it affects the outcome of the simulation.
- Using different versions of NetLogo.
- Random hardware failure or human errors in the model, NetLogo, or the java runtime-environment.

In Appendix A, an example is included of a NetLogo model that conforms to the first exception. We ran 10 simulations of this model. We ran them all on the same machine, until tick 702744. As we can see in the output of each run, in Table 4, the simulation produced a different output under the same conditions. We also observed the number of outputs increase as we set the tick slider to slower, vice verse for faster. This indicates that the every statement uses real time instead of the virtual simulation time. This makes this statement a bad choice for any model.

In our model, we omitted the use of wait and every. To check whether our model is repeatable in NetLogo, we ran each of the scenarios in Section 3.1 ten

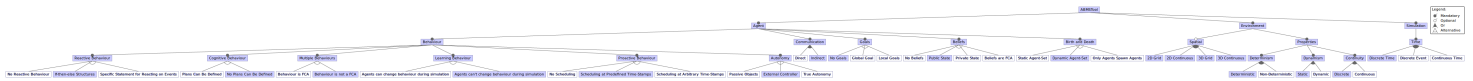

Figure 18: The feature model for NetLogo

times and compared the output traces. For the output trace, each car prints the current tick-count when he advanced to the next segment. When a car is finished, it outputs the travel-time of the car. When two cars collide, they both output that they collided. When a car is finished it outputs its travel time. To compare the output traces, we used the linux diff command to check for differences in the output.

After running the experiments, all output traces of the same scenario where identical. All experiments were run with the same random seed, 0, for the global random number generator. We can conclude that simulations of the same scenario are repeatable when we do not include the exceptions NetLogo provides on their website.

3.5 Repast

Repast [28] started out as a java implementation of the swarm agent-based modeling tool. During development, the development team decided to go their own way, however, repast features most of the functionality of swarm. Repast is regarded as a one of the most well developed tools for agent-based modeling [70]. Developing models in repast can either be done in java or in their domain specific language ReLogo. We decided to use ReLogo, since we expect it to represent repasts functionality and semantics the best.

3.5.1 The Repast Framework

The Repast framework uses a very similar approach to creating and simulating models as NetLogo. E.g. Repast also uses turtles, patches, links, and observers. We will know discuss the similarities and differences between the tools.

Interface Like NetLogo, Repast features a graphical user interface to simulate the model. In Figure 19, we see the interface after initialization. In contrast to NetLogo, the interface can not be customized by the developer. Also the model is developed outside of the interface, in the eclipse IDE [2]. When the model is run, Repast starts up the interface.

Just like NetLogo, Repast's interface provides a rendering of the model. When turtles are created, they will appear in this screen at their appropriate position.

To gather user input, the interface provides a User panel. In this panel, sliders and input prompts can be defined. Since we cannot alter the interface, these sliders need to be defined in the code. These can be added in the UserGlobalsAndPanelFactory, a standard class that is created for each new model. Here components as well as global variables can be created with built-in methods.

The interface allows three possible execution methods. Run allows users to run the full simulation. A special go function will be called repeatedly. *Step run* allows users to run the simulation stepwise. Pressing the step run button executes the go button once. The last option is the *batch run* option. This allows users to define different simulations of the model and run them in batch.

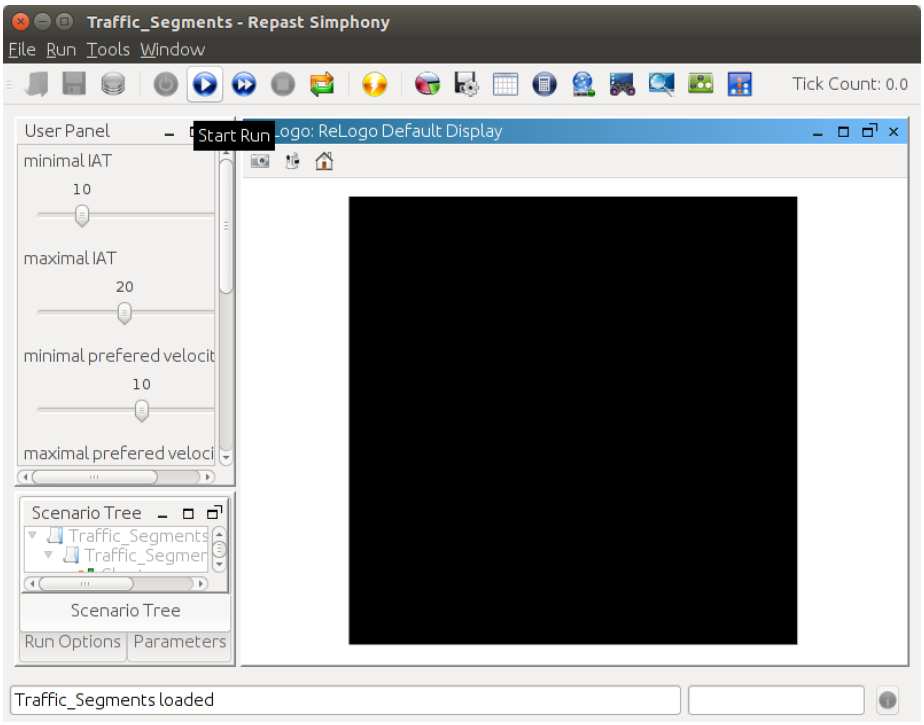

Figure 19: The RePast model interface after initialization

We did not use the last method for this study. Next to the run options, Repast has a lot of export possibilities.

Environment The environment provided by Repast is similar to that of NetLogo. It is a spatial environment where agents have both an absolute position as well as a patch on which they are located.

Relogo To facilitate the development of models in the Repast framework, the ReLogo language was developed [66]. This domain specific language is fully compatible with java, but treats Repast specific artifacts as primitives. A full list of primitives can be seen in [7]. We will not go into details into these concepts, since they are very similar to NetLogo.

Next to the UserGlobalsAndPanelFactory, a UserObserver is instantiated for each model. This agent's task is to instantiate other agents and direct their actions. This observer is the Repast equivalent as the observer in NetLogo which is implemented in the script in the code tab. In this observer, two special functions need to be defined. The first one is the setup function, which is annotated with *@Setup*. This function is run before the model is simulated and is used to initialize all necessary parameters and create an initial set of

agents. The other function is the go function, annotated with *@Go*. The go function handles a single time frame of the simulation. This function is invoked repeatedly when running the simulation or once when the step run button is pressed.

Where ReLogo differs vastly from NetLogo is the object oriented capabilities. Instead of creating breeds, a class needs to be created, which extends one of the standard type agents. This allows developers to create agent specific methods. In NetLogo, every function is considered a global function, which can be executed in the context of an agent. The increase in modularity makes ReLogo better suited for very large projects.

3.5.2 The Example Traffic Model

Since NetLogo and Repast are very similar, our implementation of the traffic model is also very similar. We created the same types of agents: cars, dividers. We omitted the connectors. Since cars move over a straight line and cars never connect to the road, these links are unnecessary. For the implementation of the go function, we based ourselves on the flow chart in Figure 17.

The difference with the NetLogo is that we moved the functionality that handles the advance car to the car. Cars now have a step function, which is called by the go function.

For a more detailed explanation, we refer to the NetLogo implementation, Section 3.4.2.

3.5.3 Running the model

To run the model, we used Eclipse IDE for Eclipse Committers version 4.9.0 and Repast Symphony version 2.6.0. Installation instructions for the tool can be found at the documentation site.

The model can be downloaded at this link. To run the model, you can load the project in your eclipse workspace. Then run the project as model. Parameters can be set in the user panel.

The outputs of the scenarios can also be found at this link.

3.5.4 Feature Overview

In this section, we will discuss the features that are present in Repast/ReLogo. The resulting feature diagram can be seen in Figure 20.

Because ReLogo allows users to create agent specific methods, it is possible to separate agent behaviour from the agent's interface. Such a structure allows users to simulate autonomous behaviour. An agent can have a public method which is called by the UserObserver, which simulates the agent's perception of its environment. Based on the information provided by the environment, agents can decide how to respond. However, this is not true autonomous behaviour. For true autonomy, we expect that agent itself can decide itself when to observe the environment and can only be contacted by external entities through messages

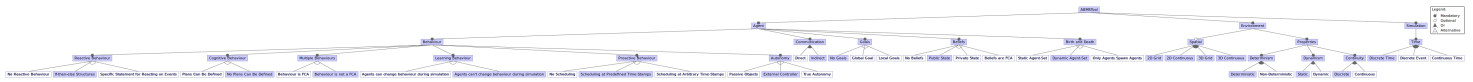

Figure 20: The feature model for Repast/ReLogo

or events. This is not the case since the observer still acts as a central point of control.

In Repast there is no specific way to define communication between agents. Again, we can simulate communication by invoking methods of other agents. However, no standard approach for such an implementation was found.

In Repast goals are not modeled explicitly.

As for the environment, Repast features a similar environment as NetLogo. Turtle agents have an absolute position, which allows a continuous spatial environment, as well as a patch on which they are situated, which allows for a grid based environment. Since patches are agents, the only responsibility of the environment is to keep track of the agent's position. This entails that the environment can not be dynamic or non-deterministic.

3.5.5 Repeatibility of Repast

Repast is a framework written in Java. Java is a deterministic programming language, so the expectation is that simulations are repeatable. Like NetLogo, Repast has built-in functionality for generating random numbers. It does provide a function, called randomseed(), where a user can set the seed for a simulation.The output traces that were generated by the simulation are the same as in NetLogo. When a car transitions to the next segment, he outputs the current ticks and his id. When a car finishes he outputs his id and travel-time in ticks.We ran each simulation in Section 3.1 ten times with random seed 0 for 100 seconds, where 1 second is 1 tick. To make sure Repast does not cache any computations, we restarted the tool for each output trace. In the resulting output traces no differences where found. We can conclude that experiments are repeatable in Repast on trace level, even when making use of the random number generator.

3.6 SARL

SARL [71] calls itself an agent-based programming language. Agent-oriented programming is a programming paradigm that relates a lot to the object oriented paradigm. Here, agents are the central concept instead of objects. The difference between an agent and an object, is that agents are autonomous and their methods can not be invoked directly from an external source, while this is very common with objects [63]. To invoke a method in an agent a message or event should be sent to the agent, who will decide autonomously whether to execute the desired function or not.To run a SARL program, an multi-agent platform should be used. Such a platform implements a representation of the environment and defines the access-points from the agent to the environment as well as to other agents. The MAS platforms that can be used for SARL programs are the JANUS-platform and the TinyMAS platform. SARL is not designed to be a tool for agent-based modeling and simulation, rather it is a domain specific language for developing MAS applications on the Janus-platform.

3.6.1 The SARL Programming Language

In this section, we will discuss the agent-oriented primitives of SARL. For this, we base ourselves on the documention provided on the SARL website [8]. SARL is also a fully object-oriented programming language, however, we will not focus on this here.

In SARL agents can communicate with each other by sending events. Events have a name, a type, a source, and optional data. Events can be created as follows:

```
event MyEvent {
    val string = "abcd"
    val number : Integer

    new(nb : Integer) {
        number = nb
    }
}
```

Here an event of type MyEvent is defined, which has two data fiels, string and number, of which string has a default value and number's value is set in a constructor method.

Another approach for communication between entities is messaging. Messages have the same properties as events, but additionally have a destination. SARL chose for the event option, because it is a more generic approach in which messages can be encoded.

A capacity is a specification of a collection of actions. It resembles closely an interface in object-oriented programming. Implementations of capacities are called skills. In an agent definition, developers can specify which capacities an agent can use. When an agent is created, it installs a set of skills. When a certain action of a capacity is called, the agent searches the associated skill and uses that implementation of the action.

The main concept in SARL is the agent. Like objects and classes, agents are instantiated according to an agent specification. Within the agent specification, developers can describe how agents react to certain events. This is done with an on statement. Let's look at following example:

```
agent MyAgent{
    uses Logging;

    on MyEvent {
        info("received an instance of MyEvent")
    }
}
```

In this example, whenever an agent of type MyAgent receives a MyEvent event. It will respond by printing the text "received an instance of MyEvent". Note that to print output, the agent makes use of the logging capacity, which is a built-in capacity.

On statements can be accompanied with a guard. In this case the reaction of the agent is only performed if the guard results to true. In following example, the agent will print "reaction 1" only if his number is smaller then 1. The agent will print "reaction 2", only when his number is higher than 40. In all other cases the agent will not react. Guards can overlap, so in some cases an agent can perform multiple reactions for the same event.

```
agent MyAgent{
    uses Logging;

    def number : Integer

    on MyEvent [number < 1]{
        info("reaction 1")
    }

    on MyEvent [number > 40]{
        info("reaction 2")
    }
}
```

In the two previous examples we defined the agent's behaviour in the agent specification. However, in SARL, behaviour is a first-class abstraction. This allows agent to have multiple behaviours and choose during runtime which behaviour is used.

Agents live in contexts, which in turn consist out of spaces. In the documentation [8], a context is defined as a boundary of a sub-system, and gathers a collection of spaces. Whereas a space is defined as the support of the interaction between agents respecting the rules defined in a Space Specification.

3.6.2 The JANUS MAS Platform

In order to run, SARL code needs to be run on a SARL runtime-environment. Such a runtime-environment executes or interprets a SARL program on a hardware platform. An overview of the compilation process can be seen in Figure 21.

Currently, there are two run-time environment available for SARL: Janus and TinyMAS. Since TinyMAS is not yet fully supported by SARL and development of this platform has stopped, we will focus on the Janus platform in this section.

Janus is a multi-agent platform that allows developers to easily create web, enterprise, and desktop multiagent-based applications [3]. It provides a comprehensive set of features that allow the development, deployment, and monitoring

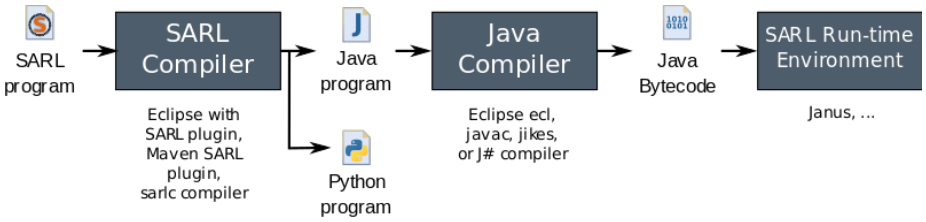

Figure 21: Compilation process of SARL

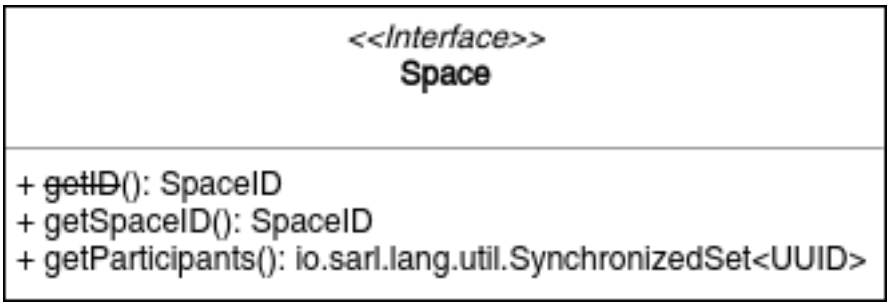

Figure 22: The Space interface

of these systems. Janus-based applications can also be distributed over a computer network.

3.6.3 The Example Traffic Model

SARL has a very poor support for environments. There is no concept of spatial environments, however it features a concept called spaces and sub-spaces, which act like a minimalistic environment. Agents are situated in a space and can also be situated in a sub-space within the space. A built-in type of space is an EventSpace. This is a space in which agents can send events to each other.

The user can however develop custom environment by extending from the Space interface, which can be seen in Figure 22. We implemented a Road space, which contains segments. Segments, in turn, contain a list of cars that are currently present on that space. The road also has a map of the *time_until_departure* of each segment

For the agents to interact with the road, we defined a capacity that defines the possible actions that a car can perform on the road. Figure 23, shows an UML diagram of the CarCapacity. We implemented the capacity in a skill. The CarSkill is initialized with a reference to the road on which the car is situated.

Next we defined a Car agent. This agent makes use of the CarCapacity to interact with the road. At each segment the agent determines whether their is enough time to observe the next section. If there is not enough time he schedules a depart action at the appropriate time and sets his time until departure for that section. Otherwise, he schedules an adjust speed action and notifies the road

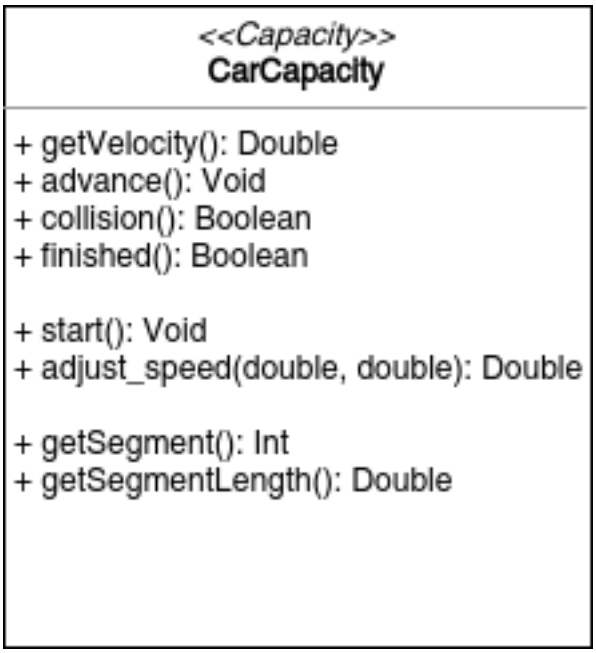

Figure 23: The car capacity

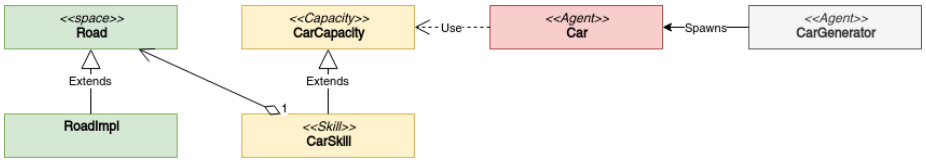

Figure 24: UML diagram of the system

with his time until departure. When the adjust speed action is executed, the car adjusts his speed according to the time until departure of the next segment. It then updates his time until departure and schedules a depart event.

Finally we create a CarGenerator agent. In SARL only agents can spawn other agents. The CarGeneretor's responsibility is to spawn new Car agents. To schedule the creation of the next car, it samples a random number generator that returns values between IAT_min and IAT_max. A UML diagram of the complete system can be seen in Figure 24.

3.6.4 Running the Application

To run the application, we used SARL version 0.9.0, which can be downloaded at this link. The project can be downloaded here. The project can then be imported in the SARL IDE and running the project as SARL agent and selecting the generator agent.

3.6.5 Feature Overview

In this section, we will discuss the features analysis of SARL. The diagram can be seen in Figure 25.

We first noticed that SARL has a very poor support for environments. There is no built-in way to define an environment. The closest alternative is spaces. Spaces are used to define how agents can interact with each other. The only built-in implementation of spaces that we found were simple event spaces. In these spaces, agents can send events either to a specific agent or broadcast them to all agents in the same space. In these environments, we could not specify dynamic or non deterministic behaviour of the environment.

As opposed to their support for environments, SARL provides a really good support for agents and agent specific concepts. We already discussed that agents can communicate by sending events either to a single receiver or by sending it into the environment. SARL also provides a specific statement to define reactive behaviour. SARL also provides a build-in capacity for scheduling actions at specific time stamps.

Agents can not influence other agents behaviour directly. Interaction is only possible with communication. This makes SARL agents truly autonomous.

Behaviour is a first class abstractions. Users can define multiple behaviours regardless of which agent will adopt it. Agents can then install a behaviour and change the installed behaviour when there is a need for it.

3.6.6 Adding M&S Functionality to SARL

The Janus platform is a MAS platform and features no concept of virtual time. We therefor did not investigate its repeatibility. Instead, we investigated the possibilities to include modeling and simulation to the SARL programming language.

Adding a Scheduling Agent The first possibility is to define a scheduling agent. All agents should schedule their actions with the scheduling agents which will keep track of a virtual clock and instruct agents to perform certain actions according to the schedule.

This approach is very similar to the observer agent in NetLogo and Repast. Here, the observer is also responsible for directing agent behaviour and increases the tick count.

The approach of creating a modeling and simulation tool from a MAS platform is not a novel approach. The TurtleKit platform [58] is a simulation platform based on the MadKit platform [43].

The advantages of such an approach is that, by explicitly modeling the scheduler, developers have a high degree of control over the simulation. The disadvantage is that this complicates the model drastically. Also, agent-based modeling is used to model individual behaviour. By including a scheduling agent, the representation of the real system becomes less clear, because the

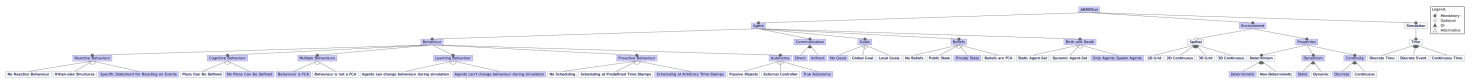

Figure 25: The feature model for SARL

behaviour has to take into account that he needs to operate with a scheduling agent.

Simulation Platform Another approach that we propose is a simulation SRE. Instead of changing the original SARL program, the same program can be deployed on either the simulation runtime-environment and the Janus platform. This approach is similar to MAS development tools such as MACE [49], which feature simulation tools for developing MAS applications.

4 Related Work

Railsback et al. reviewed five software platforms for scientific agent-based models [70]. The platforms under study were NetLogo, Repast, Swarm (Objective-C), Swarm (Java), and MASON. In each platform, they modeled sixteen versions of a model, each with increasing complexity. The primary goal of there research is to provide guidance to possible platform users and to give recommendations for which tool to use in which situation. There research focuses more on the practicality of the tools and less on the theoretical concepts of the ABM paradigm. In our study, we will aim to analyze how these tools adhere to the theoretical concepts of ABM.

In [9], a list is provided of almost the entire spectrum of agent-based modelling and simulation tools. This list clearly shows the proliferation in ABM tools. They also provide a guide for engineers, researchers, learners and academicians for selecting the most appropriate tool for their needs.

In [54], C. M. Macal gives a good introductory explanation of agent-based modeling. In this paper, he describes what agent based models are, gives a set of definitions, and discusses the different fields in which ABM is used. Another interesting paper that was co-authored by C.M. Macal is [55]. In this paper, the concepts of autonomy, agents, agent interaction, and environment is explained in more detail.

In [59], the history of agent-based modeling is discussed. They also discuss in great detail the need for modeling and simulation for multi-agent systems and how modeling and simulation can be used for multi-agent systems.

To get a better understanding about agents and multi-agent system, we recommend this paper [46]. This paper explains in detail the history of agents and multi-agents systems. They also discuss the wide variety of application of multi-agent systems.

In [80], A. Uhrmacher describes how agents can be implemented in discrete event simulations. She also discusses the James platform, a discrete event agent-based simulation platform. In our study we used a similar approach when modeling the entity centric DEVS model.

5 Conclusion and Future Work

In the first part of the document, we investigated the literature about agent-based modeling and multi-agent systems. We reviewed a set of definitions for multi-agent systems and concluded that they are very similar. This suggest that the concept of a multi-agent system is well established. On the other hand, we observed that this is not the case with agent-based modeling. The different definitions of C. M. Macal suggest that different interpretations of agent-based modeling exist. We also investigated how the concept of agent is defined. We saw that definitions from both the agent-based modeling perspective as well as the multi-agent perspective. However, if we take a closer look at agent-based modeling, we noticed that certain properties of agents are less strict.

In the second part of the document we investigated a set of tools that implement the agent-based modeling paradigm. We analyzed the features that each of the tools has implemented by creating a feature diagram of agent-based related features and instantiating the model for each of the tools. We also investigated the repeatability of the tools that feature a simulation platform. Due to the diversity in the tools we studied, it is hard to draw a single conclusion. We conclude that NetLogo and Repast are very similar tools. They are also the two tools that are specifically for agent-based modeling and simulation. However, we found that they have a very poor support for modeling behaviour. Though we established that the lack of autonomy is justified due to the single threaded execution of the models, we would have liked to see better support for implementing reactive and cognitive behaviour, like we saw in SARL. Resource Centric DEVS doesn't seem like a good fit for agent-based modeling. The fact that agent behaviour emerges from the behaviour of the resources is kind of the opposite approach to agent-based modeling.

For future work, we aim to unify the field of agent-based modeling. We will investigate the needs of the different fields that use the approach and define a formal semantics for agent-based modeling. Then, we will develop a formalism based on this formal semantics and show the relation between the formalism and existing tools.

References

[1] Charles m. macal biography. https://www.anl.gov/profile/charles-m-macal.

[2] Eclipse. `https://www.eclipse.org/`. Accessed: 16-August-2019.

[3] Janus. `http://www.janusproject.io/`. Accessed: 17-August-2019.

[4] Michael woolridge summary. http://www.cs.ox.ac.uk/people/michael.wooldridge/.

[5] Netlogo documentation. `https://ccl.northwestern.edu/netlogo/docs/programming.html`. Accessed: 15-August-2019.

[6] Nick jennings summary. https://www.imperial.ac.uk/people/n.jennings.

[7] Relogo primitives. `https://repast.github.io/docs/api/repast_simphony/ReLogoPrimitives.html`. Accessed: 16-August-2019.

[8] Sarl documentation. `http://www.sarl.io/docs/official/reference/Event.html`. Accessed: 17-August-2019.

[9] Sameera Abar, Georgios K Theodoropoulos, Pierre Lemarinier, and Gregory MP O'Hare. Agent based modelling and simulation tools: a review of the state-of-art software. *Computer Science Review*, 24:13–33, 2017. Amount of References: 44.

[10] Gul Agha. Concurrent programming using actors. *Object-oriented concurrent programming*, pages 37–53, 1987.

[11] Gul Agha, Peter Wegner, and Akinori Yonezawa. *Research directions in concurrent object-oriented programming*. Mit Press, 1993.

[12] Gul A Agha and Wooyoung Kim. Actors: A unifying model for parallel and distributed computing. *Journal of systems architecture*, 45(15):1263–1277, 1999.

[13] Fernando J Barros. Dynamic structure discrete event system specification: a new formalism for dynamic structure modeling and simulation. In *Winter Simulation Conference Proceedings, 1995.*, pages 781–785. IEEE, 1995.

[14] Maurice Stevenson Bartlett. Stochastic population models; in ecology and epidemiology. Technical report, 1960.

[15] Fabio Luigi Bellifemine, Giovanni Caire, and Dominic Greenwood. *Developing multi-agent systems with JADE*, volume 7. John Wiley & Sons, 2007.

[16] Carole Bernon, Marie-Pierre Gleizes, Sylvain Peyruqueou, and Gauthier Picard. Adelfe: a methodology for adaptive multi-agent systems engineering. In *International Workshop on Engineering Societies in the Agents World*, pages 156–169. Springer, 2002.

[17] Paul-Antoine Bisgambiglia, Paul Antoine Bisgambiglia, and Romain Franceschini. Agent-oriented approach based on discrete event systems.

[18] Eric Bonabeau. Agent-based modeling: Methods and techniques for simulating human systems. *Proceedings of the National Academy of Sciences*, 99(suppl 3):7280–7287, 2002.

[19] Alan H Bond and Les Gasser. *Readings in distributed artificial intelligence*. Morgan Kaufmann, 2014.

[20] Grady Booch. *Object oriented analysis & design with application*. Pearson Education India, 2006.

[21] Michael Bratman. *Intention, plans, and practical reason*, volume 10. Harvard University Press Cambridge, MA, 1987.

[22] Rodney Brooks. A robust layered control system for a mobile robot. *IEEE journal on robotics and automation*, 2(1):14–23, 1986.

[23] Bruce G Buchanan and Tom M Mitchell. Model-directed learning of production rules. In *Pattern-directed inference systems*, pages 297–312. Elsevier, 1978.

[24] Stephanie Cammarata, David McArthur, and Randall Steeb. Strategies of cooperation in distributed problem solving. In *Readings in Distributed Artificial Intelligence*, pages 102–105. Elsevier, 1988.

[25] Davy Capera, J-P Georgé, M-P Gleizes, and Pierre Glize. The amas theory for complex problem solving based on self-organizing cooperative agents. In *WET ICE 2003. Proceedings. Twelfth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, 2003.*, pages 383–388. IEEE, 2003.

[26] Wai Kin Victor Chan, Young-Jun Son, and Charles M Macal. Agent-based simulation tutorial-simulation of emergent behavior and differences between agent-based simulation and discrete-event simulation. In *Simulation Conference (WSC), Proceedings of the 2010 Winter*, pages 135–150. IEEE, 2010. Amount of References: 112.

[27] Adam Cheyer and David Martin. The open agent architecture. *Autonomous Agents and Multi-Agent Systems*, 4(1):143–148, 2001.

[28] Nick Collier. Repast: An extensible framework for agent simulation. *The University of Chicago's Social Science Research*, 36:2003, 2003. Amount of References: 305.

[29] Daniel D Corkill and Victor R Lesser. The use of meta-level control for coordination in a distributed problem solving network. Technical report, MASSACHUSETTS UNIV AMHERST DEPT OF COMPUTER AND INFORMATION SCIENCE, 1983.

[30] Massimo Cossentino. From requirements to code with passi methodology. In *Agent-oriented methodologies*, pages 79–106. IGI Global, 2005.

[31] Robert G Coyle. System dynamics modelling: a practical approach. *Journal of the Operational Research Society*, 48(5):544–544, 1997.

[32] A. Dorri, S. S. Kanhere, and R. Jurdak. Multi-agent systems: A survey. *IEEE Access*, 6:28573–28593, 2018. Amount of References: 0.

[33] Alexis Drogoul, Diane Vanbergue, and Thomas Meurisse. Multi-agent based simulation: Where are the agents? In *International Workshop on Multi-Agent Systems and Agent-Based Simulation*, pages 1–15. Springer, 2002. Amount of References: 330.

[34] Edmund H Durfee. *Coordination of distributed problem solvers*, volume 55. Springer Science & Business Media, 2012.

[35] Oren Etzioni and Daniel S Weld. Intelligent agents on the internet: Fact, fiction, and forecast. *IEEE expert*, 10(4):44–49, 1995.

[36] Jacques Ferber and Jean-Pierre Müller. Influences and reaction: a model of situated multiagent systems. In *Proceedings of Second International Conference on Multi-Agent Systems (ICMAS-96)*, pages 72–79, 1996.

[37] Jacques Ferber and Gerhard Weiss. *Multi-agent systems: an introduction to distributed artificial intelligence*, volume 1. Addison-Wesley Reading, 1999. Amount of References: 3893.

[38] Les Gasser and Kelvin Kakugawa. Mace3j: fast flexible distributed simulation of large, large-grain multi-agent systems. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 2*, pages 745–752. ACM, 2002.

[39] Michael R Genesereth and Nils J Nilsson. Logical foundations of. *Artificial Intelligence. New York: Morgan Kaufmann Publishers*, 1987.

[40] Michael Georgeff, Barney Pell, Martha Pollack, Milind Tambe, and Michael Wooldridge. The belief-desire-intention model of agency. In Jörg P. Müller, Anand S. Rao, and Munindar P. Singh, editors, *Intelligent Agents V: Agents Theories, Architectures, and Languages*, pages 1–10, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg. Amount of References: 719.

[41] Rhys Goldstein and Azam Khan. A taxonomy of event time representations. In *Proceedings of the Symposium on Theory of Modeling & Simulation*, page 6. Society for Computer Simulation International, 2017.

[42] Cláudio Gomes, Joachim Denil, and Hans Vangheluwe. Causal-block diagrams. 2016.

[43] Olivier Gutknecht and Jacques Ferber. Madkit: a generic multi-agent platform. In *Proceedings of the fourth international conference on Autonomous agents*, pages 78–79. ACM, 2000.

[44] Aaron Helsinger, Michael Thome, and Todd Wright. Cougaar: a scalable, distributed multi-agent architecture. In *2004 IEEE International Conference on Systems, Man and Cybernetics (IEEE Cat. No. 04CH37583)*, volume 2, pages 1910–1917. IEEE, 2004.

[45] Nicholas R Jennings. An agent-based approach for building complex software systems. *Communications of the ACM*, 44(4):35–41, 2001.

[46] Nicholas R Jennings, Katia Sycara, and Michael Wooldridge. A roadmap of agent research and development. *Autonomous agents and multi-agent systems*, 1(1):7–38, 1998.

[47] Nick Jennings and Michael Woolridge. Software agents. *IEE Review*, pages 17–20, 1996.

[48] Dennis Kafura and Jean-Pierre Briot. Actors and agents. *IEEE Concurrency*, (2):24–28, 1998.

[49] Charles Edwin Killian, James W Anderson, Ryan Braud, Ranjit Jhala, and Amin M Vahdat. Mace: language support for building distributed systems. In *ACM SIGPLAN Notices*, volume 42, pages 179–188. ACM, 2007.

[50] Barbara Kitchenham. Procedures for performing systematic reviews. *Keele, UK, Keele University*, 33(2004):1–26, 2004.

[51] David Kotz and Robert S Gray. Mobile agents and the future of the internet. *ACM Operating Systems Review*, 1999.

[52] Paulo Leitao, Stamatis Karnouskos, Luis Ribeiro, Jay Lee, Thomas Strasser, and Armando W Colombo. Smart agents in industrial cyber–physical systems. *Proceedings of the IEEE*, 104(5):1086–1101, 2016.

[53] Jing Lin, Sahra Sedigh, and Ann Miller. Modeling cyber-physical systems with semantic agents. In *Computer Software and Applications Conference Workshops (COMPSACW), 2010 IEEE 34th Annual*, pages 13–18. IEEE, 2010. Amount of References: 66.

[54] Charles M Macal. Everything you need to know about agent-based modelling and simulation. *Journal of Simulation*, 10(2):144–156, 2016. Amount of References: 70.

[55] Charles M Macal and Michael J North. Tutorial on agent-based modelling and simulation. *Journal of simulation*, 4(3):151–162, 2010. Amount of References: 1235.

[56] P Maes. [a]agents that reduce work and information overload, [o] comm, 1994.

[57] Fabien Michel. The irm4s model: the influence/reaction principle for multiagent based simulation. In *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, page 133. ACM, 2007. Amount of References: 63.

[58] Fabien Michel, Grégory Beurier, and Jacques Ferber. The turtlekit simulation platform: Application to complex systems. In *SITIS: Signal-Image Technology and Internet–Based Systems*, 2005.

[59] Fabien Michel, Jacques Ferber, and Alexis Drogoul. Multi-agent systems and simulation: A survey from the agent commu-nity's perspective. In *Multi-Agent Systems*, pages 17–66. CRC Press, 2018.

[60] Tom M Mitchell. Machine learning, 1997.

[61] Nicola Muscettola, P Pandurang Nayak, Barney Pell, and Brian C Williams. Remote agent: To boldly go where no ai system has gone before. *Artificial intelligence*, 103(1-2):5–47, 1998.

[62] Nils J Nilsson. Shakey the robot. Technical report, SRI INTERNATIONAL MENLO PARK CA, 1984.

[63] James Odell. Objects and agents compared. *Journal of object technology*, 1(1):41–53, 2002.

[64] James J Odell, H Van Dyke Parunak, Mitch Fleischer, and Sven Brueckner. Modeling agents and their environment. In *International Workshop on Agent-Oriented Software Engineering*, pages 16–31. Springer, 2002. Amount of References: 187.

[65] Guy H Orcutt. A new type of socio-economic system. *The review of economics and statistics*, pages 116–123, 1957.

[66] Jonathan Ozik, Nicholson T Collier, John T Murphy, and Michael J North. The relogo agent-based modeling language. In *2013 Winter Simulations Conference (WSC)*, pages 1560–1568. IEEE, 2013.

[67] Praveen Paruchuri, Alok Reddy Pullalarevu, and Kamalakar Karlapalem. Multi agent simulation of unorganized traffic. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 1*, pages 176–183. ACM, 2002.

[68] Gauthier Picard and Marie-Pierre Gleizes. The adelfe methodology. In *Methodologies and Software Engineering for Agent Systems*, pages 157–175. Springer, 2004.

[69] Mark Pinsky and Samuel Karlin. *An introduction to stochastic modeling*. Academic press, 2010.

[70] Steven F Railsback, Steven L Lytinen, and Stephen K Jackson. Agent-based simulation platforms: Review and development recommendations. *Simulation*, 82(9):609–623, 2006. Amount of References: 793.

[71] Sebastian Rodriguez, Nicolas Gaud, and Stéphane Galland. Sarl: a general-purpose agent-oriented programming language. In *2014 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT)*, volume 3, pages 103–110. IEEE, 2014.

[72] Stanley J Rosenschein. Formal theories of knowledge in ai and robotics. *New generation computing*, 3(4):345–357, 1985.

[73] Stuart J Russell and Peter Norvig. *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,, 2016.

[74] Douglas C Schmidt. Model-driven engineering. *COMPUTER-IEEE COMPUTER SOCIETY-*, 39(2):25, 2006.

[75] Yoav Shoham. Agent-oriented programming. *Artificial intelligence*, 60(1):51–92, 1993.

[76] Burrhus Frederic Skinner. *Science and human behavior*. Number 92904. Simon and Schuster, 1953.

[77] Peter Stone and Manuela Veloso. Multiagent systems: A survey from a machine learning perspective. *Autonomous Robots*, 8(3):345–383, 2000.

[78] Seth Tisue and Uri Wilensky. Netlogo: A simple environment for modeling complexity. In *International conference on complex systems*, volume 21, pages 16–21. Boston, MA, 2004. Amount of References: 566.

[79] Seth Tisue and Uri Wilensky. Netlogo: Design and implementation of a multi-agent modeling environment. In *Proceedings of agent*, volume 2004, pages 7–9, 2004. Amount of References: 265.

[80] Adelinde M Uhrmacher and Bernd Schattenberg. Agents in discrete event simulation. In *European Simulation Symposium-ESS*, volume 98, pages 129–136, 1998. Amount of References: 55.

[81] Adelinde M Uhrmacher and Danny Weyns. *Multi-Agent systems: Simulation and applications*. CRC press, 2009.

[82] Yentl Van Tendeloo and Hans Vangheluwe. An overview of python-pdevs. *JDF 2016-Les Journées DEVS Francophones-Théorie et Applications*, pages 59–66, 2016.

[83] Yentl Van Tendeloo and Hans Vangheluwe. An introduction to classic devs. *arXiv preprint arXiv:1701.07697*, 2017.

[84] Hans Vangheluwe et al. Devs as a common denominator for multi-formalism hybrid systems modelling. In *IEEE international symposium on computer-aided control system design*, volume 134. IEEE, 2000.

[85] José M Vidal, Paul A Buhler, and Michael N Huhns. Inside an agent. *IEEE Internet Computing*, 5(1):82–86, 2001. Amount of References: 69.

[86] Vito Volterra. Fluctuations in the abundance of a species considered mathematically, 1926.

[87] Shiyong Wang, Jiafu Wan, Daqiang Zhang, Di Li, and Chunhua Zhang. Towards smart factory for industry 4.0: a self-organized multi-agent system with big data based feedback and coordination. *Computer Networks*, 101:158–168, 2016.

[88] Gerhard Weiss. *Multiagent systems: a modern approach to distributed artificial intelligence*. MIT press, 1999. Amount of References: 5368.

[89] Michael Wooldridge. *An introduction to multiagent systems*. John Wiley & Sons, 2009.

[90] Michael Wooldridgey and Paolo Ciancarini. Agent-oriented software engineering: The state of the art. In *International Workshop on Agent-Oriented Software Engineering*, pages 1–28. Springer, 2000.

A Non-Repeatable NetLogo Example

```
breed [tests test]

tests-own[output]

to setup
  clear-all
  reset-ticks
  random-seed 0
  let index 1
  create-tests 4 [
    set output index
    set index (index + 1)
  ]
end

to go
   every 1 [ ask turtles [
    print(output)
  ]
  print("--")]
  tick
end
```

run 1	[1,4,2,3]	[3,2,4,1]	[1,2,4,3]	[3,4,1,2]	
run 2	[1,4,2,3]	[3,2,4,1]	[1,2,4,3]	[3,4,1,2]	
run 3	[1,4,2,3]	[3,2,4,1]	[1,2,4,3]	[3,4,1,2]	
run 4	[1,4,2,3]	[3,2,4,1]	[1,2,4,3]	[3,4,1,2]	
run 5	[1,4,2,3]	[3,2,4,1]	[1,2,4,3]	[3,4,1,2]	
run 6	[1,4,2,3]	[3,2,4,1]	[1,2,4,3]	[3,4,1,2]	[1,2,4,3]
run 7	[1,4,2,3]	[3,2,4,1]	[1,2,4,3]	[3,4,1,2]	[1,2,4,3]
run 8	[1,4,2,3]	[3,2,4,1]	[1,2,4,3]	[3,4,1,2]	[1,2,4,3]
run 9	[1,4,2,3]	[3,2,4,1]	[1,2,4,3]	[3,4,1,2]	
run 10	[1,4,2,3]	[3,2,4,1]	[1,2,4,3]	[3,4,1,2]	[1,2,4,3]

Table 4: Output of the NetLogo example at normal speed on a single machine