THE UNIVERSITY
of EDINBURGH

Stochastic Modelling of Spatial Collective Adaptive Systems

Natalia Zoń

Doctor of Philosophy

Laboratory for Foundations of Computer Science
School of Informatics
University of Edinburgh

2019

Abstract

Collective Adaptive Systems (CAS) are composed of individual agents with internal knowledge and rules which organize themselves into ensembles. These ensembles can often be observed to exhibit behaviour resembling that of a single entity with a clear goal and a consistent internal knowledge, even when the individual agents within the ensemble are not managed by any outside, globally-accessible entity.

Because of their lack of a need for centralized control which results in high robustness, CAS are commonly observed in nature – and for similar reasons are often reflected in human engineered systems. Researching the patterns of operation observed in such systems provides meaningful insight into how to design and optimise stable multi-agent systems capable of withstanding adverse conditions. Formal modelling provides valuable intellectual tools which can be applied to the problem of analysis of systems by means of modelling and simulation.

In this thesis we explore the modelling of CAS in which space (topology and distances) plays a significant role. Working with CARMA (Collective Adaptive Resource-sharing Markovian Agents) a formal feature-rich language for modelling stochastic CAS, we investigate a number of spatial CAS scenarios from the realm of urban planning. When components operate in a spatial context, their behaviour can be affected by where they are located in that space. For example, their location can influence the speed at which they move, and their ability to communicate with other components.

Components in CARMA have internal store, and behaviour expressed by Markov processes. They can communicate with each other through sending messages on state transitions in a unicast or broadcast fashion. Simulation with pseudo-random events can be used to obtain values of measures applied to CARMA models, providing a basis for analysis and optimisation.

The CARMA models developed in the case studies are data-driven and the results of simulating these models are compared with real-world data. In particular, we explore two scenarios: crowd-routing and city transportation systems.

Building on top of CARMA, we also introduce CGP (CARMA Graphical Plugin), a novel graphical software tool for graphically specifying spatial CAS systems with the feature of automatic translation into CARMA models. We also supply CARMA with additional syntax structures for expressing spatial constructs.

Lay Summary

Collective Adaptive Systems (CAS) are arrangements in which agents interact with each other in the context of a common environment. These agents are individual entities possessing internal knowledge (either innate or obtained from observation), as well as behaviour, which determines the actions they take. They can share their knowledge with other agents through various means of communication.

Systems like these are called "collective" because they are often analyzed from the perspective of the whole group of agents acting together, rather than each agent individually.

Examples from nature include the flocking of birds, fish swimming in formations ("schools") or ants self-managing the operation of a colony. The common feature of these ensembles is the fact that the involved agents (individual organisms) are to a large extent homogeneous and have very similar innate behaviour rules. However, when working in the context of an environment, surrounded by other agents, through means of communication and observation, they can adjust their own behaviour so that each agent plays a slightly different, yet significant role in the collective.

CARMA is a formal language designed for expressing models of CAS. It provides ways to describe agents (in CARMA they are called "components"), their knowledge and behaviour. It also allows for the specification of the environment in which these agents operate. CARMA is supplied with a set of tools which are designed to make the modelling of CAS easier, as well providing the ability to obtain values of measures applied to the system, using simulation.

The main focus of the work described in this thesis is CAS systems in which physical space plays an important role.

We extended the syntax of CARMA with constructs tailored for specifying systems in which the spatial locations of components and their movement through space are essential factors to the overall performance of the system.

We present the CARMA Graphical Plugin (CGP), an extension of the CARMA Plugin, supplying the user with the ability to define the model graphically and to automatically generate CARMA code from the graphical representation.

The first CARMA model we describe is one of mesoscopic (looking at the system from both individual-agents and collectives perspective) crowd-routing. This is to analyze

how pedestrian movement through a network of paths affects, and is affected by, congestion. We provide a case study based on The Meadows city park in Edinburgh.

The second model captures a scenario based on an urban transportation system. We worked with the Transport For Edinburgh company to obtain real data gathered from city buses in Edinburgh. The simulations run with our model are in a good agreement with the data.

Acknowledgements

My research would have been impossible without the aid and support of my supervisor, Stephen Gilmore. Thank you for all the time and effort you put into guiding me throughout my time as your student. I have been extremely lucky to have a supervisor who cared so much about my work, and who responded to my questions so promptly and earnestly.

My sincere thanks to Michele Loreti who was the source of invaluable help and advice throughout my PhD, and without whose guidance and support none of the work done on software implementation would have been completed.

Many thanks to Vashti Galpin and Pia Wilsdorf for their invaluable input and wonderful collaboration.

I would also like to thank my second supervisor, Jacques Fleuriot.

I acknowledge the contribution of Jane Hillston and other collaborators I met through working on the QUANTICOL project.

I am grateful to the QUANTICOL project for the scholarship which allowed me to pursue this degree and to LFCS which provided financial support after the completion of the project.

My PhD would have not been the same without the friends I met in the Informatics Forum. Thank you all for the times we shared laughter and for the times we shared tears, for 2am chip-shop excursions while working all-night on a paper, for talking each other out of quitting the PhD, and most of all for creating what felt like a family in a place so geographically distant from from where our respective families lived. I would not have made it without you.

Doing a PhD involves going through a lot of stress and psychological pressure while having very little money. I would like to express my deep gratitude to the The University of Edinburgh Counselling Service and the Arkordia therapy cooperative for providing free donation-based counselling sessions.

I would like to thank my parents for consistently being a source of boundless support and encouragement, and for always firmly believing that I would succeed in anything I do. Your never-ceasing faith in me gives me the confidence to keep going.

Last but not least, I am grateful to my loving and supportive partner Ricardo, who I met during the first year of my PhD. Thank you for being the pillar of stability even when the whole world was engulfed in chaos. You were there for me every step of the way, and I wholeheartedly appreciate everything you have done for me.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

- The material presented in Chapter 3 was published in [48],
- The material presented in Chapter 4 was published in [114],
- The material presented in Chapter 5 was published in [112] and [50],
- The material presented in Chapter 7 was published in [50],
- The material presented in Chapter 8 has been accepted for publication in [113].

(*Natalia Zoń*)

Table of Contents

Chapter 1

Introduction

> What we observe is not nature itself, but nature exposed to our method of questioning.
>
> *Werner Heisenberg*

1.1 Problem statement and scientific contribution

This thesis talks about Collective Adaptive Systems (CAS) and how our current understanding of their inner workings, combined with existing and novel modelling tools designed for working with those systems, has the potential to contribute towards creating better solutions in the areas of urban planning and transportation.

We will mostly consider the stability, scalability, reliability and optimization aspects of the systems, as we seek to provide modelling tools which allow designers to predict the behaviour and emergent properties of CAS.

The scientific contribution of this thesis consists of the creation of tools aiding the analysis of the properties of CAS in which space plays an important role, as well as utilizing real data in the models created using these novel tools together with pre-existing ones, in order to demonstrate their applicability to solving real world problems from the area of CAS modelling. Specifically, the following topics are included in this thesis:

- the design of a spatial-modelling extension of the CASL language which makes

it more suitable for describing CAS with important spatial aspects (described in detail in Chapter 3),

- the conception and implementation of Carma Graphical Plugin (CGP), a novel software tool which is tailor-made for the purpose of graphically specifying CAS with spatial aspects (described in detail in Chapter 4)
- the conception of two models from the area of urban planning: crowd routing through a network of paths (described in detail in Chapters 5 and 7), and bus transportation system (described in detail in Chapter 8), together with the application of real world data to these models.

The significance of the scientific contribution of this thesis lies first and foremost in demonstrating how the theoretical and software-based tools for modelling of CAS may be applied to real world data in order to obtain meaningful results and thus showing the promising potential of combining highly theoretical modalities such as formal modelling with substantial concrete problems based on real-world data.

1.2 Structure of this thesis

This thesis is structured as follows:

- *Chapter 1: Introduction* characterizes the scientific problems the research described in this thesis engages in, and places them in the context of real world scenarios. It also provides descriptions of the various existing scientific approaches to solving these problems.
- *Chapter 2: Background* provides a detailed presentation of the scientific concepts and techniques which we used as the basis of our approach to modelling and analyzing the real world scenarios, supplied with a collection of examples from nature and socio-technical systems.
- *Chapter 3: Spatial* CASL*: Extending* CASL *with syntax for spatial models* introduces in detail the specification language for CARMA and describes the extension of this language which we designed for the purpose of representing models in which space plays an important role.
- *Chapter 4:* CARMA *Graphical Plugin* presents the software tool we designed

and implemented, building on top of the CARMA tool suite, together with a set of usage examples.

- *Chapter 5: Modelling of pedestrian movement and simple crowd routing* conveys the process and the results of the research undertaken in the subject of crowd routing using the CARMA language.
- *Chapter 6: Data* introduces real-world data sources which are used for the research described in the chapters which follow.
- *Chapter 7: Data-driven modelling of pedestrian movement* is a description of our data-driven approach to the problem of modelling the behaviour of crowds.
- *Chapter 8: Data-driven modelling of urban transportation systems* presents the process and the results of designing and analysing an urban transportation model created using data obtained from a real-world system.
- *Chapter 9: Conclusions* offers a summary of the work described in this thesis.

This thesis is about applying formal methods to real world data. This is not an easy problem to solve, and as such it requires a modular, systematic approach. The work presented in Chapter 8 was built on the fundaments laid down by the earlier chapters. We started from the very simple model of pedestrian routing presented in Chapter 5, which served mainly as a proof of feasibility. We then fused this purely speculative model with a number of uncomplicated and straightforward assumptions coming from real-world data (the geographical locations of network nodes and a simple traffic measure) in Chapter 7. Finally, in Chapter 8 we were able to construct a mature version of a spatial model in which an assortment of theoretical and software tools come together with multiple and rich data sources. Perhaps most interestingly, the measures applied to the simulation of the model can be directly compared with the qualities of the real-world system measured from data only – proving the assumption which directed our research from the very beginning: that the formal and software tools we used and extended indeed do work.

1.3 Collective Adaptive Systems

Collective Adaptive Systems (CAS) [66] is a term used to describe a wide range of systems that share the following characteristics:

- They consist of a large number of concurrently acting components (often called 'agents'), which are usually similar to a certain degree, or fall within a number of classes of similarity (however, they can also be highly heterogeneous), and can enter or leave the collective at any time (in this sense, such systems are 'open'). The agents have the ability to individually control their own behaviour based on their local knowledge, as opposed to a global knowledge of the whole system, which they usually do not possess.
- The system is capable of gradually changing its behavior to adapt to a new environment or to changes within the old environment. This does not necessarily mean that the agents themselves have evolution-like capabilities, but rather that the collective as a whole does.
- The structure, environment and properties of the system can often manifest as an observable *emergent behaviour*.

There are multiple definitions of CAS, as the term itself is relatively new. An alternative term with identical abbreviation, 'Complex Adaptive Systems', exists and tends to be used interchangeably with 'Collective Adaptive Systems'; or to specifically distinguish those systems which in addition to being collective and adaptive may also involve sophisticated hierarchy structures and rich dynamic networks of interactions between components.

Some sources [43] restrict the definition of CAS to only those collectives which are of a very large scale (with the number of component of the order of magnitude 10^9). Others [4] require them to have 'fractal-like' properties, that is being comprised of numerous layers of subsystems, the hierarchy of which can be very complex, fluid and even chaotic, as opposed to having a well defined, concrete structure.

For the purpose of this thesis, we consider CAS to be those systems which are comprised of a number of autonomous agents interacting within the same environment.

An important feature distinguishing CAS from many other systems, especially those from the arena of human engineering, is that the knowledge is always internal to the

agent (*local*), and may not at all times be correct or up-to-date.

This is reflective of many systems observed in nature as well as in human engineered arrangements. There are inherent limitations imposed on the agents within any real-world system, making a system where every agent has global and perfect knowledge difficult to implement in practice. These limitations, such as geographical distance, physical obstacles and signal noise, are unavoidable in real-world scenarios and must be taken into account when performing analysis and modelling of these systems.

Because of the fact that no agent can possess the knowledge about the whole system, in CAS there is often no global management or, in fact, a global goal that the collective would try to achieve. This is what distinguishes CAS from many human engineered systems (physical or software-based) in which all components are managed by a specialized part having a clear goal and access to all other components.

The examples of CAS in which global management is present to a certain degree, oftentimes come from the areas of socio-technical development. For example, the performance of a taxi system may largely depend on local decisions (the drivers picking up passengers from the locations they happen to be at), but in some cases can be improved by adding elements of an all-knowing global entity that influences these decisions (the taxi dispatch system). These managing entities are not always a required part of the system and may only be switched on when the overall performance drops below a certain threshold.

CAS tend to be very robust and highly flexible, and often well equipped for adapting to unexpected changes. This is because the agents comprising the ensemble are to a large degree homogeneous, which means that if an agent exits the collective at any time, it can be easily replaced by another agent.

The *emergent behaviour* is a phenomenon which can be observed at the level of the collective, where the whole ensemble seems to be acting as if trying to reach a particular goal under a global management. The behaviour that emerges during the evolution of the system comprised of a large number of agents is often complex and can be very difficult to predict just by analyzing an individual agent's behaviour – even despite the ensemble's homogeneity.

In this thesis we study the CAS having the following features specifically:

- large number of agents

- homogeneity of agents
- agents' decisions based on local knowledge
- agents acting and interacting within a common environment

The above characterise the model presented in later chapters of this thesis. Our interested has been focused on socio-technical examples of CAS such as pedestrian routing (see 5 and 7) and bus transportation systems (see 8).

1.4 Space in modelling

The process of creating models representing real phenomena consists mainly of abstracting that phenomena; filtering out all the details that either have none or a negligible amount of influence on the behaviour that we are interested in.

All systems that exist in the real-world are inherently spatial, as all the elements must exist somewhere in physical space. There are of course systems in which this geographical placement of elements is not very interesting, for example when analysing software architectures it is not usually important to know where the computers which are running the software are located. In some cases however, it is critically important – co-locating server machines closer to stock exchange buildings in order to minimise the request/response times is one of the tactics employed by high-frequency traders in order to outrun their rivals [99].

In the context of designing models of systems, the term 'space' has two separate meanings; one of them being the formerly mentioned physical space, while the other is a more high-level, conceptual understanding of it. When we lay out nodes and connections of a graph to represent a topology of a given system, we are, whether internally, in our minds, or externally by drawing it on a whiteboard, laying out these elements in a two-dimensional space. This way of thinking has remained an intuitive approach employed in order to untangle complex ideas, in which the mechanisms borrowed from the physical world: the spatial separation of elements, the connections leading from one to another, the size, the shapes – often prove great and universally understood representations of intricate systems.

This thesis is an exercise in exploring and capturing aspects of the spatial approach to modelling by means of creating software tools tailored for spatial models, as well

Figure 1.1: The crowds on the Royal Mile in Edinburgh during the Fringe festival[1].

as applying them on case studies with spatial features. Specifically, we investigate the spatially-motivated subclass of Collective Adaptive Systems (CAS) (described in detail in Chapter 2).

1.5 Crowd behaviour

Research into the behaviour of crowds has remained an active topic of scrutiny across a variety of scientific fields, and has been approached from many different angles [23, 56, 80].

The three examples of crowd-related case studies listed below, not only have very diverse motivations, but are also using distinct and largely unrelated tools and techniques for approaching the problem:

- The application of fluid physics for the purpose of crowd measurement or prediction [29].
- The simulation of a large multi-agent crowd with realistic behaviour to be rendered in 3D for special effects in movies and video games [96].
- The analysis of the influence of human psychology factors on the collective behaviour of a crowd [82].

The motivations for investigating crowd dynamics are numerous. Understanding the

[1] Photograph downloaded from `https://www.edinburghfestivalcity.com/news/480-numbers-soar-as-fringe-records-big-tickets-increase`.

mechanics of the collective movement of people within a crowd plays a critical role in designing evacuation schemes for buildings and other enclosed structures. In [78] a model of crowd evacuation is investigated in the context of large and geometrically complex buildings, taking into account the influence of crowd congestion on the speed of individuals. A model based on cellular automata is presented in [97], and the visibility range of agents is taken into account for the purpose of analysing its impact on overall evacuation speed. [108] offers yet another approach, in which the model is created by applying simple behavioural heuristics to each agent within a crowd; several emergent phenomena on the level of the collective are observed, such as spontaneous creation of one-way lanes and crowd turbulence at extreme densities.

The problem of evacuation however is not limited to contained spaces, and there are models representing emergency evacuation schemes for large crowds during religious mass gatherings, such as Hajj, an annual Islamic pilgrimage to Mecca [75] or crowd gatherings at Alopi Devi temple of Allahabad, India [101].

Widening scientific insight into how to model and predict the behaviour of crowds is therefore a worthwhile endeavour, and has the potential of directly contributing to saving human lives.

Evacuation is not the only context in which the dynamics of crowd movement play an important role. Congestion of pedestrian routes is a common problem in many large or tourism-oriented cities around the world, such as Prague [89] and Edinburgh [11] (see Figure 1.1). Developing and analysing models of these scenarios contributes towards improving the principles of designing urban pedestrian communication networks, bringing forth the creation of the 'walkable city' [92]. Densely populated cities can found their development policies on these principles in order to become better adapted to handling the flow of large crowds, which can improve the overall quality of the pedestrian experience. This in turn leads to increasing the number of people who choose to walk rather than drive, and has a positive impact on the environment, public health as well as levels of motorway traffic [51, 106].

In this thesis we investigate two case studies of crowd behaviour. In the first one, described in Chapter 5, we use formal modelling in order to analyse a network of paths traversed by two groups of pedestrians, travelling in the opposite directions. The second model, described in Chapter 7, is based on real-world data and regards the flow of traffic in the Meadows public park in Edinburgh.

Figure 1.2: Designated bus lane on Earl Grey Street in Edinburgh[2].

1.6 Urban Transportation Systems

Public transportation systems of different degrees and complexity are widely employed in cities around the world. Well-organised and efficient public transportation reduces traffic and the time spent commuting to work. In addition, more people choosing public transport rather than personal cars has a positive impact on reducing the number of vehicles on city roads: lessening their effect on climate change, improving air quality, and reducing noise pollution.

The planning of transportation systems is a broad area of research, requiring looking into a number of complex problems and having to balance a variety of contributing factors such as the design of transit networks, estimating costs and financing, energy and environmental impact issues and supporting innovative technologies [30].

The process of planning the transit networks, which we look at in Chapter 8 of this thesis, requires taking decisions regarding the available routes [26, 88], the locations of bus stops [41, 64], the creation of timetables [54, 55] and the development of the road infrastructure capable of prioritizing certain vehicles over others [115] (see Figure 1.2).

Broadening the understanding of the observed overall behaviour of the urban transportation system, subject to a number of variables, contributes towards creating new and better policies for designing such systems.

[2]Photograph downloaded from `https://www.maps.google.co.uk`.

One way of achieving this is by building abstract formal models that can be simulated. The results of these simulations may then be compared with real-data in order to assess to what degree the model is predictive of the observed behaviour.

The value of having accurate models lies in the fact that the input parameters can then be tuned in order to predict how a given system might behave under a certain set of conditions. These conditions may include phenomena such as: road closures, high traffic congestion or vehicle failures.

Modern urban transportation systems must be designed to have adaptive capabilities built-in because they need to respond to the unexpected events and circumstances which unfold as the delivery of the service progresses during the working day. This is particularly the case for bus services, where timetabled public transport must share the road network with private transport users who publish no timetable of their journeys and commuting plans and whose use of the road can depend on variables as diverse as the weather conditions, public holidays, and sporting events.

Set against this backdrop of hard-to-predict capacity availability of the underlying network, public transportation service providers must meet local or governmental requirements on quality-of-service as expressed through performance metrics such as percentiles of on-time departures or arrivals, excess waiting times, buses-per-hour requirements, and other measures [84]. In order to meet quantitative targets such as these, public transport systems must have both *local* (point-of-view) and *global* (locus-of-control) adaptability, allowing system stakeholders to make both micro-scale service decisions (such as bus drivers speeding up, slowing down, or waiting at bus stops) and macro-scale decisions (such as shift operators re-routing buses, cancelling service instances, or deploying additional buses to cope with an unexpected surge in demand).

Human decision making is both in-the-loop within these systems, typically making locally autonomous micro-scale decisions, and outside-the-loop, typically making global macro-scale control decisions. Seen in this way, public transport systems can be viewed as *collective adaptive systems*, where (sometimes unexpected) behaviour emerges from the local interactions between actors in the system who are sharing resources when collaborating to meet common goals, even as they may be sometimes competing over resources in their efforts to satisfy individual priorities.

Modern smart transport systems are data-rich, making informed macro-scale decision making possible. Each vehicle in the fleet is equipped with GPS receivers and com-

munications infrastructure to allow them to regularly report their location back to a vehicle tracking system. This *automatic vehicle location* (AVL) data provides anyone with access to the data with real-time oversight of the location of each vehicle in the fleet, making it possible to design applications which predict bus arrival times, and to compute metrics which provide statistical summaries of system performance in terms of key performance indicators which are of interest to system stakeholders.

In Chapter 8 we describe a data-derived stochastic model of bus transportation system in Edinburgh. The results of simulating the model are in good agreement with similar measures applied to real-data.

Chapter 2

Background

> Those who cannot remember the past are condemned to compute it.
>
> *Steven Pinker, Words and Rules: The Ingredients of Language*

2.1 The motivation behind the exploration of CAS

Collective Adaptive Systems (CAS), introduced in Section 3 of Chapter 1 of this thesis, are ensembles of agents acting and interacting within a common environment. The analysis of systems of this nature has the potential to yield results that might constitute a meaningful contribution to the scientific understanding of a number of research problems which are currently pursued.

In [66] the authors define the key concepts that play a role in the current exploration of CAS. These are:

- Purposeful artificial self-organization – the ability of the collective to act as a whole ('a superorganism'[109]), in order to achieve a common goal, which is accomplished by having each individual component following a set of predefined rules.
- Evolvability – the potential of the system to self-improve in terms of optimizing the values of its attributes and fine-tuning the existing rules, given that the mechanism for the evolutionary process itself is already provided within the environ-

ment (the application and assessment of fit-functions, inheritance of properties and rules, the occurrence of random mutations).

- Developmental plasticity – starting from a number of identical systems, the evolutionary process can take them into very divergent directions, making it possible to generate a wide range of distinctive systems where the variables responsible for these differences are outside the collectives themselves, and are part of the environments in which the systems have evolved. This property of a system, however, dramatically decreases its capacity for predicting or controlling the future behaviour.
- Stability, scalability, reliability, optimization – these aspects are being investigated and gradually improved from the technological and computational point of view in order to advance the current solutions for simulating CAS.

The ongoing exploration of CAS has also produced results which are already being applied in existing technology. Many urban solutions can be modelled as CAS, the list of examples includes systems such as bikesharing [95], bus-commute [105] or carpooling [111].

In this thesis we will mostly consider the last point, that is the stability, scalability, reliability and optimization of the systems, as we seek to provide modelling tools which allow designers to predict the behaviour and emergent properties of CAS. The focus of our research is on socio-technical examples of CAS. These are often systems in which the agents are autonomous to some degree but at the same time externally managed in order to ensure a certain goal is met. For example, in the pedestrian routing scenario presented in Chapters 5 and 7, the pedestrians are capable of deciding which direction to turn at a road fork, but they are at the same time externally influenced to keep to the left in order to avoid collisions.

2.1.1 Modelling CAS

Creating and investigating models of systems helps us to understand them better, as well as to evaluate and predict their performance. The word ‘model’ is a wide-ranging term that can be used to describe a number of similar but distinct ideas, and the diagram below (Fig. 2.1) shows the relation between them.

A real world phenomenon (Fig. 2.1 Panel A) is observed, and its conceptual model

(Fig. 2.1 Panel B) is created by specifying which elements are and which are not a part of the model, and which aspects of the observed systems are to be put under scrutiny. The conceptual model usually is not instantiated and is more of a collection of ideas in the mind of the observer.

The conceptual model can then be translated into a concrete representation, following a set of specific rules of a chosen modelling language (Fig. 2.1 Panel C). There are many possible modelling languages to choose from, each of which places its own limitations on the accuracy of representation of the original system. The choice of the language is therefore not an arbitrary one, as some of the languages may be more suitable for representing certain classes of models than others.

The model can then be translated into a runnable computer program (Fig. 2.1 Panel D), providing the possibility of simulating the performance or acquiring values of various measures of the system. In this case, the accuracy of representation becomes dependent on technical possibilities of the machine and underlying software, such as numerical accuracy, computational power, or semantics of the programming language used for implementation.

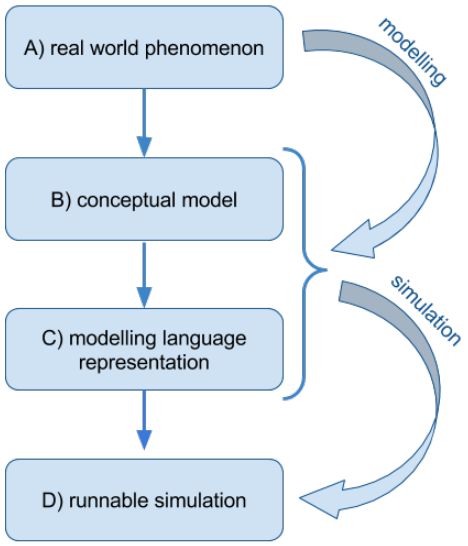


Figure 2.1: A hierarchical view of possible representations of a system as proposed by [110]. The term '*modelling*' usually refers to the creation of models which aim to represent real world systems (as well as models representing different kinds of models), while '*simulation*' refers to working with models and computers.

2.2 Examples of CAS

Examples of CAS can be found within a broad spectrum of problems which involve large numbers of individual agents interacting with each other in the context of a common environment.

2.2.1 Socio-technical systems

2.2.1.1 Overview

In urban scenarios, CAS are found in situations where there is a need for making decisions in real time to optimize the usage of a given service for a large number of people or devices. This subsection talks about such cases.

In the area of urban planning, bike-sharing systems, as well as carpooling may be modelled as CAS.

In [60] the authors present four classes of application domains for CAS that are currently being explored in terms of creating novel software and hardware specialized solutions:

- Power Management Systems,
- Cloud Computing,
- Telecommunication,
- Wearable Computational Devices.

Such scenarios, when implemented on the technological level, create the need for analytic tools that would provide insight into the inner workings and possible ways of optimization and further improvement of these systems.

Most of the CAS which belong to the ‘smart cities’ category, require some form of space (both physical and virtual) to be considered in the model. Concepts like distances, topologies of routes, one-way roads or zones often play a crucial role in urban scenarios.

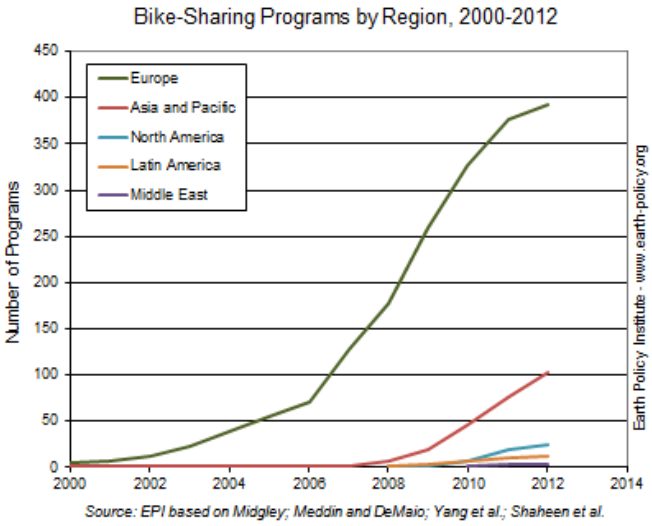


Figure 2.2: Bike sharing programs by regions of the world [70].

2.2.1.2 Bike-sharing systems

Bike-sharing systems are arrangements in which bicycles are made available (rented) for use, usually within urban areas. Most commonly, such services are paid, either per time of use or on a subscription basis, and the bicycles need to be picked up and returned at a number of specific stations located throughout the city [95].

Adopting a bike sharing system has the potential of bringing forth a number of positive effects on the city population. One of them is the reduction of traffic levels, noise and air pollution by lowering the number of personal cars travelling on roads as more people switch to using bicycles instead.

The first ever attempt at creating a bike-sharing system was undertaken in 1965, in Amsterdam, but quickly failed due to the bikes being stolen or vandalised [35]. Since then, a number of different approaches have been tested in different parts of the world, with varying levels of success. By the end of 2012, over 500 cities in 49 countries have employed bike-sharing programs [70] (see Fig. 2.2).

Bike-sharing schemes exhibit a lot of features congruent with the features of CAS [95], such as the locality of the user's knowledge, as well as the generally large number of users with homogeneous behaviour.

In the simple scenario, users concurrently interact with the system, having only local

[1]Photograph downloaded from `https://www.ed.ac.uk/files/styles/panel_breakpoints_theme_uoe_mobile_1x/public/thumbnails/image/img_0804.jpg`.

knowledge of its state. The bike stations where bicycles may be rented or returned have a limited capacity. The two most common issues arising in this kind of arrangement, from the perspective of the user, are:

- the nearest station(s) being empty when trying to rent out a bicycle,
- the nearest station(s) being full when trying to return a bicycle.

The global availability of such a system could be greatly improved by giving the users feedback on the current state of the system to influence their decisions. This can be realized, for example, by giving the users incentives to return their bike to an empty station, rather than to the closest station. Similarly, the performance of the system as a whole can be improved by encouraging people to choose full stations when deciding which station to rent a bicycle from [45].

2.2.1.3 Carpooling

Carpooling is a mode of transportation in which two or more commuters share one car in order to reach their destinations [111]. The first carpooling schemes date back to World War II, when the American government encouraged civilians to share rides as a means of rationing due to petrol shortages. At the present time, the interest in supporting carpooling systems stems mostly from their potential positive impact on the levels of traffic and environmental pollution [40, 77], but other positive effects, such as promoting the forming of social bonds between people living close to each other within large cities, have also been investigated [72].

In most carpooling scenarios there are two classes of users interacting within the system: drivers looking for passengers to share the cost of the ride with, and passengers looking for drivers with destinations located nearby their own.

Figure 2.3: Bike-sharing station in the city of Edinburgh[1].

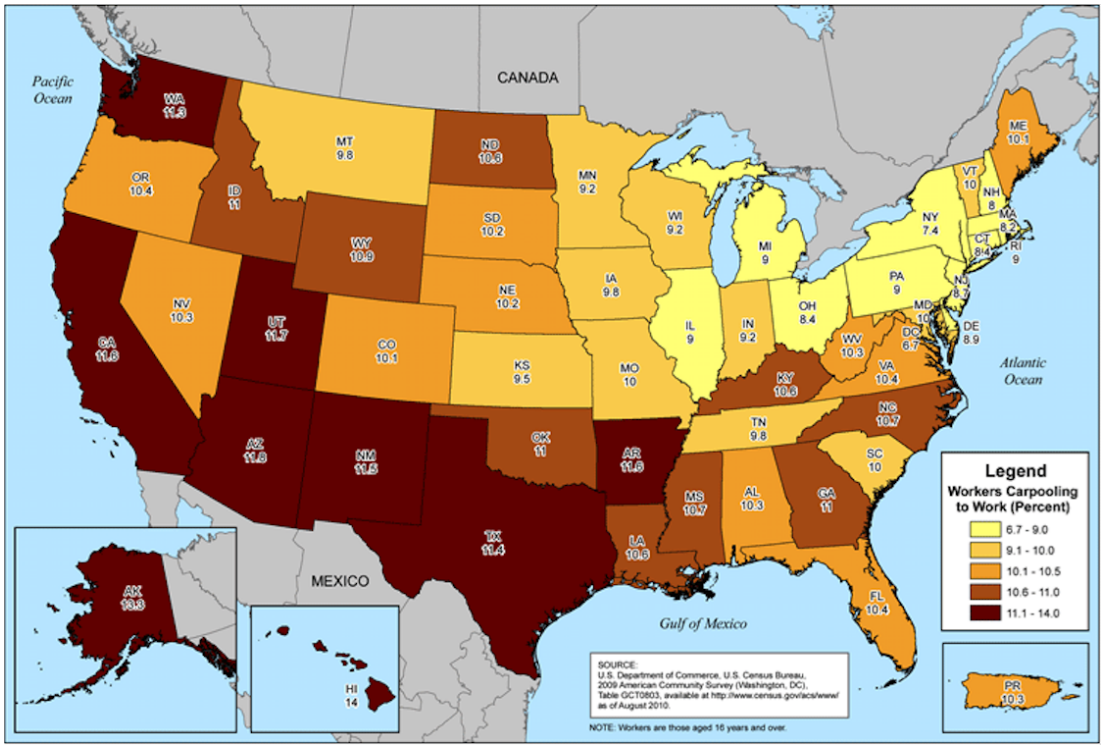


Figure 2.4: Percent of workers aged 16 years or over, carpooling by car, truck or van in the USA [103].

Examining carpooling arrangements modelled as CAS may provide insight in how the local goals and decisions made by each user belonging to one of these (internally largely homogeneous) classes impact the performance of the whole ensemble. The addition of modern communication technologies, allowing the users to exchange information adds another layer of complexity to this research problem. Traditionally, carpools were formed as queues at specific places near major roads where cars could conveniently stop to pick up additional passengers who wanted to travel in the same direction. Today additional tools, such as mobile applications, are available providing mechanisms that attempt to match the prospective ride sharers with each other [25].

2.2.1.4 Swarm robotics

Large groups of homogeneous robots interacting with each other in order to achieve a common goal are another example of human engineered CAS [62]. The ability of the swarm to adapt to the changes in the environment in order to optimize the performance of the whole collective is one interesting aspect of this problem.

In [73] the authors present a model of a collective foraging scenario, where the robots are adjusting their behaviour to achieve the most optimal (in terms of energy spent versus amount food foraged) ratio of foragers to resters.

Swarm robotics have a potential for use in search and rescue scenarios. Following natural or man-made disasters, early response is crucial for increasing the survival rate. The dimensions and boundaries of the areas that need to be explored in order to search for survivors are often unknown as they could have been altered during the disaster. At the same time, human rescue teams dispatched to the site are themselves endangered during the exploration of the area. For these reasons, a swarm of robots that would have the ability to adapt to a largely unknown and possibly still changing environment could prove very valuable for use in such scenarios [94].

The ability to self-organise is yet another desirable emergent behaviour in swarm robotics research. Robots may cluster together, forming larger entities, more capable of achieving specific goals (i.e. being able to deal with spatial barriers such as stairs and walls), or even form the required spatial structures for other robots to use in order to explore the environment (i.e. bridges) [102]. This is very much alike the behaviour of swarms in nature, such as ants or bees, described in greater detail in the next subsection.

2.2.1.5 Power management systems

In the Power Management Systems scenario, big and small power plants are grouped together and regarded as belonging to the category of CAS in order to reduce energy waste that is being generated in presently used solutions. Currently, the control and management of groups of big power plants is in the hands of several companies, while small power plants and DERs (Distributed Energy Resources) are not at all externally controlled. The management of big power plants is realized by rescheduling the desired production output using a rough estimate once in every 15 minutes. Such a naive approach to production management does not take advantage of the full capacity of the system, generating losses associated with the costs of producing and storing the surplus energy. The main benefit of taking a collective approach to this scenario is the possibility of optimizing the distribution of energy demands within a network of power plants. This can be achieved by taking into account the varying characteristics of each of the power plants and their cost-effectiveness of producing an energy unit and/or

switching on an inactive unit rather than redirecting the demand to another station [60].

2.2.1.6 Cloud computing and telecommunication

The properties of CAS such as adaptability and open-endedness are highly desirable in telecommunication and computer networks. Adaptability makes the system flexible with respect to events that are hard to predict such as power outages or hardware failures.

These systems are open-ended in the sense that they are able to continue working efficiently even when one or more of the agents fails, which results in other agents self-organising into collectively taking over the responsibilities of the failed agent. In a similar fashion, adding a new agent to such a system should not require any changes to the existing collective, as it should be able to self-adjust to its new population size. In [36] the author explores an adaptive routing policy for telecommunication networks which is reflective of the ant colony optimisation model, derived from nature.

There is ongoing research about how arrangements comprised of smart mobile telecommunication devices worn by people could be described as CAS [60, 109].

2.2.2 Nature

Many systems and structures that are present in nature can be represented as CAS models. Developing and analyzing such models can serve to improve the current state of scientific knowledge in at least the two following ways: a) by reverse-engineering the mechanisms found in nature to describe and investigate phenomena that emerge from evolutionary processes, b) by taking inspiration from nature to find better design paradigms for use in software-based systems.

These two approaches are closely related and similar in execution. The main difference between them is the result that is expected from conducting the research. In case a) the aim is to formally represent an existing system as accurately and precisely as it is possible (even if there is a possibility of improving its performance by applying changes), while in case b) the intent is to create a model that performs best in solving a preexisting problem, which may be completely unrelated to the original context of the system in nature.

In the paper [42] the authors describe a number of design patterns for use in software development, which have been inspired by various systems found in nature. The paper describes and compares a range of bio-inspired patterns for engineering self-organizing software-based systems. The patterns have been classified into three groups with respect to their complexity: a) basic patterns, b) composed patterns and c) high-level patterns.

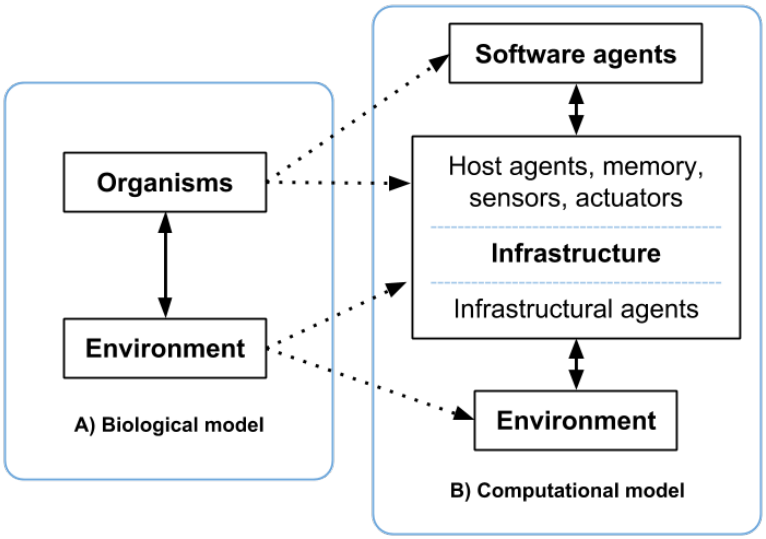


Figure 2.5: The relation between components of systems found in nature (A) and their respective computational models (B). Fernandez-Marquez et al [42] propose a computational metamodel having a middle layer called 'Infrastructure' that is required for the agents to perform actions and interact with the environment. The Infrastructure layer provides the means for performing simulation of the system, which include hosts having computational power, and hardware or software sensors and actuators.

Figure 2.5 shows how the elements of a biological model could be mapped into elements of a computational model.

Basic patterns are created using simple heuristic rules for the agents to follow when sensing and acting in the environment. For example, if an agent is able to detect the position of other agents within a certain distance, it can follow rules depending on its observations of the agents within its neighbourhood radius.

This mechanism is employed in what is perhaps the most famous example of CAS observed in nature: the *flocking* behaviour, described in [85]. The author illustrates how the emergent behavior commonly observed in nature in ensembles such as flocks of birds, herds of cattle or schools of fish, can be reproduced by aggregating three

simple heuristic rules at the level of a single agent's behaviour. The three rules are:

- Separation
- Alignment
- Cohesion

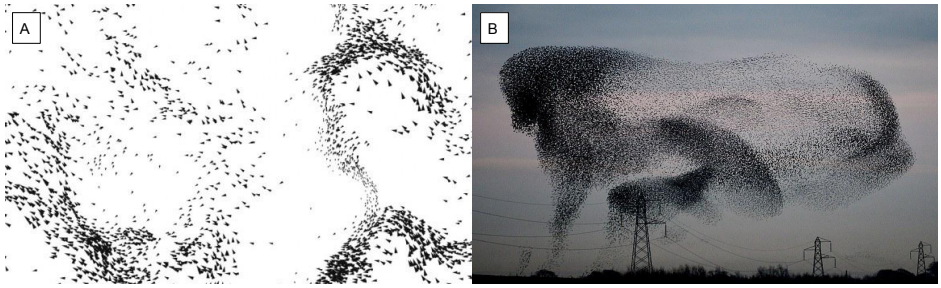


Figure 2.6: Flocking behaviour as a computer simulation (A) and observed in nature (B).

Separation (short range repulsion) is a heuristic rule which ensures that agents do not crowd together, gravitating towards a single location. From the perspective of the agent, the rule can be phrased as: 'if there is another agent within a certain range from your location, move in the opposite direction to that agent'.

There are of course multiple agents in the collective, so in practice these rules will be instantiated as forces pulling and pushing the agent in various directions with various strengths. The final movement will depend on the summation of these forces, together with those created by the alignment and cohesion rules.

Alignment is a behaviour in which the agent adjusts its direction to be the average direction of agents within its local neighbourhood.

Cohesion (long range attraction) is very similar to what would have been the opposite of separation, except the local radius determining the neighbours whose locations are taken into account when computing the forces is much larger. It is therefore responsible for the agents forming groups, while separation ensures that these groups do not collapse into single location points.

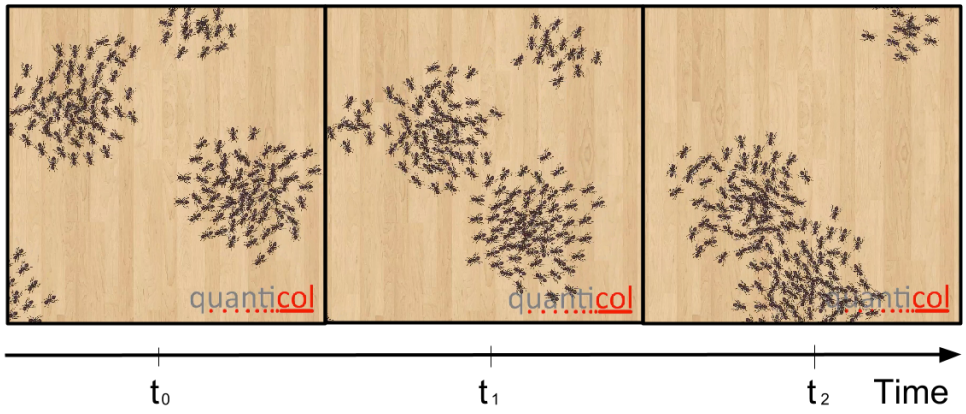


Figure 2.7: Three snapshots from a simulation of the flocking behaviour. The formation and joining of sub-groups can be observed here. Video available at `http://nataliazon.com/wp-content/uploads/2018/03/flocking_nice1.mp4`.

When aggregated, these three simple rules produce a very realistic and seemingly complex behaviour that can be observed at the level of the collective, as illustrated in Figures 2.6 and 2.7.

To an independent observer, this emergent behaviour might give the impression of being globally planned and controlled by an external agent who possesses the knowledge about the attributes and actions of every element of the system. In reality, there is no central supervision - the observed artificial intelligence exhibited by the system is a feature of the whole collective. This property of multiple-agent systems being able to make decisions is often referred to as 'swarm intelligence' [61].

Swarm intelligence is often associated with certain species of insects: bees, termites and ants. The Argentine Ant is a species of ants that are capable of finding the shortest path from the colony nest to a source of food by means of self-organized swarm decision making [53]. In the experiment conducted by the authors of [53], a colony of ants was able to find the optimal path when placed in an environment containing their nest and a source of food, connected by a bridge (see Figure 2.8).

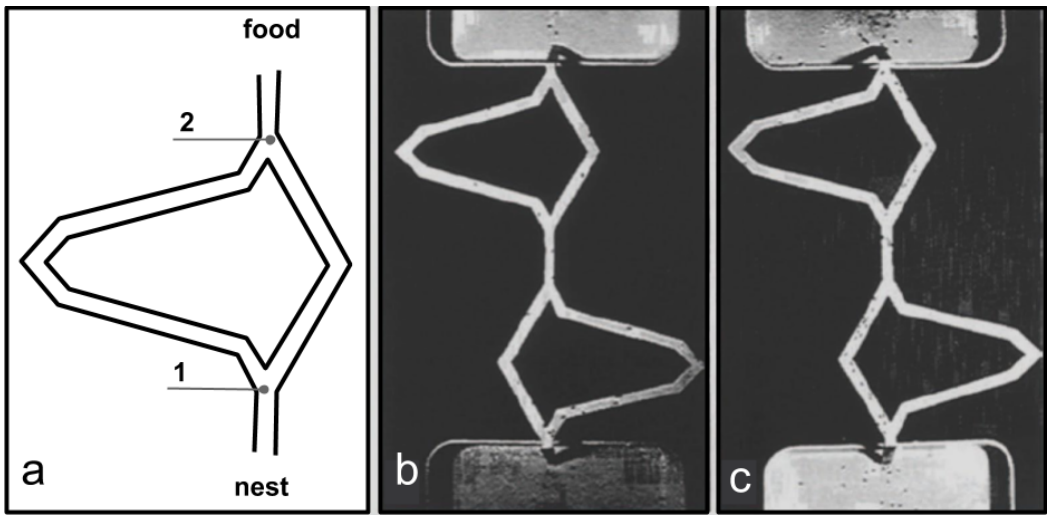


Figure 2.8: The setup of the experiment conducted by Goss et al in [53]. The bridge consists of two modules, each of which branches out into two paths, one of them being longer than the other. In the module closer to the nest, an ant ends up in the shorter path if it chooses to turn left on the fork, in the second module - if it turns right. After an initial period of exploring all paths, ants start following the shortest paths exclusively; a) a single module of the bridge, b) and c) the configuration of ants 4 minutes and 8 minutes after placing the bridge in the environment.

The discovery and analysis of this behaviour inspired researchers to develop a number of optimization algorithms for problems such as: shortest path finding, routing, set partition and distributed information retrieval [76]. In the paper [76] the authors analyze the shortest path algorithm by creating a formal model of the scenario in Bio-PEPA [100] and investigating what conditions must be satisfied for the emergence of the swarm-intelligent shortest path mechanism to emerge in stochastic simulation and fluid flow analysis. Simulating the model under such conditions yields results that can be compared with results obtained from empirical research described in [53].

Oftentimes the models representing behaviour of swarms of animals restrict their ability to exchange information with each other. In flocking, an agent adjusts its movement using information incoming from surrounding agents. This can be seen as the agent having the concept of its own neighbourhood and exchanging information only with those agents which belong to that neighbourhood. The neighbourhood is not defined by a set of points in space but rather moves along with its 'owner'. It can be spatial, which means that all the agents within a certain radius distance from the original agent belong to its neighbourhood. It can also be logical - when the neighbours' physical

positions are not important, and so, it is more of a network of all the agents that have means of communicating with the neighbourhood owner.

A number of other composed and high-level patterns are presented in [42], including: *gradient* (a pattern where information is spread in such a way that it conveys the information about the distance from the sender), *gossip* (where there is an agreement in the collective over values of certain parameters even though the information is spread in a decentralised way), and *quorum sensing* (a pattern allowing an agent to obtain information about the density of agents in the system using only local interactions).

2.3 CARMA: Collective Adaptive Resource-sharing Markovian Agents

CARMA is an expressive process calculus which has been developed specifically for the modelling of CAS. A full description of the language can be found in [48, 74].

Systems that can be represented in CARMA consist of collectives of *components* (agents) inside an *environment*. Agents' behaviour is represented by Markovian processes. The future state of the system is only dependent on its current state - the system has no memory of the sequences of decisions that were taken in order to get to the current state. In CARMA, agents perform *actions*, the rate of which is defined by an exponential distribution random variable. If at a given state there are multiple possible transitions with non-zero rates, the first one to trigger will determine the next state of the process (this is known as the *race condition*).

For the purpose of creating models in CARMA one assumes that these Markovian properties are to some extent present in the considered scenario. The independence of future on the past in a CARMA model is achieved by encoding the vital elements from the system's evolution past in its current state. For example, in CARMA one can use the components store to capture a particular value used in a past decision or even use the store variable to count how many times a particular action was triggered and then use this knowledge to influence the probabilities of future transitions. CARMA models are still truly Markovian, as each possible *combination* of component behaviour's process states and component stores' states is considered a single state in a Markovian process describing the whole system. This of course means that CARMA models become

complex fast with growing number of behaviour states and store fields in the components. The Markovian assumption is therefore not realistic for modelling a real world scenario in a fully detailed and completely accurate manner, but instead is meant for working with models where certain abstractions are introduced in order to represent the features of the system that have the prevailing influence on the performance of the whole collective.

The fact that CARMA works under the Markovian assumption allows the user to not only accurately represent, but also assess the system in terms of its performance, availability and dependability by means of simulation using a stochastic simulator, based on the *kinetic Monte Carlo* algorithm.

Components, denoted by C, are entities consisting of a *store*, denoted by γ and a *process*, denoted by P. They can also be *inactive*, which is denoted by 0:

$$C ::= 0 \mid (P, \gamma)$$

A collective of components N is defined as follows:

$$N ::= C \mid N||N$$

where $N||N$ denotes two collectives in parallel. The *store* of an agent represents its attributes, that is a set of variables associated with that agent. These attributes can change with time, and can be intuitively regarded as the agent's knowledge.

The *store* can be represented as a mapping from the set of attribute names into the set of basic type values:

$$\gamma ::= \{a_0 = v_0, ..., a_n = v_n\}$$

The *process* represents the agent's behaviour, and is defined as follows:

$$P, Q ::= nil \mid A[X] \mid P|Q$$

where P, Q are processes, $A[X]$ denotes a *process automaton* in state X, $P|Q$ is a composite process obtained by executing processes P and Q in parallel, and *nil* denotes the inactive process.

Any process can be defined as a composition of processes that are to be executed in parallel. The atomic process, which cannot be further divided, is either *inactive* or can be defined by a *process automaton*.

The process automaton is defined as the set of its states along with transition rules:

$$
\begin{aligned}
&\mathit{process}\ P = \\
&\qquad X_0 : [\pi_0^0]act_0^0.R_0^0 + ... + [\pi_0^{k_0}]act_0^{k_0}.R_0^{k_0} \\
&\qquad \vdots \\
&\qquad X_n : [\pi_n^0]act_n^0.R_n^0 + ... + [\pi_n^{k_n}]act_n^{k_n}.R_n^{k_n} \\
&\mathit{end process}
\end{aligned}
$$

where:

- π_i^j is a predicate that has to be satisfied by γ before the action can be executed,
- act_i^j is a CARMA action,
- R_i^j is either a state of the automaton $R_i^j \in \{X_0, ..., X_n\}$, the inactive process *nil*, which terminates the current process, or the process *kill*, which destroys the component

Prefix $(.)$, constant definitions $(\stackrel{\text{def}}{=})$, choice $(+)$ and parallel composition $(|)$ can be expressed in the standard manner by defining P appropriately. Additionally, there is the *nil* process which does nothing, the *kill* process which results in the component being removed from the collective, and the option of prefixing a process with a predicate $[\pi]P$, in which case the process P can only proceed if the predicate π evaluates to *true* using the values of the attributes in the component's store γ. To improve readability we sometimes parenthesise the process expression P, writing this term as $[\pi](P)$. The meaning is unchanged.

CARMA features **attribute-based communication** - predicates determine how components may send and receive messages when they perform actions. There are four types of CARMA actions, which vary with respect to the methods of communication between components:

- broadcast output $(\alpha^\star[\pi]\langle\vec{v}\rangle\sigma)$,
- broadcast input $(\alpha^\star[\pi](\vec{v})\sigma)$,
- unicast output $(\alpha[\pi]\langle\vec{v}\rangle\sigma)$,
- unicast input $(\alpha[\pi](\vec{v})\sigma)$

where α is the action name, π is the predicate and $\vec{v}$ denotes either the variable to insert the received values in (in the case of input), or the values that are being sent (in the case of output). The notation ($\vec{v}$) comes from the fact that it is a vector, that is a list of basic type variables or values. σ is an *update*, which defines changes to make in the values of variables in the component's store, when the action is executed. The predicate π needs to be satisfied for a successful synchronization of two agents on the action. If the predicate evaluates to false, the action can still be performed by the sender, provided it is of the broadcast output type.

Unicast communication is blocking; the sender cannot output values unless there is a matching input action which can be performed by another component (this is often referred to as synchronizing on an action). In contrast, broadcast is not blocking, and that is why we can use a specific form with a constant *false* predicate to allow components to act without interaction with other components, as seen in the example to follow. The constants *true* and *false* in CARMA models are written as $\top$ and $\bot$ here.

The syntax of a non-blocking broadcast on name α is $\alpha^\star[\pi]\langle\vec{v}\rangle\sigma$ where π is a predicate which must be satisfied by all processes wishing to receive this broadcast. The predicate may refer to other components' stores as well as to the local store (these references are prefixed by 'my', similar to the use of the keyword '`this`' in Java). This is how the potential partners for communication are identified. These are the only components that are able to receive the message. Importantly, even when a component belongs to this pool of potential receivers which means it is eligible for receiving a certain message, the message can become lost with a given probability specified in the evaluation context.

The vector $\vec{v}$ is a vector of values to be communicated; this vector may be empty. The suffix σ is an *update* of variables in the local store of a component. A component refers to an attribute in its own local store by prefixing the name of the attribute with the word 'my' so an update to store the value of x as the new value of $\mathsf{my}.x$ is written as $\{\mathsf{my}.x \leftarrow x\}$. As an example, the prefix process term $\mathsf{move}_k^\star[\bot]\langle\rangle\{\mathsf{my}.\ell \leftarrow k\}.M$ broadcasts that it is performing a move_k action, updates its local ℓ values, and continues as the process M. This is an individual action, since the predicate π being set to *false* prevents other components from synchronizing on it by triggering an input action of the same type.

Broadcast output is non-blocking, which means that it will be performed even if

no other component is able to receive the message. Immediately after the action is performed, the update block is used to compute the (possible) change in the local store.

In order to receive a message from a broadcast output, a component must perform the action type broadcast input. This action is used to receive any messages that have been sent via the broadcast output. The transmitted values can be used to compute the predicate.

The environment contains both the global store and an evolution rule which returns a tuple of four functions $(\mu_p, \mu_w, \mu_r, \mu_u)$ known as the *evaluation context*. Communication between sender s and receiver r on action α has both an associated *probability* (determined by μ_p) and a *weight* (determined by μ_w). These functions depend on action α and both the attribute values of the sender (in the store γ_s) and the attribute values of the receiver (in the store γ_r). The action *rate* however depends on only the attribute values in the store of the sender (γ_s); the attribute values of the receiver do not affect the rate at which a communication action is performed.

An example rate function in the evaluation context would be the following action-label dependent rate:

$$\mu_r(\gamma_s, \alpha) = \begin{cases} 1.0 & \text{if } \alpha = \mathsf{low\ rate\ action} \\ 2.0 & \text{if } \alpha = \mathsf{normal\ rate\ action} \\ 5.0 & \text{if } \alpha = \mathsf{fast\ rate\ action} \end{cases}$$

Thus the first three functions in the evaluation context determine probabilities, weights and rates that supply quantitative information about the behaviour of actions. The fourth function μ_u performs global updates, either of the attributes in the global store or of the collective by adding new components. These updates include the usual initialisation of variables, incrementing counters, or accumulating totals.

Process prefixes provide value-passing unicast and broadcast communication using predicates over the attributes in the stores of both the component which is sending the value and the component which is receiving the value. Communication between components will only take place if the predicates over both stores evaluate to *true*. The value *false* indicates that no communication partner is needed (when broadcasting). Furthermore, attribute values can be updated (probabilistically) on completion of an action.

2.3.1 CARMA example: the SIR model

SIR (Susceptible Infected Recovered) is a classic example from the field of mathematical modelling of infectious diseases.

In the simple version of this model, the population size is constant, and each agent belonging to the population is in one of the following states:

- Susceptible – these agents are vulnerable to the disease, which means they may become infected if they come in contact with an agent that is already infected
- Infected – these agents are carriers of the disease and may cause susceptible agents to become infected during contact
- Recovered – an infected agent may become recovered, in which case it is no longer able to spread the disease by infecting susceptible agents

All agents in the collective are interacting with each other in the context of a common environment, and there are no spatial obstacles (such as physical distance) preventing them from coming into contact with one another (as shown in Figure 2.9). This means that every agent has the same probability of interacting with any other agent, regardless of where in physical space they are located (as this information is simply not a part of the model).

A CARMA implementation of the behaviour of an agent in a simple SIR model is shown in Figure 2.10, and the evolution of the model in time is shown in Figure 2.11. In this case we use broadcast actions in order to take advantage of their non-blocking nature, that is we allow several susceptible agents to perform the *contact* action initiated by an infected agent, which results in the susceptibles becoming infected. In this simple scenario, no messages are sent or received when the action is triggered.

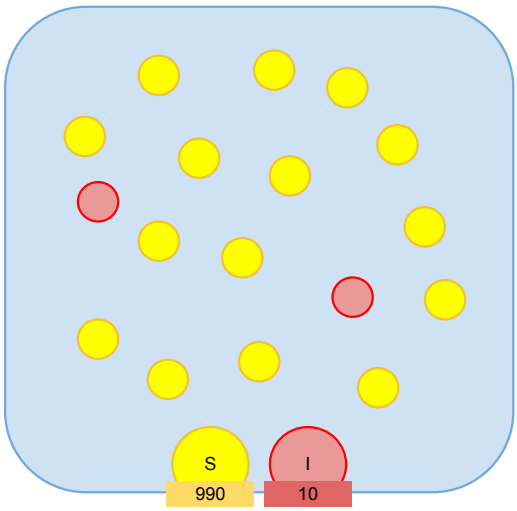


Figure 2.9: A schematic representation of a simple SIR scenario. All agents are equally likely to come in contact with one another and there is no notion of spatial location.

$$
\begin{aligned}
&process\ P = \\
&\qquad S : contact().I \\
&\qquad I : contact <>.I + recovery <> [false]^{\star}.R \\
&\qquad R : nil \\
&end process
\end{aligned}
$$

Figure 2.10: The behaviour of an agent in a simple SIR model, implemented in CARMA. '*false*' is a special value designating a spontaneous action (triggerable by a single agent) when used in place of a broadcast output action's predicate expression.

The *recovery* action is an example of a spontaneous broadcast output - there aren't any receivers with a matching broadcast input action. A special literal value *false* is passed to the predicate in order to signify that no matching receivers are needed. When an agent reaches the recover state it remains in the system but can no longer become infected (for example, due to acquired immunity). This is represented by the idle process *nil*. We assume that all the actions happen with the same constant rate. It is worth noting that there exist numerous variations of this model, for example one in which the recovered agents may lose their immunity to the disease, and become

susceptible again. Another version, in which the population is not constant, assumes that infected agents can either recover or die, in which case they are removed from the system.

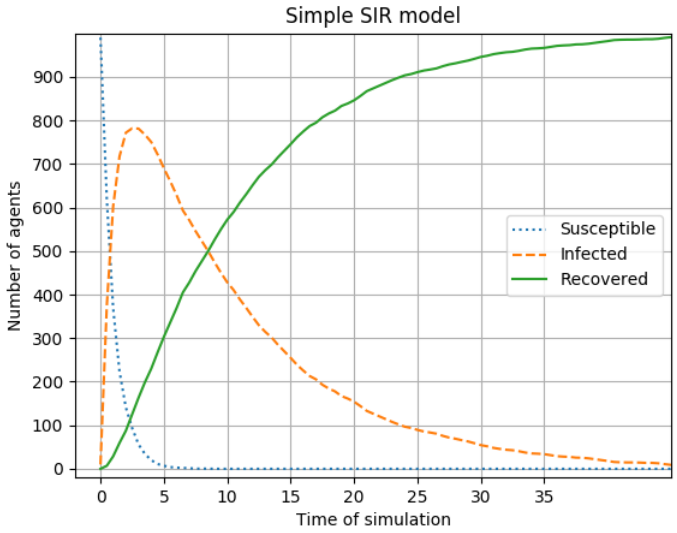


Figure 2.11: The population dynamic in a SIR scenario obtained from a CARMA simulation of the model presented in Figure 2.10. The initial populations are: 990 susceptible and 10 infected agents.

Now let us consider another similar scenario with the addition of locations in space. For example, agents may move around in a 3x3 grid as shown in Figures 2.12 and 2.13. In this case we restrict the possibility of coming into contact to only those agents which are currently residing in the same location. Agents now also have an additional process in their behaviour, running in a parallel to the one we saw in the previous example. This allows them to move around in the network by triggering the spontaneous broadcast output action *move*. The next node is chosen randomly from the nodes accessible from the current node.

Figure 2.12 shows a version (labelled 'version A') of this scenario in which both populations start at the same node. The number of pairs of agents eligible for performing the *contact* action together decreases as agents spread through the system. Since they move in a step-by-step fashion, and have a limited number of next possible nodes to move to at any given time, it it still rather likely that in the beginning phase of the evolution there will be co-located pairs of susceptible and infected agents in the system. This results in the initial early growth of the infected population, shown in Figure 2.15.

In another version of this scenario, (labelled 'version B'), we modify the initial conditions of the model so that the two populations: susceptible and infected start at the opposite corners of the spatial graph 2.13. In this case the simulation results (presented in Figure 2.16) show a much later peak of the infected population, as the infections can only begin when agents have moved through the graph resulting in infected and susceptible agents being co-located and capable of performing the *contact* action together.

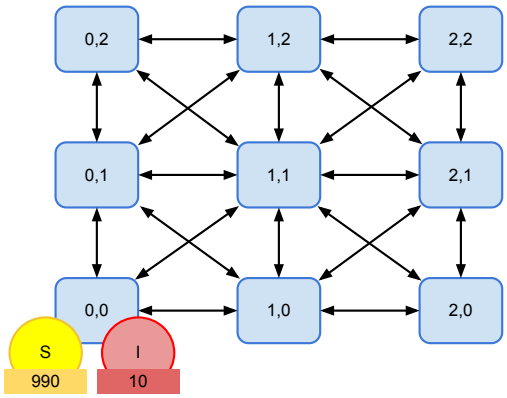


Figure 2.12: A schematic representation of a SIR scenario with locations, version A: susceptible and infected populations are co-located at the same node at the start of the system's evolution.

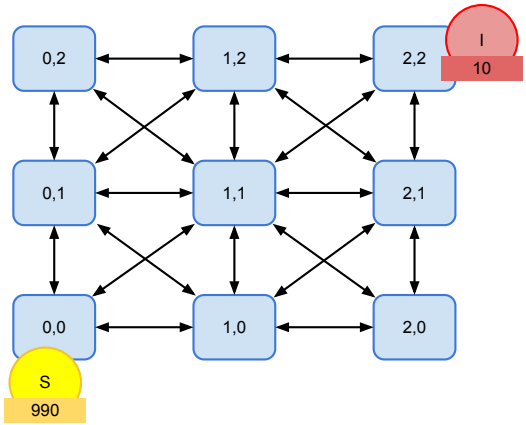


Figure 2.13: A schematic representation of a SIR scenario with locations, version B: susceptible and infected populations are located at different nodes at the start of the system's evolution.

$$
\begin{aligned}
&process\ P = \\
&\qquad R|Q \\
&endprocess
\end{aligned}
$$

$$
\begin{aligned}
&process\ R = \\
&\qquad S : contact(infected\_loc)[my.loc == infected\_loc].I \\
&\qquad I : contact < my.loc >.I + recovery <> [false]^{\star}.R \\
&\qquad R : nil \\
&endprocess
\end{aligned}
$$

$$
\begin{aligned}
&process\ Q = \\
&\qquad M : move <> [false]^{\star}\{my.loc = NewLoc(my.loc)\}.M \\
&endprocess
\end{aligned}
$$

Figure 2.14: The behaviour of an agent in a SIR model with locations, implemented in CARMA. Processes *R* and *Q* are executed in parallel manner within process *P*. We ensure that only co-located agents can come into contact with one another by adding a predicate to the *contact* action. The susceptible agent uses the value received from the infected agent in order to determine whether they are co-located or not, before triggering the action. NewLoc() is a function returning a random location accessible from the location given as the parameter. '*false*' is a special value designating a spontaneous action (triggerable by a single agent) when used in place of a broadcast output action's predicate expression.

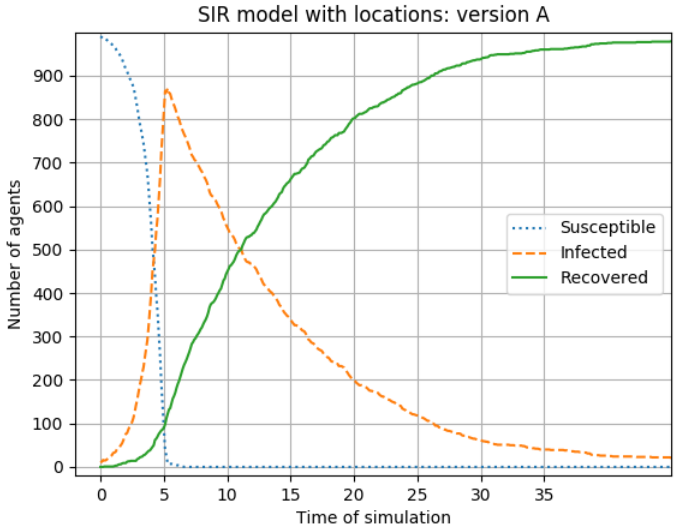


Figure 2.15: The population dynamic in a SIR scenario obtained from a CARMA simulation of the model presented in Figures 2.12 and 2.14. The initial populations are: 990 susceptible and 10 infected agents.

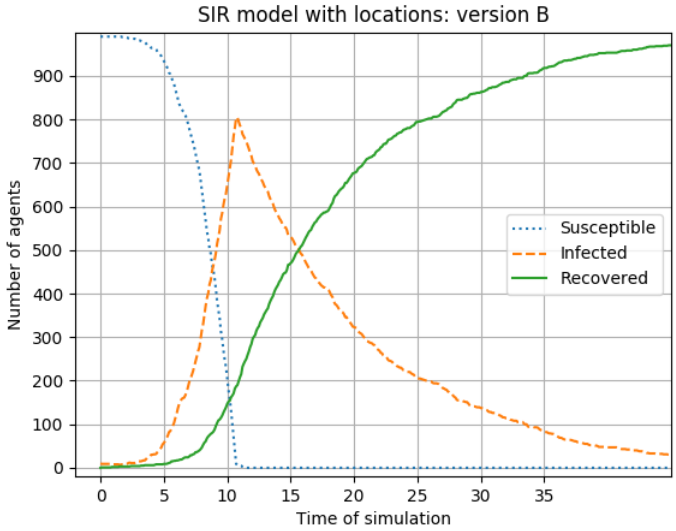


Figure 2.16: The population dynamic in a SIR scenario obtained from a CARMA simulation of the model presented in Figures 2.13 and 2.14. The initial populations are: 990 susceptible and 10 infected agents.

2.3.2 CASL

As is standard for process calculi and algebra, CARMA is the formal and mathematical language developed for collective adaptive system modelling. However, when it comes to implementing models and simulating and analysing them, a text-based language suitable for input is required. The language accepted by the CARMA Eclipse Plug-in is called CASL (CARMA Specification Language). It allows for the declaration of components and the environment as in the definition of CARMA but it also gives additional features that are necessary when making a model concrete for simulation. In particular, it allows for constants and functions to be defined to support the definition of models. In addition to this, it adds a layer of typed data structures including enumerations, record types, and heterogeneous collections such as sets and lists. This provides a level of type security which is not offered by process calculi with untyped value-passing. Furthermore, Chapter 3 introduces the new explicit spatial syntax for CASL to describe space, an important feature determining the behaviour of many CAS which was developed as a part of the research presented in this thesis. It allows the definition of nodes (either as coordinates or names) and links between these nodes. There is also syntax to support the use of this space, in particular, a way to refer to both the pre-set and post-set of a node, which then permits a generic definition of moving components that can traverse over any spatial structure specified. Taken together, these additional language features in CASL provide a basis for strong static analysis of models, catching modelling errors at compile-time which would not be detected in modelling languages without this kind of support for representation of typed data and spatial structure.

CASL provides a wrapper around the CARMA process calculus adding non-essential (but useful) features such as data types and data structures, functions, and the ability to specify real-valued *measures* of interest over the model. In some modelling languages measures of interest or Markov reward structures are defined externally to the model but in CARMA and languages such as CASPA [68], PRISM [69] and ProPPA [52], the specification of measures of interest and reward structures is included in the modelling language itself.

2.3.3 The CARMA Eclipse Plugin

The CARMA Eclipse Plugin [59] is an integrated development environment for CARMA models. It provides a helpful syntax-aware editor for CASL, implemented in the XText editor framework. Given a CARMA model, the CARMA Eclipse Plug-in compiles the model into a set of Java classes which are linked with the CARMA simulator classes to provide a custom simulator for this specific model. The compiled Java code is executed to compute the measures of interest from an ensemble of simulation runs. The operational semantics of CARMA are defined in FUTS (State to Function Labeled Transition Systems) style [32]. FUTS defines state-transition structures in which each transition is a triple of the form (s, α, P), where s is the state, α is the transition's label and P is the *continuation function*, which associates a specific value to state s' (for example the rate of the action performed in order to reach state s' from s).

CARMA defines the semantics of each model as a time-inhomogeneous continuous-time Markov chain (ICTMC). The behaviour of these ICTMCs is simulated using the CARMA Eclipse Plug-in.

2.3.4 The analysis of CARMA models

Probabilistic model checking is a procedure that can be used in order to find out whether a specific property (expressed using probabilities) is true or false in a model of a stochastic system. It is applied for the purpose of verifying specifications based on probabilities, which can for example be expressed by statements like ”the probability of the bus arriving less than 2 minutes late is at least 50%” or ”the chance of five buses breaking on the same day is at most 0.5%”.

The problem of applying probabilistic model checking to a specific model is often solved by numerical approaches, in which the desired measures are computed in an iterative (approximate) manner. There is a number of existing algorithms that have been devised for this purpose [21].

Statistical probabilistic model checking [71], based on simulation with random sampling using methods such as Monte Carlo experiments, is an alternative approach to numerical probabilistic model checking. The simulation results discussed in this thesis have been obtained by using the CARMA simulator which implements the standard kinetic Monte Carlo statistical probabilistic algorithm. The algorithm is used to select

the next simulation event to fire and draws from the appropriate weighted random number distribution to determine the duration of the event. The simulation state is updated as specified by the event which was fired and the simulation proceeds forward until a pre-specified simulation stop time is reached. This is a standard approach and research is being made on how to improve these methods in order to speed up the simulation of complex models.

The measure functions defined by the modeller are passed into the simulation environment and provide a view onto the raw simulation results at intervals which are specified by the modeller. The Apache Commons Math Library is used within the Plug-in to perform statistical analysis of the data. The *Simulation Laboratory View* provided by the CARMA Eclipse Plug-in acts as an electronic laboratory notebook, recording details of the simulation studies which have been performed.

Simulation experiments are composed and launched from the user interface via the CARMA Simulation View. Results are plotted directly into the Experiments Results View or saved to a file for post-processing.

2.4 Other existing modelling approaches

In this thesis we focus specifically on CAS models written in the CARMA modelling language, as CARMA was the tool of choice for the purpose of supporting the undertaken research. It has been extensively used by disparate groups of users in the field of formal modelling, and there are a number of existing well documented CARMA use cases in the literature [74]. The existence of this relatively wide CARMA user community, the ongoing development (at the time of planning of this thesis) of the language under the auspices of the QUANTICOL project, as well as the robust software toolkit available for the language made it an excellent candidate for the tool of choice needed in order to conduct the research presented in this thesis.

There are a number of other tools and approaches available for modelling and analysis of CAS we have considered. The purpose of this section is to provide some background information on potential alternative approaches.

- **PEPA** [58] (Performance Evaluation Process Algebra) In many ways PEPA can be considered the predecessor to CARMA. It is an earlier, less sophisticated modelling language developed within the same scientific community, and the

authors of CARMA mention it as one of the stochastic proces algebras the development of which provided lessons and experience later utilised when designing CARMA [24]. In PEPA, components undertake actions, which are assumed to have durations. The durations are represented by a random variable drawn from a negative exponential distribution. The components may synchronise on actions, but they do not have the ability to exchange messages. An extension of PEPA designed for working with biological systems, BioPEPA, was proposed in [27].

- **ProPPA** [52] (Probabilistic Programming Process Algebra) is a formal language based on Bio-PEPA, developed for working with models of dynamic stochastic systems. It includes the information about uncertainty in the model and allows for this uncertainty information to be refined through observation.

- **MultiVeStA** [104] is a statistical analysis tool which was designed with the intention of making it suitable for easy integration with discrete event simulators. An extension of MultiVeStA for the CARMA simulator is described in [49].

- **PRISM** [69] is one of the existing probabilistic model checkers. Its software implementation is as of March 2019 maintained and kept up to date. PRISM was designed for the purpose of working with various types of models, such as discrete-time Markov chains (DTMCs), continuous-time Markov chains (CTMCs), Markov decision processes (MDPs), probabilistic automata (PAs) and probabilistic timed automata (PTAs). It also supports extensions of these models, such as costs and rewards.

- **STORM** [34] is a probabilistic model checker, featuring the analysis of Markov chains and MDPs (discrete and continuous). It is compatible with, among others, the PRISM modelling language.

- **CASPA** [20] (Causality-Based Abstraction for Security Protocol Analysis) is a model checker designed for the purpose of proving properties of cryptographic protocols such as secrecy and authenticity properties. In CASPA, a given Markovian stochastic process algebra specification of a model is represented by multi-terminal binary decision diagrams (MTBDDs).

Chapter 3

Spatial CASL: Extending CASL with syntax for spatial models

> Smart data structures and dumb code works a lot better than the other way around.
>
> *Eric S. Raymond*

3.1 Overview of CASL

CASL is a specification language [48], built on top of the CARMA language, to provide a richer syntax with the aim of making the process of creating models easier for users who are unfamiliar with process algebra and similar formal notation. In CARMA, the expressions defining environments as well as the components interacting within them, are abstract from the details, which are necessary in the context of a concrete specification that is required, for example, to provide an input for the simulator. These details includes definitions of data types, structures standard functions and operators. In addition to that, supplying a syntax that follows a standard programming code style makes it easier to understand and learn by those who are not experts in formal languages. CASL provides a concrete textual syntax, which is in many ways similar to the syntax of high level programming languages, such as Java or Python, appropriate for being handled by software tools designed for parsing the models, type checking, static analysis, model checking and simulation.

The basic syntax constructs found in CASL (excluding constructs relating to space):

- **Basic data types**
  - *bool* for boolean values
  - *int* for integers
  - *real* for real values
  - *location* for spatial locations
- **Complex data types**
  - *enumerations* are sets of named values
  - *records* can be used for declaring aggregated data structures, consisting of a sequence of typed fields
  - *collections* include *lists* (sequences of values of the same type) and *sets* (unordered lists which do not include duplicate elements) and provide structures for storing homogeneous data
- **Literal expressions**
  - *none*, used to refer to an undefined value
  - *boolean literal*, one of the two constant values `true` and `false`
  - *integer literal*, a base-10 integer represented by a non-empty sequence of digits, i.e. an element in $(0..9)^+$
  - *real literal*, a floating point value represented by an element in $(0..9)^*.(0..9)^+$
- **Reference expressions**
  - *my*, used to refer to a variable in the local store, from the context of the component encapsulating that store
  - *global*, used to refer to a variable in the global store of the environment
  - *sender*, used in the context of a message (an action with data), to refer to the local store of the component from whom the action is output
  - *receiver*, used in the context of a message, to refer to the local store of the component for whom the action is input

- **Arithmetic expressions**
  - *standard arithmetic operators*: +, -, * and /, are used to combine integer and real expressions
  - *casting operators* - to convert integer values into real values (and vice versa), two cast operators `int(e)` and `real(e)` can be used (they are based on the standard casting operators from the `java.lang` Java package)
  - *built-in constants and functions* - CASL provides a set of expressions for commonly used constants, for example the values of the π number and Euler's number, as well as functions, such as the trigonometric functions and the square and cube root of a value
  - *now* is a special expression referring to the current time in the simulation
- **Boolean expressions**
  - *standard comparison operators*(`==, !=, <, <=, >, >=`) can be used to build boolean expression of the form e_1 op e_2, where op is the operator
  - *standard boolean operators* (`||, &&, !`) can be used to compute disjunction, conjunction and negation of a boolean expression
  - *the conditional operator* `(e1 ? e2 : e3)` when evaluated, this expression is equal to `e2` if `e1` is `true`, otherwise it is `e3`
- **Operations on collections** CASL is equipped with a set of expressions that can be used to perform the standard operations on collections such as insertion and retrieval of values. In addition to that, the user can use the `exist` and `filter` functions to check if the elements in the collection satisfy a particular predicate, as well as to query the collection in order to retrieve the set of values satisfying a particular predicate (using the function `filter`).
- **Random expressions**
  - *sampling random values* - the `RND` keyword returns a random variable in the interval $[0,1)$, while the `NORMAL(e1, e2)` function returns a normally distributed variable with the mean $e1$ and variance $e2$
  - *random selection* can be performed using the `U(e1, ..,en)` expression to obtain an element uniformly selected from the set of values `e1, ..,en`, or

using the `select(e1,e2)` expression, in which `e1` is a collection while `e2` is an expression that is used to compute the probability to select each element in `e1`

- **Components** A parametrized component prototype provides the general structure of a component that can be instantiated inside the system definition block. The definition of a component prototype consists of three blocks: the store, the behaviour and the initial configuration.
  - *the store* defines the data structures for keeping the values comprising the local knowledge of the component
  - *the behaviour* defines the Markovian processes that are specific to the considered components - the CASL syntax for defining those is analogous to the CARMA syntax for defining processes, described in the previous paragraph
  - *the initial configuration* defines the initial states of the processes defined in the behaviour block
- **System definitions**
  - *the collective block* contains expressions that can be used for instantiating components
  - *the environment block* provides structures for defining the global properties of the environment, such as: `store` (the globally accessible knowledge), `prob` (probabilities of actions), `weight` (weights for actions that alter the probability if more than one action can be triggered), `rate` (rates of actions), `update` (the update to be applied on the global store after each action is triggered)
- **Measures** Measures are expressions used to probe the simulation results to obtain specific desired values. These can be directly referenced variables from the model definition, or they can comprise multiple values processed into statistics, using a variety of available syntax constructs.
- **Other standard elements** CASL provides the standard Java-like syntax for declaring a variety of elements that are commonly found in programming languages, such as:
  - *variable declaration*, with or without value assignment
  - *value assignment*

– *if-then-else*

– *for loop*

– *return*

– *block*

3.2 Additional syntax for describing space

Textual formal languages are a powerful tool for the specification and analysis of complex systems. Many of them do not include an explicit representation of space in their syntax – CASL being an example. In this section we describe Spatial CASL - the additional syntax dedicated for the specification of space, which was designed as an extension of CASL. The addition of syntax constructs specifically designed for describing space is motivated by the fact that most of the scenarios that have been implemented in CARMA have some spatial aspects.

3.2.1 Locations in space

A new basic type, `location`, was added to provide a type for referring to a location in space.

The space in which the system operates can be defined as a graph, the nodes of which represent possible locations, and the edges represent connections between these locations.

A new kind of block, `space`, was introduced to provide a way of defining sets of general properties of the spatial structure of the model. A single CASL document can have multiple `space` definitions, one of which may be selected for use (instantiated) in the model within the `system` block.

In the subsections below we present each of Spatial CASL features in detail.

3.2.2 Space block name and parameters

The `space` block must be named and can be parametrized with a list of typed values:

```
space <name>(<type_1> <name_1>,...,<type_n> <name_n>) {
...
}
```

The name associated with the space would usually describe the kind of space that is being defined in that block, for example `3dGrid` or `HexagonalGrid`.

3.2.3 Universe

Each space is associated with an universe, which can be intuitively understood as the coordinate system used in the space definition. The universe is defined as a sequence of typed fields:

```
universe < <type_1> <name_1>,...,<type_n> <name_n> >;
```

For most two-dimensional spaces, the universe consists of two values, storing the x and y coordinates on the Cartesian plane. For particular kinds of regular spaces, for example the hexagonal grid, an alternative coordinate system might be more useful. An example of this is the hexagonal grid, which can be elegantly defined over a 3-valued coordinate system, as shown in Fig. 3.1.

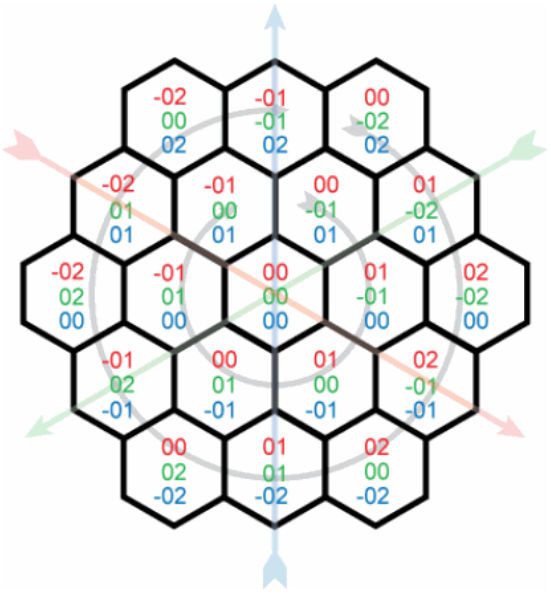


Figure 3.1: The 3-value based coordinate system of the hexagonal grid.

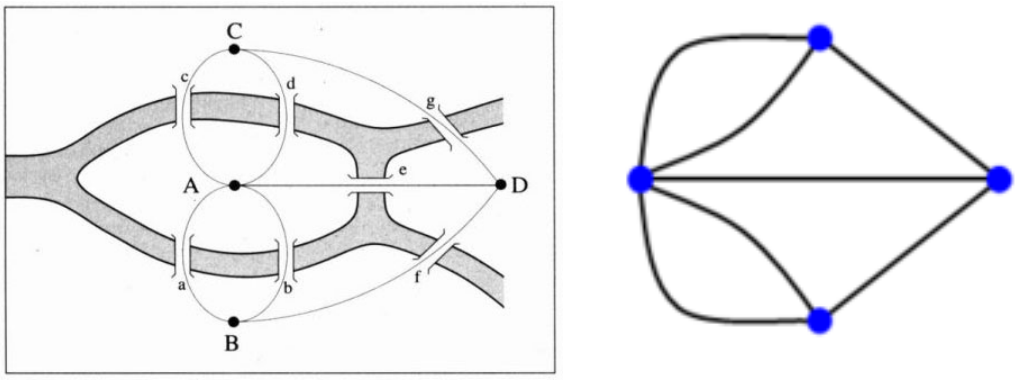


Figure 3.2: Konigsburg Graph

3.2.4 Nodes

The `nodes` block contains the declarations of nodes (which must form a subset of the nodes in the `universe`) that are used in the model. This structure was introduced to provide a way of adding limits to the spatial graph, since spaces generated by the universe declaration contain infinite numbers of nodes.

Nodes are declared via the `<node_def>` statement. This statement can be used to assign an (optional) name to each node and the corresponding position in the coordinate system of the universe:

```
1    <name_1>[e_1,...,e_n];
```

For example, the declaration of the graph corresponding to the classical Konigsburg bridge scenario of Fig.3.2 has the following form:

```
1 space Konigsburg {
2    nodes {
3        A;
4        B;
5        C;
6        D;
7    }
8    ...
9 }
```

If a particular location is assigned to each node, the same example can be defined as:

```
1 space Konigsburg {
2    universe<int x, int y>;
3    nodes {
```

```
4         A[0,0];
5         B[0,-1];
6         C[0,1];
7         D[1,0];
8     }
9     ...
10 }
```

Nodes can also be unnamed, or the names can serve as labels to differentiate between different classes of nodes (in which case multiple nodes share the same name), like in the following example:

```
1  space OfficeBuilding {
2     universe<int x, int y>;
3     nodes {
4         [0,0];
5         [0,-1];
6         [0,1];
7         [1,0];
8         Ground[0,3];
9         Ground[2,3];
10        Level1[0,0];
11        Level1[2,0];
12        Level1[2,1];
13    }
14    ...
15 }
```

In order to uniquely reference a node, the same construct as the one declaring it in the `nodes` block, must be used.

Nodes can be also defined using the following *iterative* statements:

```
1 for x from e_1 by e_2 to e_3 {
2     <node_def>
3 }
```

```
1 for x in e {
2     <node_def>
3 }
```

```
1 if e {
2     <node_def>
```

```
} else {
    <node_def>
}
```

For example, in a square grid with the origin at its centre, the nodes are defined in the following way:

```
space SquareGrid(int width, int height) {
   universe <int x, int y>;

   nodes {
        for x from -width/2 to width/2{
           for y from -height/2 to height/2{
               [x, y];
           }
        }
   }
   ...
}
```

The special `location` type, as well as the universe and space definitions provide an additional error-checking mechanisms for the model. Node which have not been declared in the `nodes` block inside the space definition, cannot be assigned to an attribute of the `location` type.

3.2.5 Areas

Areas are constructs designed to provide a way of associating collections of nodes with labels.

For example, in a power grid scenario, the modeller might want to differentiate between nodes containing a power plant and a node containing a power consumer.

```
space SquareGrid(int width, int height) {
   universe <int x, int y>;

   nodes {
        for x from -width/2 to width/2{
           for y from -height/2 to height/2{
               [x, y];
           }
```

```
        }
    }
    areas{
        plants{
            [0,0];
            [2,2];
        }
        consumers{
            for x from -width/2 to width/2{
                for y from -height/2 to height/2{
                    [x, y];
                }
            }
        }
    }
}
```

In a more abstract scenario, the distinction between certain graphical structures of the graph, such as the corners, diagonal and the border of the regular grid, might be desired. This can be obtained in the following way:

```
space grid( int width , int height ) {
    universe <int x, int y>;
    nodes {
        for x from 0 to width {
            for y from 0 to height {
                [x,y];
            }
        }
    }
    areas {
        corner {
            [0,0];
            [width-1,0];
            [0,height-1];
            [width-1,height-1];
        }
        diagonal {
            for i from 0 to min(width,height) {
                [i,i];
            }
        }
```

```
22        border {
23            for i from 0 to width {
24                [i,0];
25                [i,height-1];
26            }
27            for j from 0 to height {
28                [0,j];
29                [height-1,j];
30            }
31        }
32    }
33    ...
34 }
```

Areas can be used in models when a node needs to satisfy certain properties (be of a certain kind) in order to be a candidate for the next node in the sequence of a components' movements. For example, in an evacuation scenario, some nodes may belong to the area 'dangerous' and others to the area 'safe'. A component's behaviour may be then specified in such a way that it is never allowed move from a 'safe' node to a 'dangerous' one:

```
1 A = move[newLocation.safe || currentLocation.dangerous](newLocation)
      .{my.currentLocation := newLocation;}.A;
```

3.2.6 Connections

The `connections` block provides ways of instantiating the edges of the spatial graph, and associating them with labels and values. These are especially useful in scenarios with mobile components, whose ability to traverse a certain path might depend on a set of predicates over the connections comprising that path. One such example is the carpooling scenario, in which cars with the number of passengers higher than a certain value are allowed to move over fast road lanes, while other cars can only move over slow road lanes. A connection can be declared either explicitly or using expressions relating to the positions of nodes.

The `->` operator is used to define a directed connection, and the `<->` operator is used to define an undirected connection:

```
connections {
    node1 -> node2;
    node1 <-> node2;
}
```

```
    connections {
        for i from 0 to width-1 {
            for j from 0 to height-1 {
                [i,j] <-> [i+1,j];
                [i,j] <-> [i,j+1];
            }
        }
    }
```

Each connection can be associated with a set of *features*. These features can be later used in evaluation of predicates for component movement. For example, in the car-pooling scenario, if a connection is associated with the feature `fastLane=true`, only cars having a particular number of passengers aboard will be allowed to use them.

The *features* of the space's connections can be defined using the following syntax:

```
connections {
    node1 -> node2 {fastLane = true, weight = 1.0};
    node1 <-> node2 {fastLane = false, weight = 0.5};
}
```

In the above example we use boolean fields to identify the connection type, as there are only two types of connections in this particular model. If a scenario requires the model to distinguish between multiple types of connections, an integer field can be used instead in order to store the connection's unique identifier. Currently, CASL does not support string variables, but if these are included in the future, a more human-friendly connection's name could be held in such a variable in a similar way.

3.2.7 Location and connection expressions

The Spatial CASL syntax is also equipped with a set of expressions to access the data relating to space:

- `l.post`: indicates the locations in the postset of location `l`;

- `l.pre`: indicates the locations in the preset of `l`;
- `l.name`: indicates the location name;
- `l.xi`: refers to the element `xi` of the position of `l`;
- `l.area`: is a boolean expression that can be used to check if location `l` is part of the area `area`;
- `locations`: indicates the set of all locations;
- `area_name`: is used to access the set of locations with label `area_name`;
- `edgeValues(loc1, label, loc2)`: returns the value associated with the feature `label` of the connection `loc1->loc2`;

Below is an example of how connections and their expressions can be used in the definition of a components' movement through a graph. We assume that the component can only move to another location if there exists a connection from its current location to the location it is attempting to move to. Using the Spatial CASL syntax, this can be easily obtained with the action predicate, in the following way:

```
A = move[newLocation in my.currentLocation.post](newLocation).{my.
    currentLocation := newLocation;}.A;
```

Additionally, we may suppose that the agent is allowed to move along a connection, only if that connection's features satisfy certain properties, like so:

```
A = move[newLocation in my.currentLocation.post && 1.0 in edgeValues
    (my.currentLocation, weight, newLocation)](newLocation).{my.
    currentLocation := newLocation;}.A;
```

3.3 The merits of Spatial CASL

Before Spatial CASL existed, models with locations and connections needed to hard-code these within CASL functions, as shown in Listing 3.1. This was usually done by defining functions that would take two locations as an argument. For every connection one would want to define between two given nodes, there had to be a case within that function which would return true if these two locations were given in the function call. In the case of a dense network this quickly explodes into many lines of repetitive and hard to maintain CARMA code.

Listing 3.1: Carma code snippets showing one possible way of encoding spatial information in CASL

```
fun bool ExistsPath(int xFrom, int yFrom, int xTo, int yTo){
    if (xFrom == 0 && yFrom == 0 && xTo == 1 && yTo == 1){
            return true;
        }
    if (xFrom == 0 && yFrom == 0 && xTo == 1 && yTo == 2){
            return true;
        }
    if (xFrom == 1 && yFrom == 1 && xTo == 1 && yTo == 2){
            return true;
        }

    ...

    else return false;
}

component Mover(int x, int y, process Z){
    store{
        attrib x :=x;
        attrib y :=y;
        attrib nx :=0; //next x
        attrib ny :=0; //net y
    }
    behaviour{
    A = [ExistsPath(my.x, my.y, 1, 1)]
            choose_x1y1*[false]<>{my.nx := 1; my.ny := 1;}.M
      + [ExistsPath(my.x, my.y, 2, 1)]
            choose_x2y1*[false]<>{my.nx := 2; my.ny := 1;}.M
      + [ExistsPath(my.x, my.y, 0, 0)]
            choose_x0y0*[false]<>{my.nx := 0; my.ny := 0;}.M
      + [ExistsPath(my.x, my.y, 1, 2)]
            choose_x1y1*[false]<>{my.nx := 1; my.ny := 2;}.M

          ...
```

```
            + [AtGoalA(my.x,my.y)]finish*[false]<>.kill;   //
                remove agent when at goal
        M = move*[false]<>{my.x:=my.nx;my.y:=my.ny;}.A;
    }
    init{Z}
}
```

With Spatial CASL the definition of the spatial structures is more dense and contained within the scope of the block of code designated for this purpose, as shown in Listing 3.2. There are no global functions holding encoded data, now the data has a designated non-generic structure to be held within. In addition to that, the behaviour can be simplified by introducing the choose action. This is an action which involves message exchange between agents and nodes in the system (both represented by CARMA components). Its predicate, `nodeLocation in my.current.post` uses the new `post` operator to automatically involve only those nodes that are within the reach of the agent at its current node. This eliminates the need for the `ExistsPath` function, seen in the previous example. In addition to that the behaviour definition of the Mover component remains unchanged, regardless of the changes in the definition of the spatial network over which it moves. In the previous example, a change to the definition of the Mover's behaviour, following a change in the definition of space, would have been necessary.

Listing 3.2: Carma code snippets showing Spatial CASL structures

```
space Indexed2DGrid (){
        universe <int id, int x, int y>
    nodes {
        [0,6,7];
        [1,6,4];
        [2,1,2];
        [3,3,1];
        [4,6,0];
        [5,9,1];
        [6,12,2];
    }
    connections {
        [0,6,7] -> [1,6,4];
```

```
        [1,6,4] -> [0,6,7];
        [1,6,4] -> [2,1,2];
        [1,6,4] -> [3,3,1];
        [1,6,4] -> [4,6,0];
        [5,9,1] -> [1,6,4];
        [5,9,1] -> [4,6,0];
        [5,9,1] -> [6,12,2];
        [6,12,2] -> [1,6,4];
        [6,12,2] -> [1,6,4];
        [6,12,2] -> [5,9,1];
    }
    areas {

    }
}

component Mover(location start, location goal, process Z){
    store{
        attrib location current := start;
        attrib location goal := goal;
        attrib location next := start;
    }
    behaviour{
      ReadyToChoose =
        choose*[nodeLocation in my.current.post]
             (nodeLocation){my.next := nodeLocation;}.
                ReadyToMove;
      ReadyToMove =
         move*[false]<my.current, my.next>
         {my.current := my.next;}.ReadyToArrive;
      ReadyToArrive =
         [my.current == my.goal]arrive*<>.kill
             + continue*[false]<>.ReadyToChoose;
    }
    init{Z}
}
```

Chapter 4

CARMA Graphical Plugin

> I have all the tools and gadgets. I tell my son, who's a producer, 'You never work for the machine; the machine works for you.'
>
> *Quincy Jones*

4.1 Introduction

Formal modelling languages provide powerful tools for modelling the behaviour of a system. They are most commonly implemented in a textual form, the precise and unambiguous nature of which supports the motivation of creating explicit and accurate descriptions of a given system. However, text-based system specifications tend to grow significantly in length and intricacy when large systems are considered. This results in the development of models which are not easy to understand and work with even for users who have experience in the particular modelling language.

Graphical approaches to formal modelling provide an alternative way of building system specifications [90], which has a number of advantages over the text-based manner. It is more intuitive, especially when working with systems in which the geographical or conceptual locations of elements play a significant role, to be able to lay out these components graphically. By doing so, as an immediate by-product of the process of graphical model creation, we also obtain a schematic visual representation of the system, one which is often created separately and at a later stage, when dealing with

textual models. Graphical representations, such as plots, graphs, charts and schematics are commonly produced to illustrate the concepts behind particular instances of models, as well as to present the results obtained by means of simulation. The reason why these approaches are so commonly used, in contrast to, for example, text filled tables for data representation, is because they are easier and faster to read and understand. By allowing the modellers to work graphically, we are in effect bringing these positive aspects of graphical representation from the later stages of model analysis to the earlier stages.

Examples of such systems, described in greater detail in Chapter 2, can be found across a broad spectrum of domains. In this work however, we focused specifically on systems coming from the realm of urban planning.

The CARMA Graphical Plugin (CGP) is a software tool designed for the purpose of graphical specification of CAS. CGP is especially useful for working with those CAS in which the physical locations of agents and other model elements play an important role in the modelling and subsequent analysis of the system.

In this setting, systems often contain components whose movement in space is restricted in some way. Vehicles and pedestrians are generally moving along predefined routes imposed by the environment, such as streets or paths. CGP was developed for the purpose of working with such systems with constrained movement.

4.2 Existing graphical modelling languages and tools

There are a number of existing graphical approaches to modelling systems and architectures. They can be classified in the following way:

- Graphical languages for modelling software architecture
- Graphical languages for modelling algorithms and processes
- Graphical languages for modelling behavioural aspects of multi-agent systems
- General purpose object-oriented graphical languages

It is important to note that these categories are not exclusively disjunctive, and many of the languages belonging to one class may be applied to problems from another.

4.2.1 Graphical languages for software architecture and requirements

Architecture Description Languages (ADLs) [6] are a family of modelling languages used for specifying conceptual models of software architectures. There are many existing implementations of ADLs, and most of them have a graphical syntax. Notable examples include: ABACUS [1], a modelling language equipped with a comprehensive set of software tools implemented in HTML5, and Build Your Own ADL (byADL) [7], which consists of a metamodel that can be used for generating ADLs tailored for specific business needs.

Behavior Trees [87, 107] are a formal, graphical language often used to express the stakeholder requirements for software systems. This approach provides an intuitive graphical tree notation. Individual trees represent units of requirements, and can later be merged into a model representing the whole system.

4.2.2 Graphical languages for modelling algorithms and processes

A flowchart [2] is a graphical representation of an algorithm or a (stepwise) process. In a flowchart, different kinds of boxes are used for representing different kinds of expressions. These boxes are then connected with arrows to illustrate the progression through the algorithm. The two main classes of boxes are *activities* and *decisions*. Activities denote the execution of a particular operation, while decisions branch out through multiple outgoing arrows, the choice of which depends on satisfying a particular condition.

Business Process Model and Notation (BPMN) [5] is a graphical approach to specifying business processes. The graphical aspect of this modelling tool is realized through Business Process Diagrams (BPD), a flowchart-like schematic representation from the domain of business management. The main aim for creating BPMN was to provide a uniform standard that could be easily understood by business stakeholders.

4.2.3 Graphical languages for modelling behavioural aspects of multi-agent systems

Petri nets [81] are a modelling language used for the description of distributed systems. A petri net is comprised of *places* and *transitions*, which are linked through *arcs*. *Input places* have an outgoing arc, connecting it to a transition, and *output places* have an arc incoming from a transition. In addition to that, there are *tokens* distributed among the places. The triggering (*firing*) of a certain transition depends on whether there are sufficient tokens in its input places.

4.2.4 General purpose object-oriented graphical languages

Perhaps the most widely used graphical modelling language is the Unified Modelling Language (UML) [44]. UML is an extremely flexible tool, providing modellers with an extensive range of elements that can be used to represent conceptual, software-based and physical components of a system, as well as different types of connections establishing relationships between them. UML is in itself objective (every UML element is an abstract class with no superclass), and it is also specifically tailored for specifying object-oriented models. The advantages and disadvantages of UML both stem from the same feature of the language: its extremely rich and broad collection of basic elements. Almost any system can be expressed in UML, and more often than not, in more than one way. Because of this vastness of choice, UML models have a tendency to become large and over-complicated with time. In addition to that, modellers having different levels of familiarity with its syntax, or even simply different modelling styles, may produce very different realizations of the same system. UML however remains the standard for graphical modelling approaches and many narrower-scoped implementations have stemmed from it in response to its shortcomings [63, 86, 93].

Integration DEFinition (IDEF) [79] is a family of modelling languages which were created with the aim of being generic, neutral and reusable. The sub-modules of IDEF are indexed with a suffix and each has a specific modelling application. For example, IDEF0 deals with function modelling, IDEF1 is designed for information modelling (including IDEF1X which specializes in database design problems), IDEF4 for object-oriented design and IDEF5 for ontology description capture. Among these, IDEF0 and IDEF5 have a graphical representation.

4.2.5 Graphical languages for spatial information

In this chapter we focus on designing a graphical language for the purpose of representing spatial information. There are a number of existing systems of this kind, the examination of which has helped us understand what features should be included in CGP.

Geographical Information Systems (GIS) [28] are systems designed for working with spatial data, mainly geographical data. A GIS is a rich framework which consists of tools and specifications relating to storing, manipulating, representing and analysing spatial data. For example, it classifies geographical objects in terms of their desired digital representation into subsets such as discrete objects (i.e. buildings, landmarks) and continuous data fields (i.e. elevation or rainfall). These digital data representations are then mapped to graphical references such as raster and vector images. A GIS applies composition to graphical primitives (points, lines and polygons) in order to construct graphical representations of the stored information. A number of graphical representation of specific spatial features are defined, such as shading and contour lines to represent altitude.

In [33], the authors identify the key elements for spatial analysis and divide these into four classes: basic primitives, spatial relationships, spatial statistics and spatial data infrastructure. A subset of these elements is relevant to this thesis scope and consists of: place, attributes, objects, multiple properties of places, fields, topology, distance, direction, neighbourhood, spatial metadata. The building blocks of spatial analysis are identified and consist of models which allow to represent the spatial information, as well as geometric operations for modifying the models and queries for applying measures.

Google Maps [12] is a widely used system for graphical representation of spatial data. We used the API provided by this service in order to obtain the information about the geographical locations of points in interest (see Chapter 6).

4.2.6 CGP among other graphical modelling tools

CGP, being an extension of CARMA, can be classified as a graphical tool for modelling the behavioural aspects of multi-agent systems. It differs from the languages described

above in its strong focus on the movement mechanics and spatial distribution (topology and distances) of the agents within the collective.

4.3 The two perspectives of graphical model representation

In order to design a graphical language suitable for modelling a concrete set of problems (which in this case are CAS) there are a number of things that one needs to consider.

One of them is the possibly divergent interests of potential users of the tool. The graphical tool forms an additional layer in the hierarchy of model types (see Fig. 4.1). Because of that, the process of designing it should be considered from at least two disparate points of view - those of modellers viewing the version of the system through layers on top of and underneath the new graphical model layer (see Fig. 4.1). In this case, the two layers are 'conceptual model' and 'modelling language representation'.

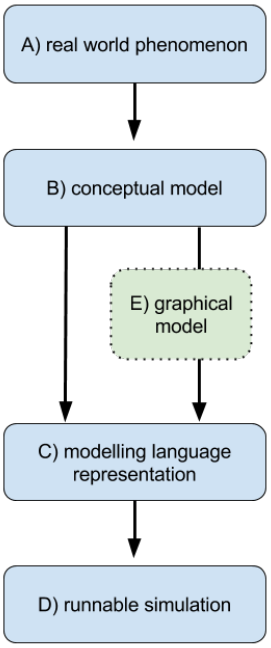


Figure 4.1: A hierarchical view of possible representations of a system, including the new graphical layer.

When trying to instantiate a conceptual model, the modeller tries to represent all of

its aspects as closely as it is possible to the original. The graphical tool can therefore be a great aid when representing space, however it may also limit the flexibility and accuracy of the original model in order to support translation into formal language notation, or even, to some degree, in order to impose a simplification of the original idea. For example, in the bike sharing scenario [95], bike stations are distributed within various places of an urban area, which can be accessed by their geographic coordinates, represented by real numbers.

A good simplification (one that reduces the complexity while not diverging too far from the original) of this system, is to restrict the stations' locations to a regular grid, superimposed on a map of the examined area.

For the purpose of this research, the modelling language of choice is CARMA. One of the planned capabilities of the graphical tool is to be able to generate syntactically correct CARMA code files from the graphically specified input. Because of that, the same assumptions that the structure of the CARMA language might force us to make about the model will also be embedded in the graphical language, and thus the same limitations will be imposed on the modeller.

These limitations might not be pronounced in the visualisation, nevertheless the user will need to remember that the provided specification will ultimately be translated into CARMA, where these limitations are present. For example, it might not be obvious just from looking at the graphical view, that CARMA actions are probabilistic, however this is a piece of information that the modeller needs to take into account while specifying the chosen system.

4.4 Graphical and textual input

Another important point is the fact that the graphical layer is not going to be a complete representation of the model, which means that the conceptual model cannot be simply replaced by the graphical model in order to provide input for CARMA code generation (as depicted in Fig.4.1 by two inputs to the CARMA layer). This is because some features of models must be fully customisable, in order to provide the user with the ability to accurately represent a broad spectrum of systems. These features include constructs like functions and actions.

In CARMA the textual syntax allows the user to specify a mathematical or logical

expression of any complexity to be used in definitions of functions or predicates. A mathematical or logical expression can be easily translated into code for the purpose of computation. It is neither easy nor intuitive to work with such concepts graphically, and therefore it is best to leave the user with the ability to define them directly as CARMA code, possibly in a text editor embedded in the graphical GUI of the proposed tool.

4.5 Systems with restricted movement

There are many examples of CAS in which the components not only move through physical space, but at the same time obey specific sets of movement rules.

For example, in bus systems, we can distinguish components that never change their location (bus stops), components whose movement follows a specific path (buses), as well as components that can move without additional restrictions (bus repair service vehicles, pedestrians).

Other urban transport systems (such as carpooling, trams and bike-sharing) also have components subject to one or more movement restrictions.

An important characteristic of these systems is that the spatial locations of individual components can have a significant influence on the performance of the collective as a whole. We can classify these effects into two categories: *direct* and *indirect* influence.

- A direct influence is observed when an agent is allowed (or forbidden) to perform specific actions based on the values of its location attributes.
- An indirect influence is not related to the state of an individual component but is instead imposed by the environment or the state of the whole system. One common example of this is a situation in which the time taken to traverse a path connecting two points is proportional to the distance between the locations of the two points in space.

When working with the CGP we focus on systems in which the movement of components is constrained to follow certain routes in space, each route defined by a path.

More precisely, we consider systems which have the following properties:

1. The environment of the system contains the definition of one or more *paths* (represented by graphs) which specific groups of components can traverse in order to change their location.
2. Components can be classified into one of three groups based on their ability to move in space:
   (a) *Stationary components* – their location attributes are constant.
   (b) *Path-bounded components* – can only move along specified paths, their location attribute values belong to the set of node locations of nodes within the specified paths.
   (c) *Free components* – can freely change their location attribute to any value (but are still bound by the environment's definition of space, i.e. a grid).
3. The spatial locations of components within the system contribute either directly or indirectly to measures calculated during model evaluation.

In other words, we are interested not only in the topological arrangements of the locations of components but also in the distances between nodes.

4.5.1 Example: The Meadows city park

The Meadows city park in Edinburgh and its surrounding area provide a good example of an environment in which different classes of components obey specific movement restrictions (see Figure 4.2). Motor vehicles, bikes and pedestrians can all move along roads surrounding the park. In addition to this, there are paths that only bikes and pedestrians can use.

This classification can go a level further if we also consider the fact that roads have special lanes for buses, taxis and service vehicles to use. In addition to that, some roads that lack pavement space do not allow pedestrian movement. Some, but not all, of the paths inside the park have separate bike lanes; this results in the cyclists being able to move at a higher speed. Some rules may not be enforced by law, but are generally accepted in most urban scenarios. For example, pedestrians moving along congested paths are more likely to keep to a particular side (in Britain - to the left), which increases the overall throughput of the path (we further explore this mechanism of crowd-routing in Chapter 5).

Examples of systems with constrained movement from outside the realm of urban planing include heterogeneous computer networks, secure computer networks, animal migration networks, and many others.

4.6 Representing Space Graphically

It is no coincidence that maps are made by drawing the shapes of the elements observed in a specific location, rather than, say, providing a list of GPS coordinates of the points that make up those shapes. Human minds are very visually oriented and it is generally much easier for us to grasp locations and distances when they are laid down in the form of a drawing (even with the extra scale factor that we need to take into account), than it would be if we were instead provided with a long list of numbers and associations, defining the nodes and edges of shapes comprising the area.

The CARMA suite provides a great toolset for working with CAS. CASL is a textual language for system specification. The aim of the work described in this chapter was to enrich the CARMA toolset with a Graphical User Interface (GUI) for an additional, more intuitive way of specifying CAS which have components obeying movement restriction rules.

The following subsections outline the key elements available to the modeller in our graphical editor; essentially these are a graphical interface for specifying components, paths and movement rules, as well as a drag-and-drop palette for laying down the

Figure 4.2: An aerial view of the Meadows City Park in Edinburgh. The roads are coloured in purple, and the paths are coloured in green. (Screen-shot from Google Maps.)

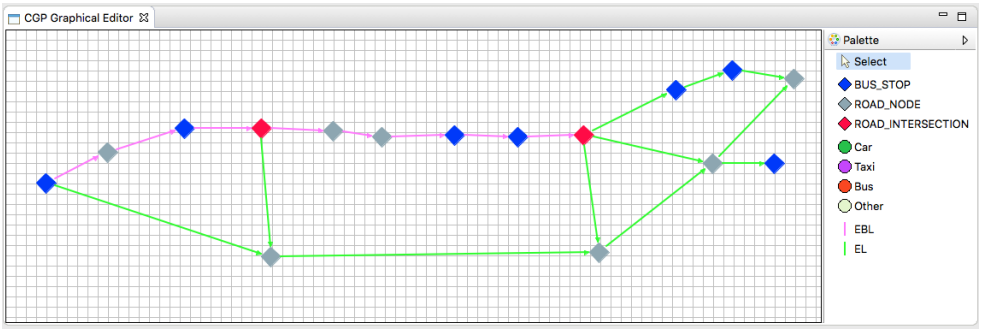


Figure 4.3: A screenshot of the graphical interface for path and components layout.

graphical elements at specific locations in space.

4.6.1 Paths

Paths are graphically represented by graphs consisting of nodes, connected by edges. Nodes are placed on a grid which is an unbounded 2D plane, tessellated by rectangles to define grid points. To reflect their placement on grid points every node has a location attribute which is a coordinate in two-dimensional space. The edges in a path graph are directed and coloured (see Fig. 4.3). The direction of an edge constrains movement on that edge to be in that direction. The colour of an edge constrains the types of components which can move along the edge.

The graphical palette allows the user to instantiate nodes, and the paths connecting them, by laying out the nodes on the grid. From the user's point of view, the creation of path node instances is very similar to the creation of component instances.

The edges of this graphical representation are translated into CASL *connections* in the process of code generation. For clarity, in this thesis the word 'edges' is used when referring to the graphical elements connecting nodes in the graphical specification and 'connections' when referring to the CASL model elements. The relation between edges and connections is very straightforward - each edge from the graphical model is represented by a connection in the CASL model.

The relationship between graphical nodes, CASL locations and CASL components is of a slightly more complicated nature. Path nodes are distinct from components, and their instances are processed differently for the purpose of CASL code generation. In CARMA, there is only one type of agent - the component. Every component has a store and behaviour, and these can be defined by the user with no restrictions. In

graphical scenarios, we can categorise these components into two classes: those who visit (mobile components) and those who are visited (nodes). In CARMA, both kinds are represented using the component template. When instantiated in this way, within each of the classes we can observe a set of common design patterns and features.

- Nodes have data structures for storing information about their (immutable) location (stored as the `location` type from Spatial CASL) as well as their current (or: minimum, maximum) occupancy. Their behavior involves managing the flow of mobile components; advertising themselves to potential visitors and registering incoming and outgoing components.
- Mobile components are usually the main subject of the analysis. Their inner mechanics may be very complex depending on the particular model. However, on the level of movement through space (which is what CGP is concerned with), they share a number of similarities. The internal knowledge must include their current (variable) location as well as the next, potential location, together with behaviour structures for making the choice about which location should be the next one. They should also know their start and goal location, for the purpose of implementing their ways of entering and exiting the system. Additionally, there needs to be a transition that updates the components' current and next location variables, in order to realize their movement through the nodes.

4.6.2 Nodes

In CGP, nodes are coloured, and components may be restricted to be able to visit only specific colours of nodes. Visiting a node from the perspective of a component means that the component will assume the same value of location attributes as the node.

As mentioned before, at the level of CASL code, nodes are just another kind of CARMA component. Their behaviour is automatically generated and provides a mechanism for mobile components to choose, visit, and leave a given node.

It is important to note that the use of the term 'node' in the context of CGP is distinct from the 'node' discussed in Chapter 3 as an additional element of Spatial CASL syntax. In Spatial CASL, *nodes* are specific *locations* belonging to the defined *universe*, which can be used in CASL expressions when specifying the model. In the context of CGP nodes are higher-level compund entities encoded in CASL with the use of

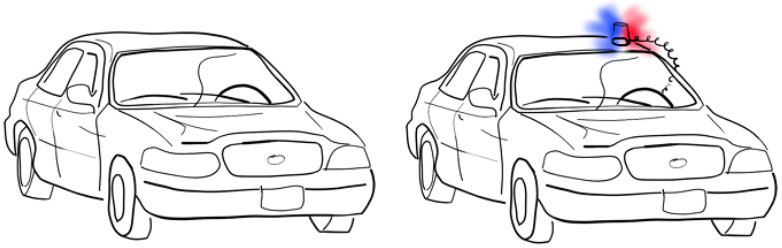


Figure 4.4: One real-world scenario in which the current state of a component changes its movement rules is when an unmarked police car becomes a privileged vehicle by turning on a siren.

components and *locations*.

4.6.3 Components

The user can specify a component type using structured input. The name and appearance of the component can be defined as well as the processes defined in the component, its allowable path and non-movement actions (see Fig. 4.5).

Once a component type has been defined, instances of that component type can then be placed within the graphical layout (by drag-and-drop). Component instances of the same type differ only in the values of their attributes, and therefore can be represented by identical symbols. Their placement on the grid determines their location attribute. The state of a component, given by the value of one of its attributes, can determine if that instance is allowed to move on a particular path.

For example, the user can specify a `Car` component with the states `NORMAL` and `PRIVILEGED`. This reflects a real-world scenario in which an unmarked police car (which most of the time follows the same rules as normal cars do) may become a privileged vehicle by turning on a siren (as illustrated in Fig. 4.4). In this example, instances of the `Car` component in the state `NORMAL` are able to travel along paths labelled `ROAD_LANE`, while instances in the state `PRIVILEGED` are able to use both `ROAD_LANE` and `BUS_LANE` paths. This mechanism applies to other scenarios from the area of urban planning, for example the movement of ambulance vehicles in their normal and privileged state. This particular case study has been modelled in CARMA and described in [47].

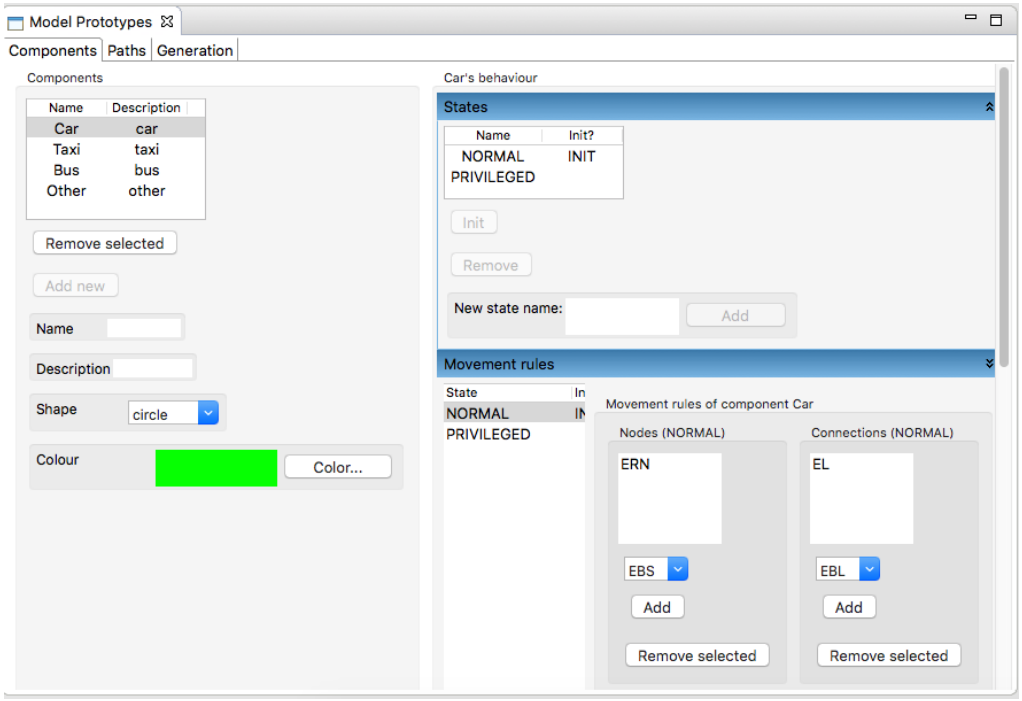


Figure 4.5: A screenshot of the graphical interface for defining component, node and edge prototypes.

4.7 Translating the graphical specification into CASL code

There is usually more than one possible way of specifying a particular model in CASL. In order to decide on a specific representation, one must answer the following questions:

- Which real-world elements are to be represented by constants?
- Which real-world elements are variables or functions?
- Which real-world elements are autonomous agents with internal states and behaviour?

For example, the node of a spatial network, as specified in the graphical input, can be translated into something as simple as an integer constant. Each component capable of visiting nodes would then need to have a variable field in their internal store, to keep the value of the node they are currently located at. If the component is only allowed to visit a subset of nodes, there needs to be an additional predicate on the movement action which ensures that the node value belongs to the set of values that the component

invoking the action is allowed to visit. This version of CASL representation was used in the first version of CGP.

This approach, however, has a number of limitations. Two important features, which the method described above is lacking, are first the ability of nodes to store data and second to control the number of incoming and outgoing visitors. Those features have proved to be a very useful mechanism in systems where traffic plays an important role. The strength of this approach is the ability of the incoming components to use the information shared by the node to either slow down their movement rate (which they might do, for example, in response to high congestion levels, simulating the slowing down of movement in traffic jams), or to realize a variety of routing behaviours, the simplest of which is to choose the least congested node among the available ones.

For this reason, in the CASL code generated by CGP from the graphical input, nodes are represented by CARMA components, rather than by any other simpler structure.

4.8 Example scenario

Examples of systems that can be defined in the CARMA graphical editor include networks of paths. Each path is specified by a directed graph. Nodes can belong to more than one graph. In this case, a component at a node may have a choice over the available paths, depending on the location, the type of the component, or the state of the instance, as explained above.

4.8.1 CARMA model representation

To represent a scenario defined in CGP using CASL code, the following CARMA components are created:

- Agent components
- Generator components
- Node components

Agent components represent the mobile agents in the system, specified through the CGP interface, in this case: `Bike`, `Car`, `Pedestrian` and `Rollerblader`.

For each agent component, a singleton *generator component* is created, having the name `<NameOfAgent>Generator`. It is responsible for handling the creation of new instances of a particular agent type in the system. In a trivial scenario, it triggers the instantiating of a single agent with a constant rate. In cases where a more complex behaviour is desired, the rate at which new agents are created may be defined using a non-trivial function whose parameters may include values of measures of the current state of the system. For example, if roads are highly congested with cars, more people may choose to travel by bike, and so the rate at which new `Bike` components are introduced to the system will increase.

For every type of node defined through the CGP interface, a *node component* is added to the CASL model. The basic purpose of node components is to provide the infrastructure for registering and unregistering visitors (agent components). In addition to that, in scenarios where agent components have to choose the next node to visit among those which are accessible through connections, the propensity of choosing one node over another can be adjusted using the value of the probability of the `choose` action. All nodes constantly advertise themselves, trying to synchronize with agents on the `choose` action in order to send out their location, which the agent can then move to. In simple scenarios, the probability of receiving a particular node's `choose` message may be constant – in which case all nodes are equally likely to be selected. In a more complex system, the probability is defined by a function, and may depend, for example, on the current value of occupancy at the available nodes, or the physical distance between the agent and the node.

Figure 4.6 is a schematic representation of eight stages of a typical CGP-generated system. In stage 1, the `BusGenerator` component performs three actions. The first two change only its internal state and store. They invoke a globally accessible function which randomly assigns two locations to be the start and goal location of a `Bus` component. Once this preparation is done, the `BusGenerator` component performs an action which results in a global update of the whole system – instantiating a new `Bus` component. At the same time, the `BusGenerator` component changes its internal state, to make sure that before instantiating another `Bus` component, the start and goal locations are going to be randomly determined once again.

During stage 2, the newly created `Bus` component performs its first action. It checks

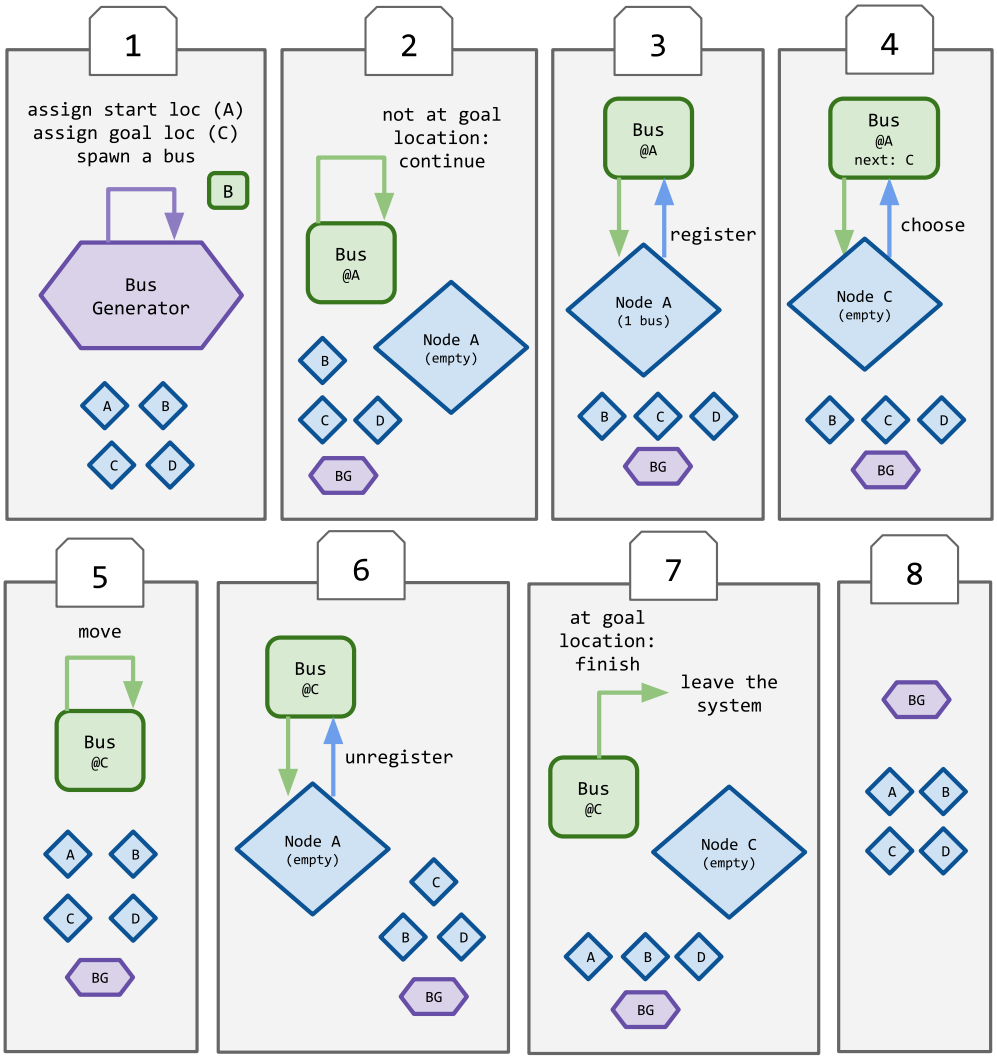


Figure 4.6: The evolution of a simple CGP system.

whether it happens to already be at its goal location. In this case it is not - and therefore it continues, changing its internal state to one in which it is able to perform the `register` action (stage 3).

4.8.1.1 Movement rules for connections

One example of a system with components that have movement constraints is an urban environment with four types of path-bounded components: `Bike`, `Car`, `Pedestrian` and `Rollerblader`, which move within the environment using paths of the following three types: `Pavement`, `Road`, `Cyclepath`.

Components access to these paths is shown in the table below:

Component:	Pavement	Road	Cyclepath
Bike	✓	✓	✓
Car		✓	
Pedestrian	✓		
Rollerblader	✓		✓

In this example, the ability of a component to move along a path segment of a specific type depends only on the type of the component, not its attribute values. There is only one type of node, and all components are allowed to visit them.

Listing 4.1: The generated CASL code for the Rollerblader component.

```
component Rollerblader(location start, location goal,
                       real startTime, process Z) {
  store {
    attrib location currentLoc = start;
    attrib location goalLoc = goal;
    attrib location nextLoc = start;
    attrib location previousLoc = start;
    attrib real startTime = startTime;
  }
  behaviour {

    ReadyToChoose =
      choose*
        [((nodeLoc.NODE) &&
          (edgeValues(currentLoc, Pavement, nodeLoc)
          || edgeValues(currentLoc, Cyclepath, nodeLoc)))]
        (nodeLoc){my.nextLoc = nodeLoc;}.ReadyToMove;

    ReadyToMove = move*[false]
      <my.currentLoc, my.nextLoc>
      {my.previousLoc = my.currentLoc;
           my.currentLoc = my.nextLoc;}.ReadyToUnregister;

    ReadyToUnregister = unregister[(my.currentLoc == nodeLoc)]
      (nodeLoc).ReadyToArrive;
```

```
    ReadyToArrive =
      [(my.currentLoc == my.goalLoc)] arrive*[false].kill
        +
      [(my.currentLoc != my.goalLoc)] continue*.
         ReadyToRegister;

    ReadyToRegister =
       register[(my.currentLoc == nodeLocation)]
      (nodeLocation).ReadyToChoose;
  }
  init {
    Z
  }
}
```

Listing 4.1 shows the CASL code for the definition of the `Rollerblader` component, having the appropriate movement rules. Figure 4.7 is a schematic depiction of the internal behaviour of each component with movement rules in a CGP system. These movement rules are realized through the five internal states generated automatically by CGP:

- `ReadyToArrive`
- `ReadyToRegister`
- `ReadyToChoose`
- `ReadyToMove`
- `ReadyToUnregister`

When a new component enters the system, it starts in the state `ReadyToArrive`.

4.8.1.2 Movement rules for nodes

In the previous example all nodes were of the same kind, `N`. We can now extend that scenario by introducing a new type of node (`BusStop`) as well as a new type of component (`Bus`) and connection (`BusLane`).

Below is the updated table showing the movement rules for every component:

Component:	Pavement	Road	Cyclepath	BusLane	N	BusStop
Bike	✓	✓	✓		✓	
Car		✓			✓	
Pedestrian	✓				✓	✓
Rollerblader	✓		✓		✓	
Bus	✓	✓		✓	✓	✓

All components are by default allowed to visit N type nodes, but now there is a special type of node, the BusStop, that only Bus and Pedestrian components can be located at.

4.8.1.3 Movement rules for component states

Finally, we can add another kind of movement rule to the system - one that depends on the current internal state of a component. The table below shows the movement rules for a Car component having two possible states: NORMAL and PRIVILEGED.

Component:	Pavement	Road	Cyclepath	BusLane	N	BusStop
Car (NORMAL)		✓			✓	
Car (PRIVILEGED)		✓		✓	✓	

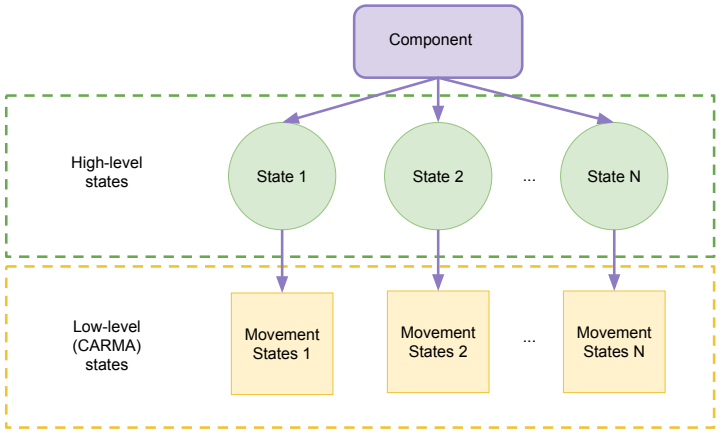


Figure 4.7: Component behaviour in CGP systems with movement rules.

So far we only discussed CARMA model states in the context of realizing the movement cycle. This creates just a single template-like CARMA model, of no immediate value in

terms of applying it to real world scenarios. In order to make this template useful, we also need to consider a new layer of states; high-level states, the transitions in and out of which may be completely independent of the transitions within the movement cycle. This gives us full flexibility with regard to what kind of moving component our system involves. Are they cars, carrying passengers, visiting petrol stations and looking for parking lots? Are they swarms of ants probing locations for pheromones left by their fellow workers? These types of behaviours can be specified at a higher level than the mechanics of movement through space, shared among those very different real-wold systems, which we described in the previous sections.

In any CARMA model, the behaviour is always comprised of a number of states and transitions between them; there is no explicit way of organizing the states into a hierarchical structure. In contrast, in the discussed scenarios the states can be conceptually organized into low-level (movement cycle) and high-level ones. This allows us to define systems in which high-level states, specified through the CGP interface, can be given different sets of movement rules.

The low-level states are only enough for components in which the internal sets of movement rules remain unchanged during the evolution of the system (see an example in Fig.4.6).

In more sophisticated cases, for example in the scenario of the police car that changes its state between `NORMAL` and `PRIVILEGED` (shown in Fig.4.4), more low-level states are needed to fully express the component's behaviour (both the states of the movement cycle as well as the states of the car). This is solved by creating a set of low-level states which realize the movement mechanics for each high-level state defined in CGP. In the case of the police car with two high-level states, the total number of low-level CARMA states generated for the component is ten.

The hierarchical structure of component behaviour in CGP is shown in Fig.4.7.

In the CARMA model, each high-level state is associated with a namespace that the low-level states can belong to. This is realized using a prefix with movement cycle state names (for example, `PRIVILEGED_ReadyToMove`). This allows for different movement behaviour in components with more than one high-level state.

Listing 4.2: The generated CASL code for the `ReadyToChoose` movement cycle state of the `Car` component.

```
NORMAL_ReadyToChoose =
```

```
  choose*[((nodeLoc.NODE) && (true in edgeValues(currentLoc,
     Road, nodeLoc)))]
    (nodeLoc){my.nextLoc = nodeLoc;}.NORMAL_ReadyToMove;

PRIVILEGED_ReadyToChoose =
  choose*[((nodeLoc.NODE) &&
    (true in edgeValues(currentLoc, Road, nodeLoc)
      || (true in edgeValues(currentLoc, BusLane, nodeLoc)))]
    (nodeLoc){my.nextLoc = nodeLoc;}.PRIVILEGED_ReadyToMove;
```

4.9 The CGP in use

In this chapter we have presented a newly-developed software tool which assists with the creation of CARMA models of systems in which location, movement and topology play a significant role. CAS by their nature are large-scale systems so concepts such as location, separation, distance and movement very often have roles to play in their models.

By concentrating on location and movement, our graphical modelling tool provides a convenient separation of concerns between the spatial aspects of a model (such as location, proximity and movement) and the dynamic aspects of a model (such as attribute and state update, communication, and synchronization). We believe that this separation can be helpful in allowing the modeller to focus their attention on particular aspects of the model in isolation.

One could argue that CGP is a strictly domain-specific tool, having evolved from models of urban scenarios involving mobile components traversing roads. However, we can abstract the movement and location specific features from these case studies, and find similar mechanisms in other, not directly related domains.

For example, in nature, ants often move along specific paths (path-bounded movement) designated by the trail of pheromones left by their fellow workers [31]. The intensity of pheromones influences the ant's likelihood of choosing a particular path. This is similar to a crowd routing scenario in which high congestion on a particular path may influence a pedestrian's likelihood of choosing that path over another. This scenario is also particularly interesting, because the stationary components (path nodes) are

dynamically created (when an exploring ant establishes a route to a food source) and removed (when the source of food is depleted), through the interaction with mobile-components.

A different scenario involves smart grids [46], distributed networks of electricity generation, storage and usage. Distance, topology, and the current state of the grid nodes all play an important role on the movement of energy in the system.

Our graphical model-generation tool handles all of the low-level aspects of location representation such as placement on a co-ordinate system and the consistent handling of co-ordinate values throughout the model. This level of detail is often tedious and error-prone to maintain manually so we believe that the model generation approach also benefits modellers here.

We published the preliminary version of CGP in [114]. We demonstrated the usage of CGP in the work published in [50], as described in detail in Chapters 5 and 7 of this thesis. This gave us insights into the dynamics of crowd routing, and provides some validation of the correctness of the transformation of our graphical design into running code. The CGP was also presented during the QUANTICOL Project Final Review in Lucca, Italy on 23rd of May 2017.

4.10 Potential for future work

CGP could be greatly improved by adding additional layers of graphical specification.

Mobile components form a hierarchical structure in which all elements extend one superclass. The superclass contains the implementation of the low-level mechanics of movement. Any component extending that class (for example, using UML-style class boxes and inheritance relation arrows) would therefore be a mobile component and have the capacity to define its own additional high-level behaviour. This behaviour, rather than being added at the level of generated CASL code may also be graphically laid out in the form of graphs representing Markovian processes. Going one step further, such a component could be extended again, adding more layers to the hierarchy of behavioural patterns. The process of generating CASL code would consist of "flattening out" the inheritance tree into a set of individual CARMA components with complicated, intricate behaviour that would be otherwise very difficult to develop manually.

Chapter 5

Modelling of pedestrian movement and simple crowd routing

> Walking into the crowd was like sinking into a stew – you became an ingredient, you took on a certain flavour.
>
> *Margaret Atwood,*
> *The Blind Assassin*

5.1 Introduction

In this chapter we explore the dynamics of pedestrian movement through a network of paths. This could be a specific part of a city, a pedestrianised network of lanes, or paths through a large park. The defining feature of our example is that there are essentially two groups of pedestrians that start on opposite sides of the network who wish to traverse the paths to get to the side opposite to where they started. This scenario could arise in a city where there are two train stations on opposite sides of the central business district serving the eastern and the western suburbs of the city, and a number of people who commute from the west work close to the east station and vice versa.

During rush hour in the morning and afternoon, people want to traverse the park or lanes as quickly as possible so that they are not late, and we wish to investigate what features enable these pedestrians to pass through the network efficiently. If there are multiple paths, it would seem in advance that it makes sense to use some paths for one direction and other paths for the other direction. This raises the question of what routing or information such as signs is sufficient for the two groups of pedestrians to separate out onto different paths.

This chapter presents an initial investigation into the modelling of this scenario, and we demonstrate how this can be achieved using the CARMA modelling language and its software tools: CARMA Eclipse Plugin and CARMA Graphical Plugin (see Chapters 2 and 4), considering different possibilities for the network.

The model presented in this chapter is a simple and preliminary conception which we extend further in Chapters 7 and 8.

5.2 Automatic code generation

The CARMA Graphical Plugin (CGP), described in detail in Chapter 4 of this thesis, was designed for the purpose of working with models of CAS and allows the user to specify the structure of movement in a model by laying out graphical symbols on a plane [114]. The editor generates CARMA code from the graph which the user has defined. In addition to normal attributes, such as the identifier and start time, CARMA components which are defined in this way have a set of distinguished attributes to specify their current location in space.

Each CARMA component in the model may have its mobility restricted to a given set of paths through the graph defined by the user. Paths in this context are subgraphs of the user-defined graph consisting of a set of uniform vertices connected by directed, coloured edges. At any given time in the system's evolution, the location attributes of a component instance must be equal to the location of one of the nodes belonging to the subgraph where that component is restricted. A component can change its location

attributes only if there exists a path from its current node to the new node, and if this path belongs to the subgraph where the mobility of this component is restricted. This is realised through providing a separate `ExistsPath` function for each subgraph.

In the work described in this chapter we used a version of CGP where nodes are not represented by CARMA components, they are identifiable through the location variables of a mobile component instead. The topology of the network is encoded within the `ExistsPath` function that accepts two location parameters (each encoded by a pair of integer coordinates) and returns a boolean value. If there exists a connection from the first to the second location, the function returns `true`, otherwise it returns `false`.

Component actions query these functions during the execution of their predicates, and can modify the component's location attributes accordingly, in the update block. For each node which can be accessed by a particular component type, a movement action must be included in the component's behaviour. If the node can be accessed by a component in more than one state, the action must be specified separately for each state.

In systems with complex mobility restriction graphs, topology-defining functions, as well as component behaviour blocks may require a large number of cases in the definition. This function definition is automatically generated by the CGP, freeing the modeller from the task of manually producing this CGP model code.

5.3 Basic Pedestrian model

5.3.1 Overview

The CARMA model is illustrated in Figures 5.1 and 5.2. It assumes that there are two types of pedestrians, A and B, and that P and Q are variables of type pedestrian.

The two classes of pedestrians, A and B, reflect the desired direction in which a particular pedestrian would like to move. In our simple model, all pedestrians starting at the location A want to get to location B and vice versa. This is a considerably simplified representation, however in its general properties it is not unlike a situation common in large cities, in which at any given location we abstract the flow of crowd to a 1-dimensional movement, distinguishing only between pedestrians moving to and from the city centre, and ignoring all the other directions.

The two *Arrival* components generate pedestrians at two different locations (on opposite sides of the graph), and the pedestrians move from their origin side to the opposite side. Once a pedestrian has reached its goal, the count for that type of pedestrian is incremented and the time taken for traversal is added to the total time so that the average traversal time can be calculated for each pedestrian type.

The model is parameterised by a number of functions that capture the graph information and are generated automatically as described above.

- **ExistsPath**(P,x,y,i,j) is a Boolean function that determines if an edge exists between a pedestrian's current position and another node in the graph, hence a $\mathsf{move}^{\star}_{ij}$ action can only occur when such an edge exists.

- **AtGoal**(P,x,y) is a Boolean function that checks if the pedestrian has reached its goal, hence the finish action $\mathsf{fin}^{\star}$ can only occur once the destination has been reached. After this the pedestrian does not move any more.

- **ArrivalRate**(P) determines the arrival rate for each type of pedestrian.

- $\mathbf{Start}_x(P)$ and $\mathbf{Start}_y(P)$ define the initial location of a new pedestrian depending on its type.

A function that is not directly related to the graph structure is **MoveRate**$(P,x,y,i,j,\ldots)$ which determines the rate of movement along a particular edge, and can take additional parameters which can affect this rate such as the current count of other pedestrians of

the same or different type. We use the following definition that uses the numbers of pedestrians of the other type at the target node to reduce the movement rate.

$$\textbf{MoveRate}(P,x,y,i,j,A_{ij},B_{ij}) = \begin{cases} move_A/(B_{ij}+1) & \text{if } P = A \\ move_B/(A_{ij}+1) & \text{if } P = B \end{cases}$$

where A_{ij} are the number of A pedestrians at the target node and B_{ij} are the number of B pedestrians at the target node, and $move_Q$ is a basic movement rate for each pedestrian type. The values of A_{ij} and B_{ij} can be used in this context as they can be accessed within the `environment` block's scope in CASL by using the `measure` expressions to determine the number of *Pedestrians* having particular locations assigned to their current and next location store variables. The definitions of rates for each action are also defined in the `environment` block, which is where the values of A_{ij} and B_{ij} are ultimately used.

Store of *Pedestrian* **component:**

P	pedestrian type — an enumeration with values A and B
x	current x coordinate
y	current y coordinate
stime	time of arrival

Behaviour of *Pedestrian* **component:**

$$Ped \stackrel{\text{def}}{=} \sum_{(i,j)\in V} \left[\textbf{ExistsPath}(P,x,y,i,j)\right]\left(\mathsf{move}^{\star}_{ij}[\bot]\langle\rangle\{\mathsf{my}.x \leftarrow i, \mathsf{my}.y \leftarrow j\}.Ped\right) + \left[\textbf{AtGoal}(P,x,y)\right]\left(\mathsf{fin}^{\star}[\bot]\langle\rangle.\textbf{nil}\right)$$

Initial state of *Pedestrian* **component:** Ped

Store of *Arrival* **component:**

P	pedestrian type

Behaviour of *Arrival* **component:**

$$Arr \stackrel{\text{def}}{=} \mathsf{arrive}^{\star}[\bot]\langle\rangle.Arr$$

Initial state of *Arrival* **component:** Arr

Figure 5.1: The *Pedestrian* and *Arrival* components

Constants:

V	set of coordinate pairs representing nodes in the graph

Measures:

$average_P$	average time for traversal by pedestrians of type P

Global store:

$count_P$	number of P pedestrians to complete traversal
$total_P$	total time for all completed P pedestrian traversals

Evolution rule functions:

$$\mu_p(\gamma_s,\gamma_r,\alpha) = 1$$

$$\mu_w(\gamma_s,\gamma_r,\alpha) = 1$$

$$\mu_r(\gamma_s,\alpha) = \begin{cases} \mathbf{ArrivalRate}\big(\gamma_s(P)\big) & \text{if } \alpha = \mathsf{arrive}^\star \\ \mathbf{MoveRate}\big(\gamma_s(P),\gamma_s(x),\gamma_s(y),i,j,\ldots\big) & \text{if } \alpha = \mathsf{move}^\star_{ij} \\ \lambda_{fast} & \text{otherwise} \end{cases}$$

$$\mu_u(\gamma_s,\alpha) = \begin{cases} \{\}, \big(Pedestrian, \{P \leftarrow \gamma_s(P), x \leftarrow \mathbf{Start}_x(\gamma_s(P)), y \leftarrow \mathbf{Start}_y(\gamma_s(P)), stime \leftarrow \mathsf{now}\}\big) & \\ \qquad\qquad \text{if } \alpha = \mathsf{arrive}^\star & \\ \big\{count_{\gamma_s(P)} \leftarrow count_{\gamma_s(P)} + 1, total_{\gamma_s(P)} \leftarrow total_{\gamma_s(P)} + (\mathsf{now} - \gamma_s(stime))\big\}, 0 & \\ \qquad\qquad \text{if } \alpha = \mathsf{fin}^\star & \\ \{\}, 0 & \text{otherwise} \end{cases}$$

Collective:

$$PedAB \stackrel{\text{def}}{=} \big(Arrival, \{P \mapsto A\}\big) \parallel \big(Arrival, \{P \mapsto B\}\big)$$

Figure 5.2: Environment and collective

Figure 5.2 specifies the four functions $(\mu_p,\mu_w,\mu_r,\mu_u)$ known as the evaluation context. Probabilities and weights on activities are not used in this model so the μ_p and μ_w functions are trivially constant functions.

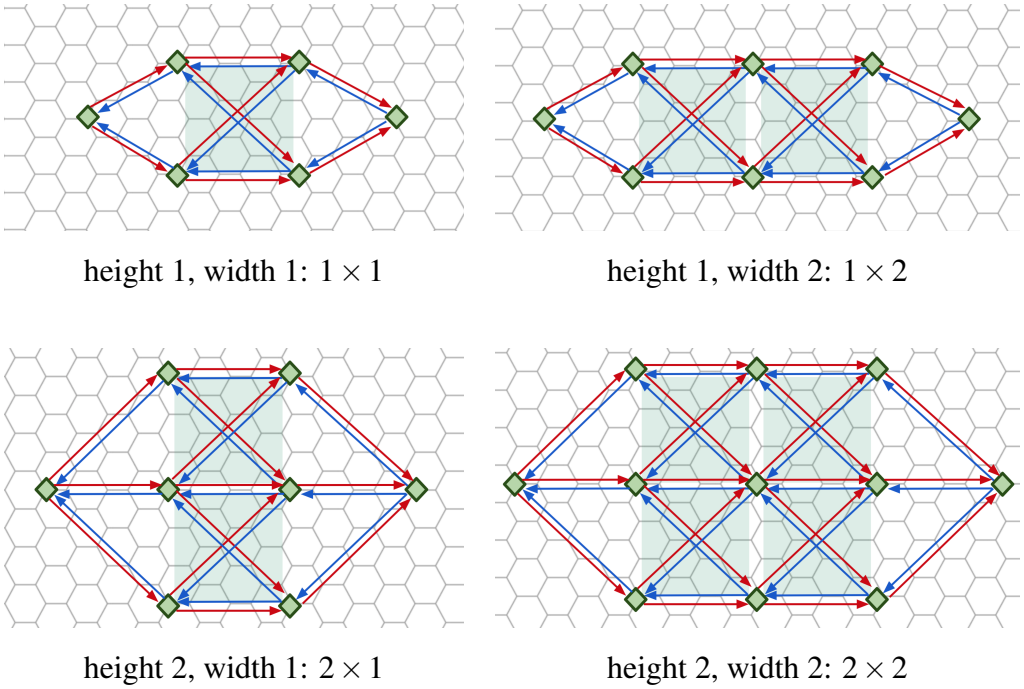


Figure 5.3: Four model instances of increasing size and complexity: height indicates the number of crossbar elements from top to bottom and width the number of crossbar elements from left to right.

5.3.2 Model instances

Four instances of this CARMA model are shown in Figure 5.3.

These show instances of the general CARMA model (the topology is the same as in the previous section) with increasing size and shape complexity and the same rates for traversing an edge. It is important to note that the central repeating features of the path network are the cross-bars in the centre of the network. In the simplest instance we have only one cross-bar and we describe this instance as having height 1 and width 1, representing it as instance 1×1. The height of an instance is the number of cross-bar elements from top to bottom, and the width is the number of cross-bar elements from left to right. As the cross-bar structure is repeated we have instances 1×2 (which has height 1 and width 2), 2×1 (which has height 2 and width 1), and 2×2, depending where the additional structure is added into the network.

An increase in the width of the network has the obvious consequence that journeys

across the network take longer. An increase in the height of the network has the consequence that pedestrians are offered an increased choice of routes, with the implicit consequence that individual paths are less congested (because there are more of them on offer). The edges are equally long and thus the time to traverse them is the same under comparable conditions.

In each instance of the network of paths there are two sub-networks which restrain the movement of the pedestrians of type *A* and type *B*. *Pedestrians* of type *A* are restricted to the red sub-network and must cross the network from left to right. *Pedestrians* of type *B* are restricted to the blue sub-network and must cross the network from right to left. The networks illustrated in Figure 5.3 are symmetric but this is of no particular significance and it would pose no difficulty to work with networks which were not symmetric, which we will show with the Meadows example later in this thesis (see Chapter 7).

These graphs were drawn in the CGP and CARMA code was generated from it, including all necessary instances of the **ExistsPath**, **AtGoal**, and **ArrivalRate** functions and applications of these in predicate guards on processes.

5.3.3 Limitations

It is important to remember that the model presented in this chapter is a highly simplified abstraction of a speculative scenario and does not represent any specific real-life system.

For example, it assumes that only two types of agents exist, one for each direction of travel. There is only one start and one goal location for each of these agent types. At any node that offers a choice as to which node to move towards next, the potential choices are equally likely to be triggered (with the exception of the updated model presented in Section 5.4.2, where congestion information influences this probability). This is untrue of real-life systems, as one can imagine a large number of possible factors that could influence such decision, for example the relative location of the next

node with respect to the goal node, or the length of the connection.

The four instances of the model explored in this chapter are highly regular and symmetrical, and all the connections are assumed to be of equal length. It is also important to note that the overlapping of the connections has no effect on the movement and traffic, and such crossings are not considered as true junctions in the scope of this model, but rather as pairs of connections completely separated from each other.

5.3.4 Analysis and results

We analysed our CARMA model using the CARMA Eclipse Plugin (see Chapter 2).

We designed a suite of experiments to explore the behaviour of the model. To provide a baseline for average travel time we investigated the travel time in the presence of only one type of pedestrian (thereby giving a model which has no congestion). Thereafter we investigated the models with congestion in the presence or absence of pedestrian routing.

We adapted a very simple routing mechanism in which at the start node all *Pedestrians* always choose the left most node as their next destination. All decisions taken at further nodes were not affected. This simple routing technique is similar to an attempt at decreasing pedestrian congestion seen in many real-world systems, where signs are placed next to busy pedestrian lanes with directions such as 'Keep to the left'. It is important to note that this is a simple and naive mechanism - every single *Pedestrian* unfailingly follows the guideline at the first node – so all *Pedestrians* travelling in the same direction end up travelling on the same connection towards their second node, at which point they all 'forget' the keep-to-the-left rule and start behaving exactly the same way as they do, for all nodes, in the scenario with no routing at all. This means that the connections in the middle of the structure (not connected to the start node) are still likely to have a number of pedestrians travelling each way, contributing to congestion.

When routing is present, only one starting route has a non-zero rate, and the non-zero rate is assigned in order to direct pedestrians away from each other.

Model	Nodes	Connections	LoC
1x1	6	8	208
1x2	8	12	248
1x3	10	16	288
2x1	8	13	258
2x2	11	20	328
2x3	14	27	398
3x1	10	18	308
3x2	14	28	408
3x3	18	38	508

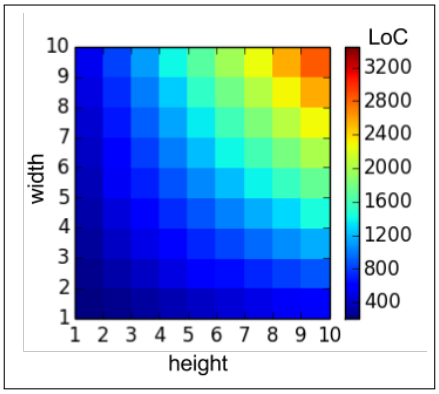


Figure 5.4: The number of lines of CARMA code per model structure. Left, for small values of width and height. Right, for larger values of width and height.

We used the CGP to automatically create CARMA code models. Fig. 5.4 shows how the number of lines of CARMA code grows with model structure complexity.

The results from our experiments are presented in Figure 5.5. We have three results (no congestion, routing, and no routing) for each of the four model instances considered (1×1, 1×2, 2×1, and 2×2).

An inspection of the results shows that, unsurprisingly, for any structure the best average travel times are obtained when there is no congestion in the network. As anticipated, networks with greater height have lower average travel times because they have greater capacity, due to the inclusion of additional routes (thus 2×1 results are better than 1×1 results, and 2×2 results are better than 1×2 results).

Finally, we see that routing is always advantageous, especially so in the case of narrow networks where congestion in experienced most (i.e. in the 1×1 structure and the 1×2 structure).

5.4 Modelling congestion influence from microscopic and macroscopic perspectives

5.4.1 Overview

This section investigates another approach to modelling the movement of pedestrians through networks of paths by taking an approach to this spatial modelling which combines certain aspects of *microscopic* and *macroscopic* modelling.

Modelling approaches can be classified as *microscopic*, *mesoscopic* and *macroscopic*. In the case of microscopic models, each pedestrian is modelled individually, and is typically located in two-dimensional space. This can be done through agent-based models, cellular automata, magnetic force models or social force models [23, 56, 80]. For cellular automata, discrete two-dimensional space (a grid or lattice) is used, whereas the other approaches consider continuous space.

In contrast, macroscopic models consider densities of pedestrians rather than individ-

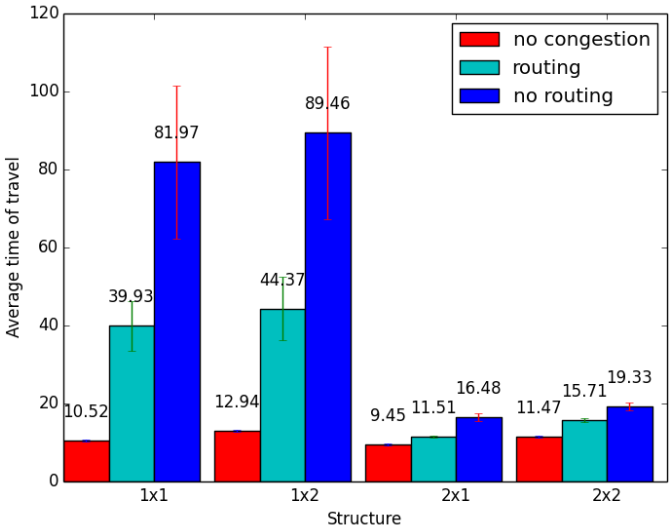


Figure 5.5: Average travel time results from the experiments on structure and network usage.

uals at specific locations in space. They are typically defined by partial differential equations, allowing for the continuous variation of time as well as location in space.

The difference between the microscopic and macroscopic perspective is therefore usually considered in the context of how important for the analysis it is to track and distinguish between the local knowledge and behaviour of each agent *individually*, versus the collective influence they have on the performance of the system.

One can combine these two distinct ways of looking at systems in a number of different ways. In [37], the authors present an extension of Generalised Stochastic Petri Nets (GSPN) with 'tagged tokens'. In GSPN, tokens are indistinguishable, so it is not possible to track a specific one during the evolution of the system (a macroscopic approach). Adding tags to one or more tokens allows for microscopic analysis of its position and movement through the net. For example, Bellomo and Bellouquid define a multi-scale model, where micro-scale interactions are described [22]. These lead to a kinetic model from which macroscopic equations can be obtained by considering asymptotic limits. This means that the macroscopic description emerges from the microscopic interactions rather than being defined *a priori*. An alternative use is where physical and logical groups of pedestrians are used within the simulation [98].

This combined approach is a middle ground between considering a discrete (microscopic) model of individual movement in two-dimensional space or a continuous (macroscopic) model based on partial differential equations (PDEs) where both change over time and change over space are described by continuous quantities. Another approach to modelling the dynamic interaction of populations, such as species of molecules in biology, is the mean field/fluid approximation technique that assumes that populations are well-mixed and makes no spatial distinctions. This would not be useful for the type of pedestrian modelling considered here which is inherently spatial. This is because the techniques mentioned above are based on high level abstraction, which, albeit providing good optimization for large scale simulations, removes the possibility of examining the behaviour of individual agents – the decisions they make at each fork of the communication network. In our model, the behaviour at the level of individual is being influenced by the observation of a subgroup of the collective.

In other words, in our approach, the model components which represent pedestrians act as individual decision-making entities when they make probabilistic choices about their future movement and function as a collective population mass when we consider how they impede the movement of other pedestrians. Neither of these views of model components (as individual entities or as population mass) is dominant and the model needs to make use of both views to represent how congestion arises along paths and how it impedes the progress of individuals. Together the two views allow us to represent *intelligent density-dependent movement* which captures the behaviour of decision-making individuals adapting to continuously-changing information about the collective to which they belong.

What distinguishes this model from the one presented in the previous section is the ability of a pedestrian to choose what they do to next. In contrast to the model presented in the previous sections, the *Pedestrian* component now has an explicit choice which results in assigning a next node to a variable in its store.

To summarize, we consider the *Pedestrian* components from the *microscopic* perspective when modelling their decision-making capabilities, while at the same time looking at them from the *macroscopic* perspective when modelling the influence their movements through the network has on other pedestrians, i.e. the congestion.

Our modelling approach is reminiscent of approaches described by the term *mesoscopic* in the field of modelling of molecular phenomena, in particular, diffusion of molecules in three-dimensional space. In this domain, microscopic and macroscopic are used as above. In the former, individual molecules are modelled in three-dimensional space, including their collisions; and in the latter, PDEs are used to describe changes in density of the molecular species involved. An alternative approach is to assume well-mixedness and use stochastic simulation or ODEs based on the Chemical Master Equation (CME) to describe the changes in quantities of chemical species but this does not take space into account. The mesoscopic approach involves dividing three-dimensional space into areas of volume known as voxels, and the CME (or suitable approximations) assume that within each voxel, species are well-mixed and then can be used to express this interaction. However, this approach takes into account the fact that some molecules will move between voxels, and determines this based on a

Reaction-Diffusion Master Equation (RDME) [38, 39].

We take a similar approach, although without defining a master equation. We consider pedestrians moving along paths but we do not model the individual movement of each pedestrian as they are moving along the path, only as they move from one path segment to another. Their movement within a path segment is determined by an exponentially-distributed duration and this duration is state-dependent, is the sense that it is dependent on the number of pedestrians on that path segment.

In our focus on pedestrian movement along a network of paths, we consider that each pedestrian has only local information about the number of pedestrians on path segments, and they wish to get to their destination as efficiently as possible. An obvious approach to take when presented with the choice of two paths (that both lead to the destination) is to take the one with the least oncoming pedestrian traffic, although in very crowded situations, considering the pedestrian traffic flow in their current direction of travel may be important as well. The network of paths could be a specific part of a city, a pedestrianised network of lanes, or paths through a large park.

As with our earlier model, there are two groups of pedestrians starting on opposite sides of the network who wish to traverse the paths to get to the other side (opposite to where they started). Consider again the scenario arising in a city where there are two train stations on opposite sides of the central business district serving the eastern and the western suburbs of the city, and a number of people who commute from the west work close to the east station and vice versa. During rush hour in the morning or evening, people want to traverse the park or lanes as efficiently as possible, and we wish to investigate what features of the network of paths help towards this goal. If there are multiple paths, it would seem in advance that it makes sense to use some paths for one direction and other paths for the other direction. This raises the question of what routing information – such as signage – is sufficient for the two groups of pedestrians to separate out onto different paths. Even for this first basic model our experiments demonstrate that this is not as straightforward as one may at first think.

5.4.2 The updated model

Our CARMA model of pedestrian movement is presented in Figures 5.6 and 5.7. There are still two types of pedestrian, *A* and *B*, and we again have *P* and *Q* are variables of type pedestrian. The two *Generator* components generate pedestrians at two different locations (on opposite sides of the graph), and the pedestrians move from their origin side to the opposite side. *Pedestrian* behaviour is split into two alternating processes. When at a node pedestrians are in the state *Choose*. After the next edge has been chosen, pedestrians go to the state *Move*. Once a pedestrian has reached its goal, the count for that type of pedestrian is incremented ($count_P$ in Figure 5.7) and the time taken for traversal is added to the total time so that the average traversal time can be calculated for each *Pedestrian* type. These variables are global and form part of the environment.

Store of *Pedestrian* component:

P	pedestrian type
ℓ	current location
$n\ell$	next location
stime	time of arrival

Behaviour of *Pedestrian* component:

$$Choose \stackrel{\text{def}}{=} \sum_{k \in V} \left[\mathbf{ExistsPath}(P,\ell,k)\right] \left(\mathsf{choosePath}^{\star}_{k}[\bot]\langle\rangle\{\mathsf{my}.n\ell \leftarrow k\}.Move\right) + \left[\mathbf{AtGoal}(P,\ell)\right] \left(\mathsf{fin}^{\star}[\bot]\langle\rangle.\mathbf{nil}\right)$$

$$Move \stackrel{\text{def}}{=} \left(\mathsf{move}^{\star}[\bot]\langle\rangle\{\mathsf{my}.\ell \leftarrow \mathsf{my}.n\ell\}.Choose\right)$$

Initial state of *Pedestrian* component: *Choose*

Store of *Generator* component:

P	pedestrian type

Behaviour of *Generator* component:

$$Arr \stackrel{\text{def}}{=} \mathsf{arrive}^{\star}[\bot]\langle\rangle.Arr$$

Initial state of *Generator* component: *Arr*

Figure 5.6: The *Pedestrian* and *Generator* components of the basic model

Constants:

V	set of coordinate pairs representing nodes in the graph
λ_P	arrival rate for pedestrians of type P
$move_P$	movement rate for pedestrians of type P

Measures:

$average_P$	average time for traversal by pedestrians of type P

Global store:

$count_P$	number of P pedestrians that have completed the traversal
$total_P$	total time for all completed P pedestrian traversals

Evaluation context:

$$\mu_p(\gamma_s,\gamma_r,\alpha) = 1$$

$$\mu_w(\gamma_s,\gamma_r,\alpha) = 1$$

$$\mu_r(\gamma_s,\alpha) = \begin{cases} \mathbf{ArrivalRate}\big(\gamma_s(P)\big) & \text{if } \alpha = \mathsf{arrive}^\star \\ \mathbf{MoveRate}\big(\gamma_s(P),\gamma_s(\ell),k,A_{\ell,k},B_{\ell,k}\big) & \text{if } \alpha = \mathsf{move}_k^\star \\ \mathbf{ChooseRate}\big(\gamma_s(P),\gamma_s(\ell),k,A_{\ell,k},B_{\ell,k}\big) & \text{if } \alpha = \mathsf{choosePath}_k^\star \\ \lambda_{\mathit{fast}} & \text{otherwise} \end{cases}$$

$$\mu_u(\gamma_s,\alpha) = \begin{cases} \{\}, \big(\mathit{Pedestrian}, \{P \leftarrow \gamma_s(P), \ell \leftarrow \mathbf{Start}(\gamma_s(P)), \\ \quad \mathit{stime} \leftarrow \mathsf{now}\}\big) & \\ & \text{if } \alpha = \mathsf{arrive}^\star \\ \big\{\mathit{count}_{\gamma_s(P)} \leftarrow \mathit{count}_{\gamma_s(P)} + 1, \mathit{total}_{\gamma_s(P)} \leftarrow \mathit{total}_{\gamma_s(P)} + (\mathsf{now} - \gamma_s(\mathit{stime}))\big\}, 0 & \\ & \text{if } \alpha = \mathsf{fin}^\star \\ \{\}, 0 & \text{otherwise} \end{cases}$$

Collective:

$$\mathit{PedAB} \stackrel{\text{def}}{=} \big(\mathit{Generator}, \{P \mapsto A\}\big) \parallel \big(\mathit{Generator}, \{P \mapsto B\}\big)$$

Figure 5.7: Environment and collective of the basic model

- **ExistsPath**(P,ℓ,k) is a Boolean function that determines if an edge exists between a pedestrian's current position ℓ and another node k, hence a $\mathsf{move}^{\star}_{k}$ action can only occur when such an edge exists.

- **AtGoal**(P,ℓ) is a Boolean function that checks if the pedestrian has reached its goal, hence the finish action $\mathsf{fin}^{\star}$ can only occur once the destination has been reached. After this the pedestrian does not move any more.

- **ArrivalRate**(P) is a function that returns λ_P, the arrival rate for pedestrians of type P. These rates are defined as constants in Figure 5.7.

- **Start**(P) defines the initial location of a new pedestrian depending on its type.

A function that is not directly related to the graph structure is **MoveRate**$(P,\ell,k,A_{\ell,k},B_{\ell,k})$ which determines the rate of movement along a particular edge (ℓ,k), and can take additional parameters that can affect this rate such as the current count of other pedestrians of the same or different type. We use the following definition that uses the numbers of pedestrians of the other type on the current edge to reduce the movement rate.

$$\textbf{MoveRate}(P,\ell,k,A_{\ell,k},B_{\ell,k}) = \begin{cases} move_A/(B_{\ell,k}+1) & \text{if } P = A \\ move_B/(A_{\ell,k}+1) & \text{if } P = B \end{cases}$$

where $A_{\ell,k}$ is the number of A pedestrians on the edge and $B_{\ell,k}$ is the number of B pedestrians on the edge, and $move_P$ is a basic movement rate for each pedestrian type.

Another function is **ChooseRate**$(P,\ell,k,A_{\ell,k},B_{\ell,k})$ which determines the rate (in effect, the probability) of choosing a particular edge (ℓ,k) for the next move. Similar to the definition of **MoveRate**, our definition takes into account the number of pedestrians of the other type on that edge, such that pedestrians will favour edges with lower traffic.

$$\textbf{ChooseRate}(P,\ell,k,A_{\ell,k},B_{\ell,k}) = \begin{cases} fast/(B_{\ell,k}+1) & \text{if } P = A \\ fast/(A_{\ell,k}+1) & \text{if } P = B \end{cases}$$

where $A_{\ell,k}$ is the number of A pedestrians on the edge and $B_{\ell,k}$ is the number of B pedestrians on the edge. It is assumed that decisions are made relatively quickly, thus we use a constant factor $fast \geq 100$ to make $\mathsf{choosePath}^{\star}_{\mathsf{k}}$ a fast action.

We have chosen to work with these functions as they meet our expectations of the behaviour of pedestrians in these circumstances.

Figure 5.7 specifies the four functions $(\mu_p, \mu_w, \mu_r, \mu_u)$ known as the evaluation context. Probabilities and weights on activities are not used in this model so the μ_p and μ_w functions are trivially constant functions.

As mentioned above, the model is a combination of the microscopic and macroscopic perspective. Each edge in the graph represents a path segment, and the path segment that each pedestrian is on is determined by their current and next position. The model only records that they are on a specific path segment and there is no explicit knowledge about where on the path segment a pedestrian is, only a time duration for traversing the segment. This is an abstraction that allows for efficient modelling as it is not necessary to know the specific location when traversing a path segment.

5.4.3 Design of experiments

The instances of the model are identical with regards to the topology of the network, and presented in Figure 5.3.

The two main parameters which influence the behaviour of the model are the movement rates and arrival rates. Since the movement rates depend on congestion, it was important to set plausible base rates for the non-congested case. Therefore, parameters were calibrated under the assumption that all links are equidistant (approximately 100m long) and using the results of a study conducted by [57], which suggests an expected pedestrian speed of $1.34 m/s$ under normal conditions.

Having set base rates for pedestrian movement, then the focus of the experiments was

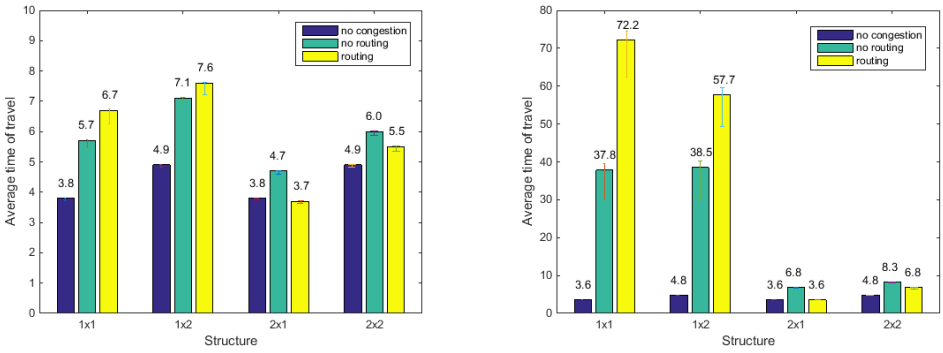
to investigate traffic under different arrival rates, i.e. different numbers of pedestrians in the network. In a first experiment arrival rates are low, which should lead pedestrians to pass through the network smoothly. In a second experiment arrival rates are doubled, which we expect to cause more congestion in the network and thus longer average travel times.

So far, arrival rates of both pedestrian groups had been equal. Therefore, we conducted a third experiment, in which we explored how the arrival rate of pedestrian type A affects the average travel time of B. In order to do that, the arrival rate of A was set to $s * arrivalRate_B$, $s \in \{1,2,3,4,5\}$.

5.4.4 Analysis

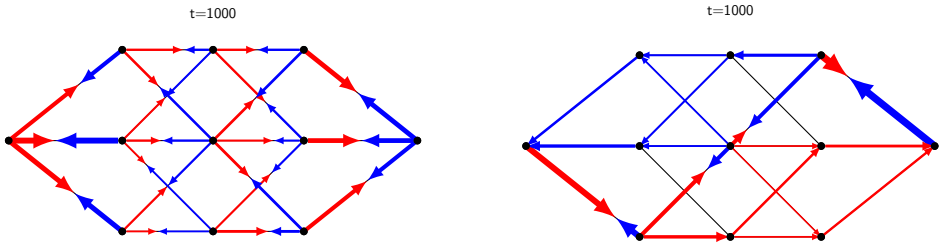
We are able to obtain values for the length of time it takes each pedestrian to traverse the whole system, even though our approach abstracts away from the details of where each pedestrian is on a path segment. We can then compare on average how long it takes different types of pedestrian to move from their entrance point to their exit point for different scenarios.

The results from our first two experiments are presented in Figure 5.8. For each experiment we have three results (no congestion, routing, and no routing) for each of the four model instances considered (1×1, 1×2, 2×1, and 2×2). Figure 5.8a shows the results for low overall traffic. An inspection of the results shows that, unsurprisingly, for any structure the best average travel times are obtained when there is no congestion in the network. As anticipated, networks with greater height have shorter average travel times because they have greater capacity, due to the inclusion of additional routes (thus 2×1 results are better than 1×1 results, and 2×2 results are better than 1×2 results). In these networks routing is advantageous, whereas when applying routing to narrow (low height) networks, the opposite behaviour is observed, in fact routing leads to an increase in the average travel time. Similar results were obtained for the case of high traffic, which are displayed in Figure 5.8b. Here, where average travel times in narrow networks are already extremely high in the absence of routing, they increase



(a) Low traffic (b) Higher traffic

Figure 5.8: Average travel time results from the experiments on structure and network usage



(a) Without routing (b) With routing

Figure 5.9: Congestion on network edges at time $t = 1000$ for the 2×2 topology

even more when routing is present. The observation that routing increases travel time is contradictory to our expectations and this is discussed further below.

Congestion in the 2×2 network is illustrated in Figures 5.9a and 5.9b. Red and blue arrows indicate direction of movement of the two pedestrian types. The overall number of pedestrians on a particular edge is represented by line width, while the proportion of A and B on an edge is represented by arrow length. When there is no routing present, congestion especially occurs near start and end nodes where all paths connect. All other edges (the ones closer to the center of the network) are less congested, since traffic is equally split over all possible paths. In comparison to that, when routing is enabled, pedestrian groups are guided away from each other, such that most edges are only used by one pedestrian group. Congestion occurs only near start and end points

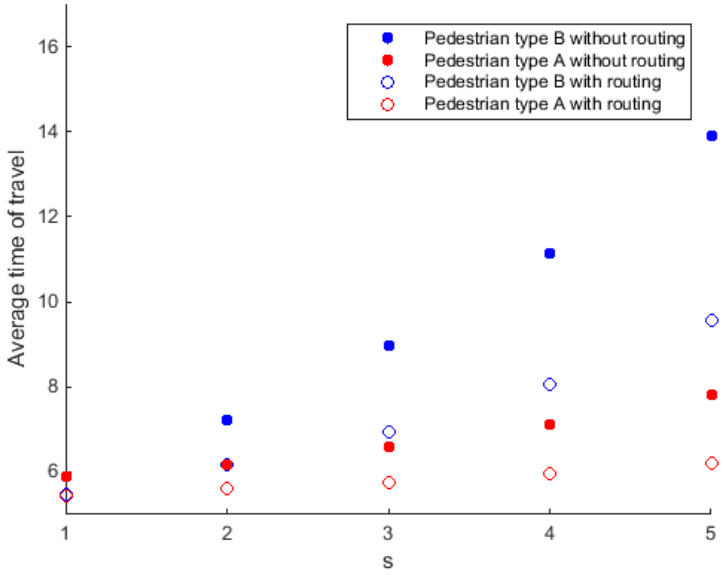


Figure 5.10: Average travel time of B against arrival rate of A

and on the main diagonal, as not all individuals choose the path with lowest traffic.

The results of our third experiment on the 2×2 network are shown in Figure 5.10. It can be seen that an increase in the amount of pedestrian type A leads to a significant slowdown of pedestrian type B, whereas the average travel time of type A rises only slightly. Using pedestrian routing helps to reduce average travel time in all cases.

5.4.5 Discussion

As mentioned above, some of the results we obtained were counter-intuitive, and differed from our expectations. In fact, it turns out that our expectations were incorrect as we had not fully appreciated one particular aspect of our model. Namely, that the functions that determine which path to choose and how fast movement is possible along a path are dependent on local information. In the case of movement speed, it is reasonable just to use the quantity of other pedestrians to determine this. However, for the choice of path, using only the local information of the paths that can be chosen imposes a notion of visibility onto the model, which can also be used to infer the topography of the landscape over which the paths are defined.

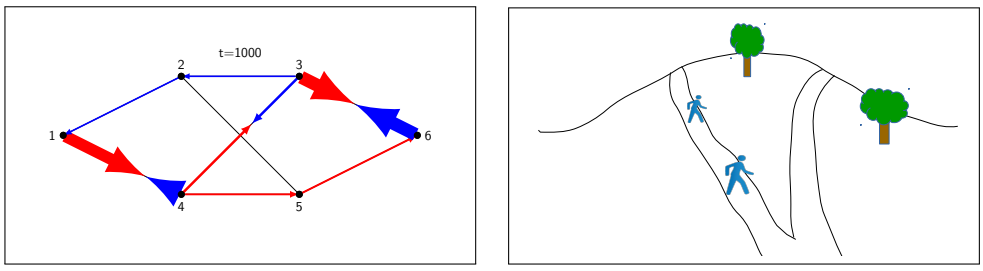


Figure 5.11: Congestion of edges for 1×1 network (left) and illustration of path visibility and how it may affect choice (right)

Consider the 1×1 model as given in Figure 5.11, a pedestrian entering from the right (node 6) in the routing case, will be directed along the upper branch (to node 3) and one entering from the left (node 1) will be directed along the upper branch (to node 4). This pedestrian at node 4 will then have a choice between the upper (to node 3) or lower branch (to node 5), both of which will appear reasonable because there will be little oncoming traffic on either. In fact, on the lower path there will be none, because of the routing applied to pedestrians to the right, but on the other path, traffic will also be low, so some proportion of pedestrians will take the upper path to node 3 and then take a very long time to reach node 6 because they will be facing the full flow of *all* pedestrians coming in other direction, and their average time to traverse the last link increases, as does the average time for the pedestrians that are traversing their first link. Hence routing causes some pedestrians to take a path that looks good but in practice is not (node 4 to 3). This does not occur to the same extent in the $2 \times n$ cases because even though a similar poor choice can be made, it will have a less pronounced effect.

The reason why this occurs is that looking at only the next paths ahead does not give complete information. This situation is reflective of network routing mechanisms where each node has only local information (for example, network of queues with local balancing, described in [67]).

This situation arises when pedestrians have limited visibility on the paths ahead. To obtain better pedestrian flow, there are a number of solutions including signposting at the point where the poor choice is made, and allowing path look-ahead of more than one path segment so that congestion further ahead can be seen. Another way to consider this is to view the paths as being laid over a landscape that is hilly – the poor choice is made because the hilly landscape prevents a pedestrian from seeing further

ahead as illustrated in Figure 5.11.

These results differ from those obtained from the model described in the first part of this chapter and in the paper [112], where we did not observe this counter-intuitive behaviour. The explanation for this is the fact that our model now uses path congestion to determine behaviour rather than node congestion as in the prior work. The change was made to reflect human behaviour better; however, it also gave us an improved understanding of model behaviour, and a more general model could then be developed which captures explicit notions of visibility.

The model and our results demonstrate that modelling which combines the microscopic and macroscopic perspectives is possible for this pedestrian movement scenario. Moreover, by abstracting from the specific location a pedestrian is at a path segment, we ensure that the simulation of the model is feasible, both in terms of model size and time required for simulation.

5.5 Conclusions

We have demonstrated a simple model of pedestrian movement over a number of different graphs, to illustrate the modelling of spatial aspects of CAS. The CGP allowed us to automatically generate the CARMA code for different networks which simplified the task, and allowed our pedestrian components to be generic in nature. Our initial experiments have considered situations with and without congestion as well as with and without explicit routing of pedestrians as they enter the network.

From the work presented in this chapter we learnt that even a simple system in which movement in physical space plays an important role may require a complex and large CASL code representation. The tools presented in Chapters 3 and 4 proved remarkably valuable for the purpose of encoding these spatial networks as well as automatic code generation, which ratifies our confidence in their usefulness and versatility.

5.6 Future work

There are many directions for future work. For example, another group of pedestrians could also be introduced, namely, tourists, and the focus would be on efficient traversal of the network during afternoon rush hours when commuters want to get home quickly and tourists wish to sightsee, and hence move slowly.

We are also interested in identifying when the model shows emergent behaviour, in the sense that different groups of pedestrians use different paths through the network in response to environmental cues such as information about congestion or routing suggestions (rather than explicit routing).

Following this work, we investigated another case study, analysing a model of crowd behaviour based on real-world data. It is described in detail in Chapter 7 of this thesis.

Chapter 6

Data

If you torture the data long enough, it will confess.

Ronald Coase

6.1 Introduction

In the work described later in Chapters 7 and 8, we follow the data-driven approach.

In particular, there are two ways in which data was used:

- To be compared with the results of model simulation,
- To create accurate representations of systems during the process of modelling.

The main strength of the data-driven approach lies in the fact that any tool or model created or evaluated using data is likely to produce meaningful results that provide insight into the real-world scenario which is being modelled. In addition to that, working with

data requires a high level of attention to detail and forethought with regard to computational complexity and quality of the available data; the undertaken research must deal with problems such as data cleaning and filtering, as well as optimization of the model size and complexity. Without taking data into account, one runs the risk of producing models which have no straightforward application to any real-world problems. Even when artificial models are created to exemplify the performance of the analysis tool, it is no guarantee that the particular kind of data needed for the simulation is available or will ever be. Because of all this, using real-world data remained a strong motivation throughout the process of producing the research comprising this thesis.

6.2 The Meadows city park bike counter

The Meadows is a large public city park located in Edinburgh, Scotland. Most of the park area consists of open grassland, with tree-lined paths crossing it in various directions. Some of these paths are to be shared between pedestrians and cyclists; others have designated cycle lanes. The Middle Meadow Walk is a wide path connecting the south of the park (adjacent to Argyle Place) with the three-way intersection of Teviot Place, Forrest Road and Lauriston Place streets. Middle Meadow Walk has a cycle lane, and is not available for motorized vehicles, except for privileged ones. It is an important route for cyclists leading from the south of the city towards the city centre.

The bike counter [13] (see Fig. 6.3) is located in the northern part of the Middle Meadow Walk, slightly to the north of where Simpsons Lane connects to it. It collects data 24/7, and increases its counter every time a bike passes it in either direction. The current and historical data gathered from the device can be accessed at [10].

The City of Edinburgh Council has published an open dataset of the bike counts collected from this device on their website [9]. This dataset differentiates between counts of cyclists travelling in the north and south directions, giving an indication of the intensity and direction of traffic at different times of the day.

[1] https://i.rcahms.gov.uk/canmore/d/SC00760475.jpg

Figure 6.1: An aerial view of the Meadows City Park in Edinburgh [1].

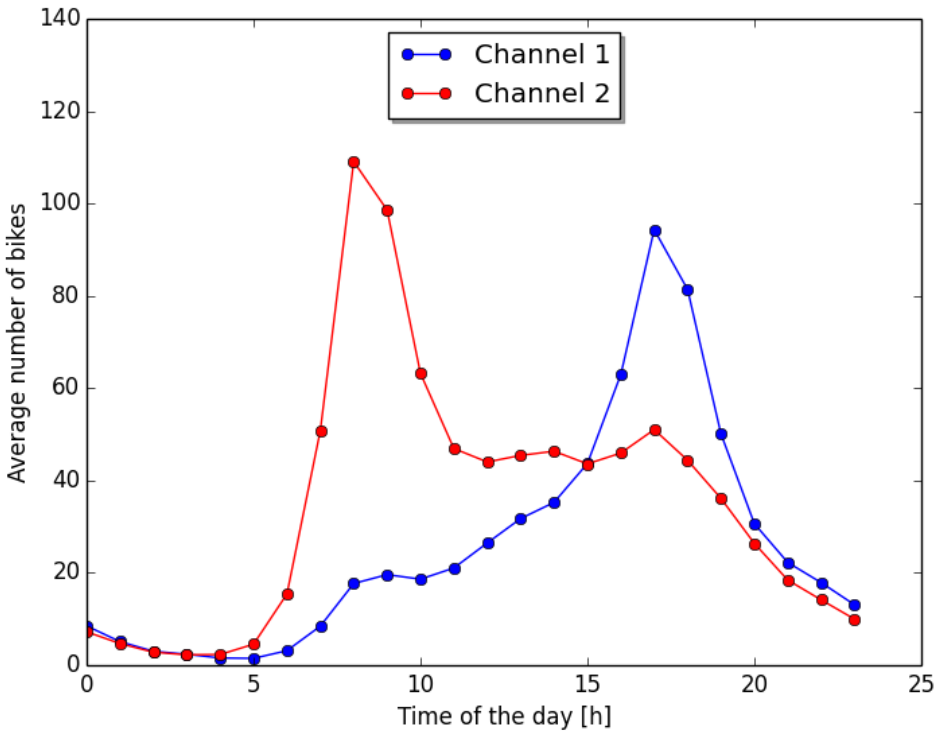


Figure 6.2: The data available at the City of Edinburgh Council Website [9] shows how the traffic of bicycles changes at different times of the day for bikes travelling towards: Channel 1–the south (from city centre) and Channel 2–the north (towards city centre).

Figure 6.3: The bike counter device located by The Middle Meadow Walk in Edinburgh. Photographs downloaded from [13].

By analysing this data (see Fig. 6.2) we can approximate the rush hours on an average day. The value of counted bikes peaks around 8:00 - 9:00 am in the morning, and then again in the afternoon at 5:00 - 6:00 pm, in the opposite direction.

6.3 Google Maps

Google Maps is a geographic mapping service provided by Google [12].

The Google Maps system uses latitude and longitude values in accordance with the World Geodic System 1984 (known also under the names: WGS84 and EPSG:4326), established in 1984 and last revised in 2004 [3].

WGS84 assumes all geographic data points lie on the surface of an oblate spheroid. Each point can be uniquely referenced by its latitude and longitude values. Because latitude and longitude are expressed as angles [2], we decided to convert them using the Universal Transverse Mercator (UTM) projection [91] in order to obtain the x and y values in the context of a cartesian coordinate system on a plane. This conversion is made for the reason of optimisation: it is less computationally complex to compute distances between points in cartesian coordinate system than it is in the WGS84 [8].

[2]Latitude is the angle between the equatorial plane and the line that passes through the point. Longitude is the angle between the reference meridian to another meridian that passes through the point.

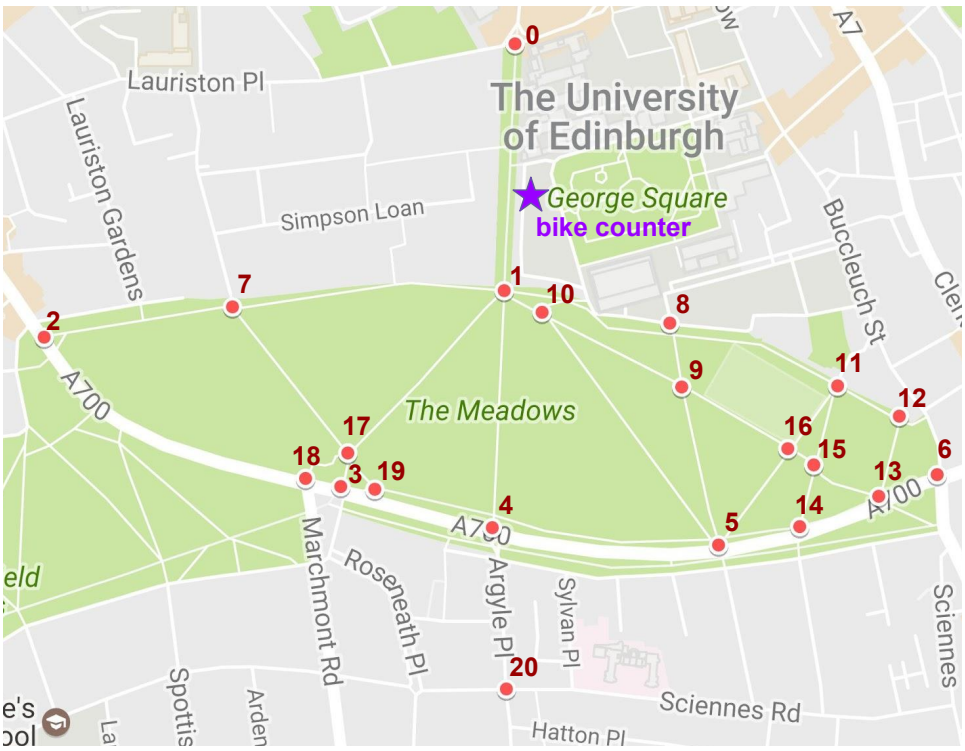


Figure 6.4: The Meadows public park in Edinburgh. We used Google Maps for obtaining the latitude and longitude of selected points.

The UTM coordinate system assigns to each point two values: easting and northing (expressed in the unit of metres, and referred to as 'x' and 'y' in this thesis), as well as its 2-dimensional Zone Number (see Fig. 6.5). The values of easting and northing coordinates are unique within each zone, but not globally unique - for this reason one needs to be extra cautious when dealing with systems geographically located in more than one zone. In the case studies presented in this thesis, we used geographical data from the UTM Zone 30U exclusively.

The translation from WGS84 coordinates to UTM coordinates was made at the time of data collection as part of data pre-processing.

Figure 6.5: The UTM Zone Grid in Europe [3].

6.4 Transport for Edinburgh

The Transport for Edinburgh company [18] has provided us with access to the REST API that can be used to obtain data gathered from their bus and tram system. The data used for the purpose of the research described in this thesis is relevant to the routes, timetables and live vehicle locations of bus services provided by Lothian Buses [14], a bus operator within the Transport for Edinburgh company. The data consists of JSON-formatted text files which can be requested from the following endpoints:

- **stops** (the latest information on bus and trams stops served by Transport for Edinburgh [18], including fields such as name of the stop, available services, destinations, and its geographical location)

- **services** (the information on each available service's name, destination, routes,

[3] https://commons.wikimedia.org/wiki/File:LA2-Europe-UTM-zones.png

and stops)

- **timetables** (the full timetable information per stop, including the list of services with corresponding departure times)

- **journeys** (the list of expected stops and corresponding departure times for a given service leaving from a particular start stop at a particular time and arriving at a particular destination point)

- **stop-to-stop timetables** (timetables in the form of journeys between two stops which share a route, for a requested time)

- **service status** (up-to-the-minute information about disruptions affecting services in real time)

- **live vehicle locations** (real time information on the position of currently active vehicles)

6.5 Tom Tom Traffic Flow

Tom Tom Traffic Flow [16] is a service provided by the Tom Tom company, dedicated to providing traffic information in a number of large cities around the world. The service offers live traffic data as well as historic traffic data gathered from Tom Tom GPS devices in vehicles on roads.

The traffic data is available in the following three formats:

- Live Traffic Level: a measure of traffic level per time, expressed as a percentage, and described on the service website [16] [17] as 'Indication of the current severity of traffic congestion on monitored roads in the city area compared to the normal expected congestion level.'

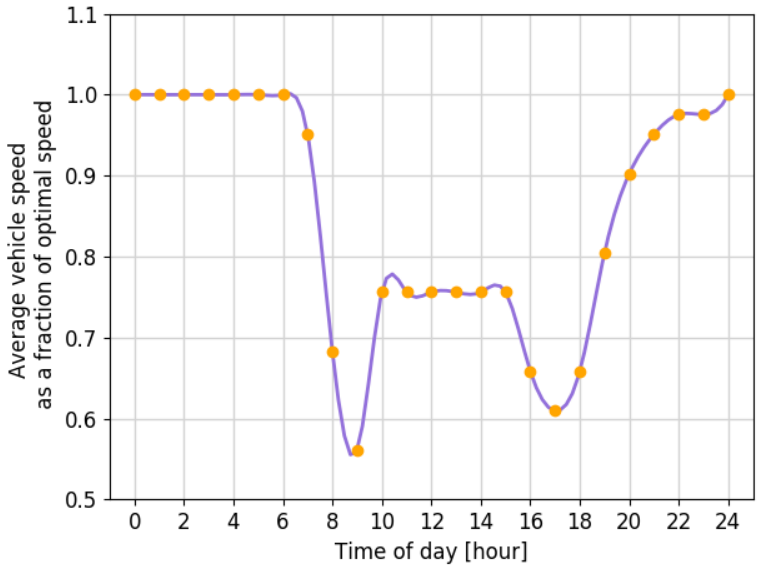


Figure 6.6: The average speed of a vehicle in The City of Edinburgh, expressed as a fraction of the optimal speed. The data shows a congestion peak at 9 am and a smaller congestion peak at 5 pm. On that particular day, the optimal vehicle speed was equal to 25.5 mph (41 km/h), represented by 1.0 on the graph above.

- Live Traffic Speed: a measure of the average vehicle speed in the unit of km/h, per time. The service also provides a value of 'optimal speed', that is the average speed of vehicles when no traffic is present.

- Live Traffic Reports: provides information on the causes of increased congestion, classified into three categories: roadworks, jams and closures.

In our model, we used the data from the Live Traffic Speed service, as presented in Fig. 6.6. To reduce the likelihood that an incident (roadwork, closure) not included in the simulation would affect the results, we used data from a day in which all the causes of increased congestion were categorised as "jams" in the Live Traffic Reports service.

Other existing traffic APIs which could have been used for the purpose of obtaining the information about congestion include Google Traffic [12] (an extension of Google Maps that displays traffic conditions in real time) and HERE Traffic [19] (a traffic information provider for commercial purposes). Google Traffic is free but does not

provide historical information – only real time data, whereas HERE Traffic does provide such historical data but it is a paid product. For these reasons Tom Tom Traffic Flow was chosen as the source of the traffic information used in the work presented in this thesis.

6.6 Conclusions

In this chapter we learnt about the sources of data which are used to create and analyse the models described in the later chapters of this thesis.

The data from the Meadows city park bike counter was involved in the work presented in Chapter 7. We used the information about the cycle traffic trends observed from the data collected by this device in order to form assumptions about the traffic of pedestrians in the park.

We sampled specific locations in the Meadows city park using the Google Maps API in order to obtain their geographical coordinates (latitude and longitude). These were translated into planar coordinates and constituted the nodes of the spatial graph forming the basis of the model presented in Chapter 7. The spatial locations of these nodes were also used to compute distances between each pair of connected nodes, and these distances were factored into the rates of the movement actions of the CARMA components in the model (in order to represent the fact that a longer path takes more time to traverse).

The model presented in Chapter 8 was heavily based on the data obtained from the Transport for Edinburgh company. In the process of creating the model, we used the information about the spatial locations of the bus stops, the sequences of stops within routes as well as the live vehicle location data (this data was collected over a period of five days). Tom Tom Traffic Flow historical congestion data was used in the model in Chapter 8 in order to introduce a realistic estimate of the traffic conditions that are present in the real-world system.

Chapter 7

Data-driven modelling of pedestrian movement

> A straight line may be the shortest distance between two points, but it is by no means the most interesting.
>
> *The Third Doctor*

7.1 Introduction

The case study we present in this chapter provides a model of an existing network of pedestrian paths and cyclepaths in the Meadows public park in Edinburgh, Scotland.

The work presented in this chapter builds upon what we described earlier in Chapter 5 and provides a foundation for further work presented in Chapter 8.

In order to include real data in the analysis, we re-designed the model from Chapter 5 so that it is based on a real-world scenario. It now uses CGP-like node representation style in which each node in the spatial network is represented by a CARMA component

- this made the model more concise and allowed us to capture more types of measures to the system, as the component nodes were able to keep information in their store (such as the number of *Pedestrians* currently at a given node, whereas in the basic model we could only directly measure the number of pedestrians at an edge). This scenario involves a more complex spatial network from the ones discussed in Chapter 5. Perhaps the most interesting difference here is that in the basic model *Pedestrians* only move in one general direction (from left-most node to the right-most node and vice versa). For a *Pedestrian* at any given node, regardless of which node ends up being chosen as the one to move to next, the realisation of this movement step will always result in bringing the *Pedestrian* closer to the goal. This is no longer true in the case of the model presented in this chapter. The *Pedestrians* now have the ability to choose any node connected to the current one, even when moving to the new node would result in the *Pedestrian* being further from the goal node than before. This means that potentially in this model it could happen that some *Pedestrians* never manage to reach their goal and are therefore condemned to wander the paths of the Meadows forever (it is however very unlikely as the distance to the goal takes part in influencing the probability of a *Pedestrian* choosing one node over another). This is impossible, by design, in the basic model from Chapter 5.

The aim of modelling this scenario was to explore the extent of expressiveness of the CARMA language, rather than to compare the model's predictions to data. This is because data suitable for such a comparison are not available.

Unlike the basic models described in Chapter 5, we based this case study on real-world geographic data. We used the Google Maps API to obtain the location of 20 path lane intersections in The Meadows area (see Fig. 6.4) and used these to generate the structure of connections and nodes in the model.

In the models presented in Chapter 5 the start and finish (goal) location of each pedestrian is predetermined by the graph representing the network of the paths. In the more complex example of the Meadows, multiple start and finish locations are allowed and all connections are bidirectional. For this reason, in order to mimic the behaviour of the real system, in which commuters cross the park with the intention to get from one place to another, each *Pedestrian* is assigned a start and finish location when they first

appear in the modelled system. The probability of choosing a given location as a start or a finish (goal) location for a given component depends on the time of the day to reflect the data published by the City of Edinburgh, obtained from the bike counter (see Chapter 6). We used a sum of Gaussian functions to represent these traffic changes throughout the day, which is a reasonable fit to the available data.

Store of *PathNode* component:

nodeL	location of node
arrived	number of pedestrians that have arrived at this node as a goal
timeSum	sum of times taken by pedestrians arriving at this node as a goal

Behaviour of *PathNode* component:

$$Advert \stackrel{\text{def}}{=} \mathsf{choose}^{\star}[\top]\langle \mathsf{my}.nodeL\rangle.Advert$$

$$Arrive \stackrel{\text{def}}{=} \mathsf{arrive}^{\star}[\mathsf{my}.nodeL = arrL](arrL, startTime)\ \{\mathsf{my}.arrived \leftarrow \mathsf{my}.arrived + 1, timeSum := timeSum + (now - startTime)\}.Arrive$$

$$StartAssign \stackrel{\text{def}}{=} \mathsf{assignStart}^{\star}[\bot]\langle \mathsf{my}.nodeL\rangle.StartAssign$$

$$GoalAssign \stackrel{\text{def}}{=} \mathsf{assignGoal}^{\star}[\bot]\langle \mathsf{my}.nodeL\rangle.GoalAssign$$

Initial state of *PathNode* component: $Advert \mid Arrival \mid StartAssign \mid GoalAssign$

Figure 7.1: The *PathNode* component of the Meadows model

Store of *Pedestrian* component:

startL	start location
goalL	goal location
currL	current location
nextL	next location
prevL	previous location
stime	time of arrival

Behaviour of *Pedestrian* component:

$$
\begin{aligned}
\mathit{Choose} &\stackrel{\text{def}}{=} \mathsf{choose}^{\star}[\ell \text{ in } \mathsf{my}.\mathit{currL}.\mathsf{post}](\ell)\{\mathsf{my}.\mathit{nextL} \leftarrow \ell\}.\mathit{Move} \\
\mathit{Move} &\stackrel{\text{def}}{=} \mathsf{move}^{\star}[\bot]\langle \mathsf{my}.\mathit{currL}, \mathsf{my}.\mathit{nextL}\rangle\{\mathsf{my}.\mathit{prevL} \leftarrow \mathsf{my}.\mathit{currL}, \mathsf{my}.\mathit{currL} \leftarrow \mathsf{my}.\mathit{nextL}\}.\mathit{Arrive} \\
\mathit{Arrive} &\stackrel{\text{def}}{=} [\mathsf{my}.\mathit{currL} = \mathsf{my}.\mathit{goalL}]\mathsf{arrive}[\top]\langle \mathsf{my}.\mathit{goalL}, \mathsf{my}.\mathit{stime}\rangle.\mathbf{kill} + \\
&\qquad \mathit{continue}^{\star}[\bot]\langle\rangle.\mathit{Choose}
\end{aligned}
$$

Initial state of *Pedestrian* component: *Arrive*

Store of *Generator* component:

startL	start location
goalL	goal location

Behaviour of *Generator* component:

$$
\begin{aligned}
\mathit{AssignStart} &\stackrel{\text{def}}{=} \mathsf{assignStart}^{\star}[\top](\mathit{start})\{\mathsf{my}.\mathit{startL} \leftarrow \mathit{start}\}.\mathit{AssignGoal} \\
\mathit{AssignGoal} &\stackrel{\text{def}}{=} \mathsf{assignGoal}^{\star}[\top](\mathit{goal})\{\mathsf{my}.\mathit{goalL} \leftarrow \mathit{goal}\}.\mathit{Spawn} \\
\mathit{Spawn} &\stackrel{\text{def}}{=} \mathsf{spawn}^{\star}[\bot]\langle\rangle.\mathit{AssignStart}
\end{aligned}
$$

Initial state of *Generator* component: *AssignStart*

Figure 7.2: The *Pedestrian* and *Generator* components of the Meadows model

Space:

Nodes	set of coordinate pairs representing nodes in the graph
Connections	set of node pairs representing edges in the graph
Distances	a two-dimensional lookup table of pre-computed distances between pairs of nodes

Measures:

$\mathit{traffic}(n_1, n_2)$	the number of pedestrians whose current location is n_1 and next location is n_2
$\mathit{arrived}(n)$	the number of pedestrians who finished their route at the node n

Evaluation context:

$$\mu_p(\gamma_s, \gamma_r, \alpha) = \begin{cases} \mathbf{Pr_{goal}}\big(\gamma_s(\mathit{nodeL}), \mathit{now}\big) & \text{if } \alpha = \mathsf{assignGoal}^\star \\ \mathbf{Pr_{start}}\big(\gamma_s(\mathit{nodeL}), \mathit{now}\big) & \text{if } \alpha = \mathsf{assignStart}^\star \\ \mathbf{Pr_{choose}}\big(\gamma_s(\mathit{nodeL}), \gamma_r(\mathit{currL}), \gamma_r(\mathit{nextL}).\mathit{goal}, & \\ \qquad \mathit{traffic}(\gamma_r(\mathit{nextL}), \gamma_r(\mathit{currL}))\big) & \text{if } \alpha = \mathsf{choose}^\star \\ \mathbf{1} & \text{otherwise} \end{cases}$$

$$\mu_w(\gamma_s, \gamma_r, \alpha) = 1$$

$$\mu_r(\gamma_s, \alpha) = \begin{cases} \mathbf{Rate_{spawn}}\big(\mathit{now}\big) & \text{if } \alpha = \mathsf{spawn}^\star \\ \mathbf{Rate_{move}}\big(\gamma_s(\mathit{currL}), \gamma_s(\mathit{nextL}), & \\ \qquad \mathit{traffic}(\gamma_s(\mathit{nextL}), \gamma_s(\mathit{currL}))\big) & \text{if } \alpha = \mathsf{choose}^\star \\ \lambda_{\mathit{fast}} & \text{otherwise} \end{cases}$$

$$\mu_u(\gamma_s, \alpha) = \begin{cases} \emptyset, \big(\mathit{Pedestrian}, \{\mathit{startL} \leftarrow \gamma_s(\mathit{startL}), \mathit{goalL} \leftarrow \gamma_s(\mathit{goalL}), \mathit{stime} \leftarrow \mathsf{now}\}\big) & \\ \qquad \text{if } \alpha = \mathsf{spawn}^\star & \\ \{\}, 0 \quad \text{otherwise} & \end{cases}$$

Initial collective:

$$\mathit{Meadows} \stackrel{\text{def}}{=} \big(\mathit{Generator}, \emptyset\big) \parallel \big(\mathit{PathNode}, \{\, \mathit{nodeL} \mapsto \mathit{Nodes}_1 \,\}\big) \parallel \ldots \parallel \big(\mathit{PathNode}, \{\, \mathit{nodeL} \mapsto \mathit{Nodes}_p \,\}\big)$$

Figure 7.3: Environment and evaluation context of the Meadows model

The definitions of the components are shown in Fig. 7.1 and 7.2. The definition of the environment, evaluation context, and the collective of the CARMA model are shown in Fig. 7.3.

In particular, the evolution rule involves a number of functions that are described in the next subsections and shown in Fig. 7.4, 7.6, 7.7, 7.8.

7.2 Spawning new Pedestrians

7.2.1 Spawn rate

The rate at which new *Pedestrians* arrive into the modelled system depends on the current time of the day, to reflect the changing intensity of traffic. The real-world data from the city bike counter shows a trend of increasing bike traffic in the rush hours (Fig. 6.2). The average bike traffic increases in one direction in the morning hours, and in the opposite direction in the late afternoon, this presumably being the result of people travelling to and from the city centre for work. It also shows a small increase around midday (lunchtime).

$$\mathbf{Rate_{spawn}}(t) = (\mathcal{N}(t,\mu_1,\sigma^2) + \mathcal{N}(t,\mu_2,\sigma^2) + \mathcal{N}(t,\mu_3,\sigma^2) + 0.1) \cdot C,$$

$$\text{where: } \mathcal{N}(t,\mu,\sigma^2) = \frac{1}{\sigma\sqrt{2\pi\sigma^2}} e^{\frac{(t-\mu)}{2\sigma^2}},$$
$$\sigma = 60\text{min}, \mu_1 = 540\text{min}, \mu_2 = 750\text{min}, \mu_3 = 1020\text{min}, C = 400.0.$$

Figure 7.4: The rate at which new components arrive in the system depends on the hour of the day. The three values of the mean of the Gaussian function μ_1,μ_2, μ_3, represent three time points of the day: 9 am, 12.30 am and 5 pm, respectively, in the unit of minutes. σ is the standard deviation of the Gaussian function with mean μ_i. C is a dimensionless scaling factor, introduced in order to obtain a realistic spawn rate value.

In our model we used a traffic intensity pattern loosely based on these observations from the real-world data. The function we used is a sum of three Gaussian functions having the standard deviation of 1 hour and the means at 9:00 am, 12:30 pm, and 5:00 pm, respectively. The reason for using this simplified function of traffic intensity for the first version of the model, over one that fit more accurately to the data, is that it made the influence of the traffic intensity on the results of the simulation more visible and easy to control. The spawn rate formula is shown in Fig. 7.4 and Fig. 7.5 shows the original data from the bike counter as well as the spawn rate function.

7.2.2 Spawn locations

Examination of the real-world data from the city bike counter suggests that new bikes arriving into the modelled system should not be uniformly distributed at all possible entry points, at all times of the day. In the morning rush hours, cyclists are much more likely to be travelling in one direction and in the afternoon rush hour, the opposite. The bike counter, from which this data has been obtained, is located in the north part of The Middle Meadow Walk, the widest path through the Meadows, connecting the south-most and the north-most parts of the park (see Fig. 6.4). There is no available

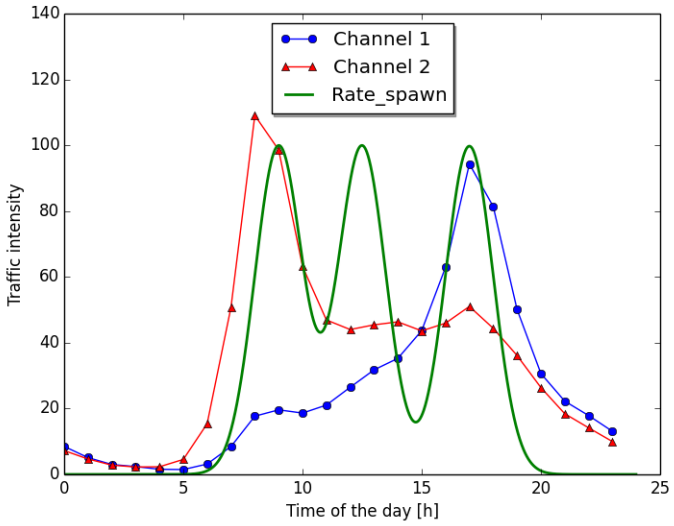


Figure 7.5: This graph shows the data from the bike counter, discussed in detail in section 6.2. In addition, the $Rate_{spawn}$ function (See Fig. 7.6) is shown in green.

data about the traffic changes on all of the other paths in the park, however based on everyday observations, we have assumed the two extreme points of our model of The Middle Meadow Walk to be the most likely entry points to the modelled system for the *Pedestrian* components during the rush hours. In our model, every time a new *Pedestrian* component is introduced to the modelled system, it is assigned a start and a goal location. New *Pedestrian* components are instantiated as a result of the Generator component performing the spawn* action, which uses the start and goal location values saved in the store of the Generator component. At each location there is a *PathNode* component, which attempts to perform two actions assignGoal* and assignStart*, in order to communicate with the Generator component and update the values of the start and goal location in its store with the *PathNode*'s location. The probability of receiving this message (μ_p, see Chapter 2) can vary for each location, and the resulting value of start or goal location is determined by the *PathNode* component that manages to communicate with the Generator component first. As a result, multiple *PathNodes* compete with one another in order to decide which one will become the start or the

$$\mathbf{Pr}_{goal}(n_{loc},t) = \begin{cases} \mathcal{N}(t,\mu_1,\sigma^2)\cdot C + 1.0 & \text{if } n_{loc} = 0, \\ \mathcal{N}(t,\mu_3,\sigma^2)\cdot C + 1.0 & \text{if } n_{loc} = 20, \\ 1.0 & \text{if } n_{loc} \in [2,3,5,6,7,8], \\ 0.0 & \text{otherwise.} \end{cases}$$

$$\mathbf{Pr}_{start}(n_{loc},t) = \begin{cases} \mathcal{N}(t,\mu_1,\sigma^2)\cdot C + 1.0 & \text{if } n_{loc} = 20, \\ \mathcal{N}(t,\mu_3,\sigma^2)\cdot C + 1.0 & \text{if } n_{loc} = 0, \\ 1.0 & \text{if } n_{loc} \in [2,3,5,6,7,8], \\ 0.0 & \text{otherwise.} \end{cases}$$

$$\text{where: } \mathcal{N}(t,\mu,\sigma^2) = \frac{1}{\sigma\sqrt{2\pi\sigma^2}} e^{\frac{(t-\mu)}{2\sigma^2}},$$
$$\sigma = 60\text{min}, \mu_1 = 540\text{min}, \mu_3 = 1020\text{min}, C = 10.0.$$

Figure 7.6: The two values of the mean of the Gaussian function μ_1, μ_3, represent two time points of the day: 9 am and 5 pm, respectively, in the unit of minutes. σ is the standard deviation of the Gaussian function. C is a dimensionless scaling factor, introduced in order to obtain a realistic propensity value.

$$\mathbf{Rate_{move}}(n_{from}, n_{to}, \mathrm{Trfc}(n_{to}, n_{from})) = \begin{cases} \dfrac{1}{Dstn(n_{from}, n_{to})} & \text{if } \mathrm{Trfc}(n_{to}, n_{from}) = 0 \\ \dfrac{1}{Dstn(n_{from}, n_{to}) \cdot \mathrm{Trfc}(n_{to}, n_{from})} & \text{otherwise.} \end{cases}$$

where:
$\mathrm{Trfc}(n_{to}, n_{from})$ is the number of pedestrians travelling from n_{to} to n_{from}
(These are travelling in the opposite direction to the pedestrian evaluating this function),
$Dstn(n_{from}, n_{to})$ is the distance between n_{to} and n_{from}.

Figure 7.7: The rate at which components move from current to the next node depends on the traffic on the edge between the current and the next node, in the opposite direction. This model only reflects the case where only the people travelling in the opposite direction slow the pedestrian down.

goal location of the new *Pedestrian* instance. The probability of performing a particular broadcast output action in CARMA is computed using a propensity function associated with each communication candidate. The propensity function can be parametrized using values from the store of the sender and the receiver as well as on the global store of the modelled system. In the presented model we used the propensity functions shown in Fig. 7.6. All of the possible entry locations $n_{loc} \in [0,2,3,5,6,7,8,20]$ (see Fig. 6.4) have the propensity value ≥ 1.0. Other locations have the propensity value of 0.0, which means that they are never chosen as the start or goal location. Two particular locations, loc_0 and loc_{20}, situated at the extreme points of the model of The Middle Meadow Walk have a propensity that varies over the time of the day, as shown in Fig. 7.6. In this way, when spawning a *Pedestrian*, the propensity of assigning location loc_0 (most-north) as the start location reaches its peak at 5:00 pm and as the goal location - at 9:00 am. For location loc_{20} (most-south) this trend is reversed, i.e., the propensity of it being assigned as the start location of a new *Pedestrian* is highest at 9:00 am and as the goal location - at 5:00 pm. We used the Gaussian function (see Fig. 7.6) to model this increase in propensity.

7.3 Movement and routing

A *Pedestrian* component is always trying to reach its goal location (see Fig. 5.6). In order to do that, it performs a sequence of actions, which changes its state in the cycle

$$\mathbf{Pr_{choose}}(n_{next}, n_{curr}, n_{goal}) = \begin{cases} \dfrac{Dstn_d}{\mathrm{Trfc}(n_{next}, n_{curr})} & \text{if } \mathrm{Trfc}(n_{next}, n_{curr}) > 0 \\ & \text{and } Dstn_d \geq 0, \\ \dfrac{-1}{Dstn_d \cdot \mathrm{Trfc}(n_{next}, n_{curr})} & \text{if } \mathrm{Trfc}(n_{next}, n_{curr}) > 0 \\ & \text{and } Dstn_d < 0, \\ \dfrac{-1}{Dstn_d} & \text{if } \mathrm{Trfc}(n_{next}, n_{curr}) = 0 \\ & \text{and } Dstn_d < 0, \\ Dstn_d & \text{if } \mathrm{Trfc}(n_{next}, n_{curr}) = 0 \\ & \text{and } Dstn_d \geq 0. \end{cases}$$

where:
n_{curr},n_{goal},n_{next} are the current, goal and (prospective) new location of a *Pedestrian* component,
$Dstn_d = Dstn(n_{curr}, n_{goal}) - Dstn(n_{next}, n_{goal})$,
$Dstn(n_1, n_2)$ is the distance between two given nodes,
$\mathrm{Trfc}(n_{next}, n_{curr})$ is the number of pedestrians whose current
location is n_{next} and next location is n_{curr}.

Figure 7.8: The propensity of choosing a particular node as the next location.

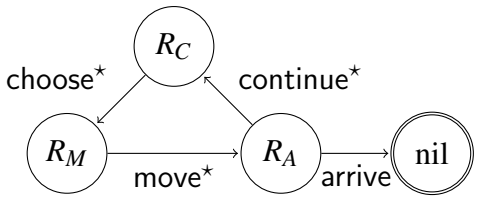


Figure 7.9: The states of the process of the *Pedestrian* component. R_C is ReadyToChoose, R_M is ReadyToMove, R_A is ReadyToArrive, and reaching the nil state is equivalent to the component being removed from the system.

shown in Fig. 7.9. The behaviour of a *Pedestrian* component in the Meadows model is very similar to that of the *Pedestrian* component in the basic model (see Fig. 7.1). A path is chosen after which the move takes place. There is also a check to see if the final destination has been reached. When the *Pedestrian* component is in the ReadyToChoose state, it must perform the choose* action, which will determine the next node which the component will change its current location to.

In a fashion similar to the way the start and goal locations are chosen before a new *Pedestrian* is created, multiple *PathNode* components compete with each other to de-

cide which one of them is going to synchronize with the *Pedestrian* component on the choose* action, determining the next location for the *Pedestrian* to move to.

The propensity of synchronising with a particular *PathNode* on the choose* action is given by the function shown in Fig. 7.8. The propensity is calculated for each potential next node. This formula was chosen in order to model the intention of the *Pedestrian* component to move towards its goal node, while at the same time avoiding routes with heavy congestion. Let us first consider the case when the value of traffic is 0 (there are no other *Pedestrian* components travelling along the same connection). In that case, the propensity depends only on the value of the *Dstn* factor. $Dstn_d$ is a measure of how much closer the *Pedestrian* will get to its goal node, by moving to the next node. It is computed by calculating the difference between the *Pedestrian*s distance to its goal from the current node and from the next potential node. If the value of $Dstn_d$ is positive, the current node is further from the goal node than the next potential node - and so, moving to the next node is desired since it brings the *Pedestrian* closer to its goal. The propensity value of such a move is proportional to the potential decrease in distance from the goal, represented by $Dstn_d$. If the value of $Dstn_d$ is negative, moving to the next potential node takes the *Pedestrian* further away from its goal node, and so the propensity value associated with this move is inversely proportional to the value of $Dstn_d$.

The same relationship between the value of $Dstn_d$ and of propensity is also used in the case when there is a non-zero traffic on the connection to the potential node. The propensity of choosing a node is always inversely proportional to the measured traffic.

The graph structure of the network of paths is incorporated into the system using a different approach than in the basic model. Rather than invoking the *ExistPath* function in order to determine whether two nodes are connected by an edge or not, a special expression from the space syntax available in the newest version of CASL is used. The loc_A *in* $loc_B.post$ expression evaluates to a boolean value which is *true* if the loc_A location is in the post set of the loc_B location, and *false* otherwise.

The predicate of the choose* action uses the above syntax to ensure that there exists an edge from the *Pedestrian* component to the *PathNode* component which the *Pedestrian*

is attempting to synchronize with.

If there is more than one *PathNode* in the postset of the *Pedestrian* component location, a given *PathNode* is more likely to be chosen over another if it is closer to the goal node of the *Pedestrian*, and less likely to be chosen if the edge between the current location of the *Pedestrian* leading to that node has more traffic than the edge leading to the other node.

This introduces a form of routing - *Pedestrian* components are more likely to choose paths with less traffic.

The distances between nodes are pre-computed and are encoded into the model in the form of a globally accessible two-dimensional look-up table. When the *Pedestrian* component arrives in its goal node, it is removed from the system.

Moving from one node to another is a CARMA action. The rate of the action was chosen to reflect the delay caused by the distance between two nodes, as well as the congestion on the path leading from one node to another (see Fig.7.7). The value of the distance as well as the value of the measured traffic are inversely proportional to the resulting rate of the movement action. This means that, for example, on average a *Pedestrian* will traverse two 50 m long paths in the same time as it would traverse a single 100 m long path.

7.4 Results

We simulated the model for the following two cases:

1. all the connections are available to the *Pedestrians*,

2. one of the segments (the middle segment of the Middle Meadow Walk, the edge between nodes 1 and 4) is closed, for example because of having been flooded.

The scenarios described above are two possible example situations to explore. If required, the model can be easily adapted to different path closure situations.

The results presented in Fig. 7.17 are dependent on the shape of the network of paths as well as the constants in the formulas. At any node a *Pedestrian* has to make a decision, which in our system is modelled by the stochastic nature of CARMA actions. The outcome depends on how congested the potential model lanes are, as well as on how preferable the next node is in terms of its distance from the goal location. Varying the weights of these two factors can have an influence on the behaviour in the system.

For each of the two scenarios we obtained snapshots of the state of the systems at four times of the day: $t \in [9, 13, 17, 21]$ (hours). The snapshots presented in Fig. 7.10–7.17 show average amounts of *Pedestrian* components on graph edges measured in the time period $(t-\delta, t+\delta)$, where $\delta = 1$hour. The two colours of arrows denote two opposite directions of the movement of *Pedestrian* components. The length of an arrow represents the proportion of *Pedestrian* components travelling in one direction and *Pedestrian* components travelling in the other direction to the total number of *Pedestrian* components on that edge. The total number of *Pedestrian* components on a given edge is represented by the arrow's thickness.

7.4.1 Discussion

The results show that in the initial scenario, a large portion of *Pedestrian* components in the modelled system choose to travel along the Middle Meadow Walk lane. This is not surprising as our goal and start assigning functions are designed in such a way that most of the *Pedestrians* would have started either at the North-most or South-most end of the Middle Meadow Walk and would be heading towards the other end. The Middle Meadow Walk constitutes the shortest, most optimal (if traffic is not an issue) route between these two points. There are of course a number of new *Pedestrian* components arriving into the system at different nodes along the edges of the Meadows, but the frequency with which they are spawned is much lower. The number of *Pedestrian* components who are travelling along the Middle Meadow Walk in the North direction

is roughly the same as the number of *Pedestrian* components travelling in the South direction, at most time snapshots. It is also noticeable that *Pedestrian* components who started their journey at either of the Middle Meadow Walk ends, when presented with the first decision point (and do not choose to continue travelling along the Middle Meadow Walk at that point) are most likely to choose the connection first to the right (for those travelling South-wise) or to the left (for those travelling North-wise) of the Middle Meadow Walk. This is true for both groups of *Pedestrian* components, as from both directions such choice brings them closer to what is the goal node to most of them (the opposite end of the Middle Meadow Walk to the one they started at) than any other alternative connection does. The spread of *Pedestrians* with respect to which direction they are travelling in at any given connection is relatively uniform throughout the network, at all snapshot points.

In the second scenario, the closure of the central segment of the model of The Middle Meadow Walk forces *Pedestrian* components to choose a different path instead. This increases the traffic on the nearby lanes, however the lanes that are farther away are not significantly affected, as the traffic disperses through the network more evenly after a number of movement steps. It is visible from the results that in this case *Pedestrian* components are still more likely to choose certain connections at their first decision point, namely the connection first to the right (for those travelling South-wise) or to the left (for those travelling North-wise) of the Middle Meadow Walk.

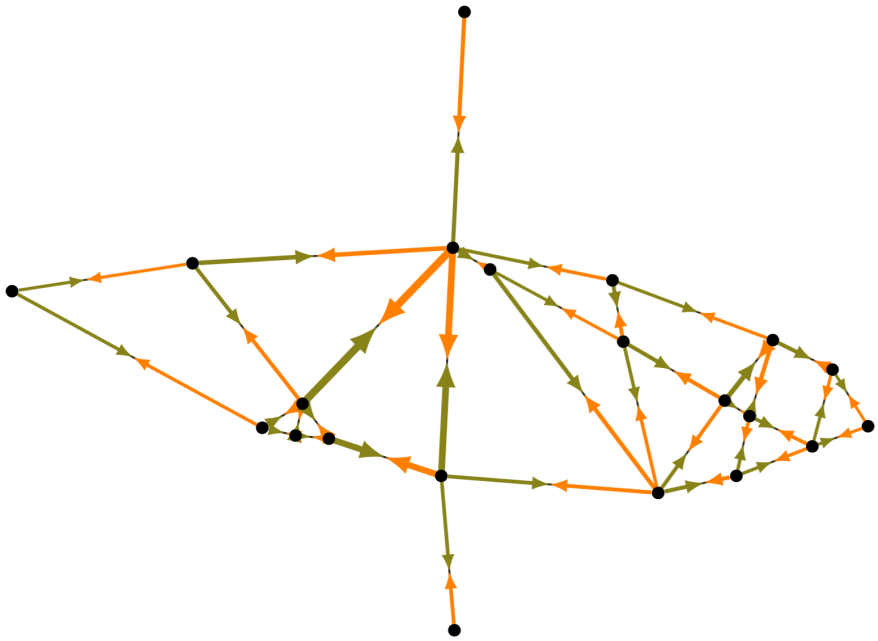


Figure 7.10: Scenario (1) at time $t = 9$

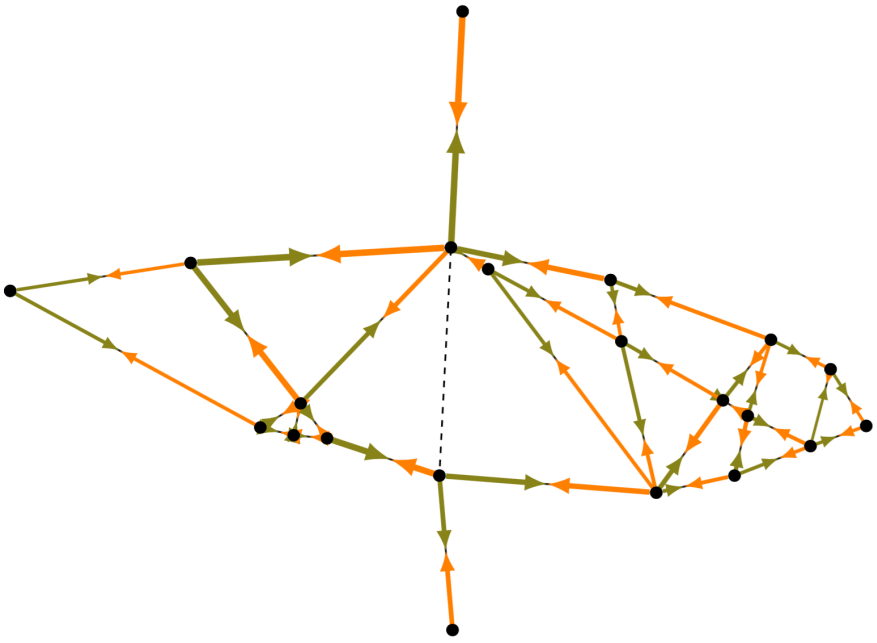


Figure 7.11: Scenario (2) at time $t = 9$

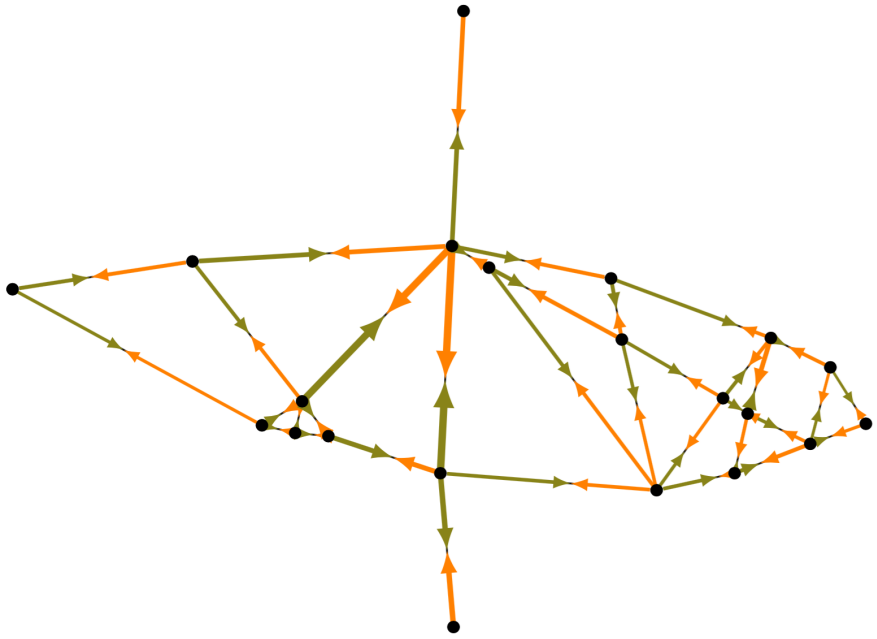


Figure 7.12: Scenario (1) at time $t = 13$

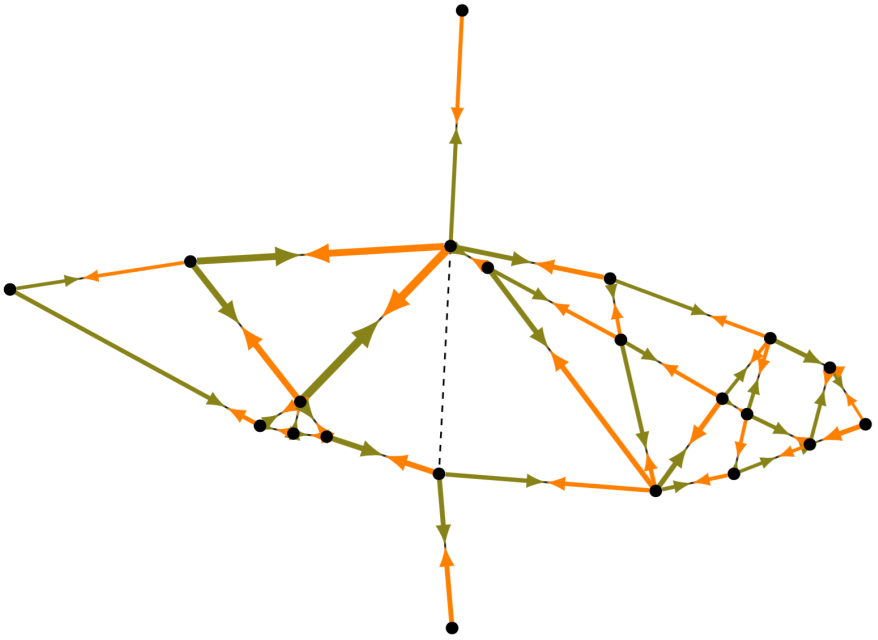


Figure 7.13: Scenario (2) at time $t = 13$

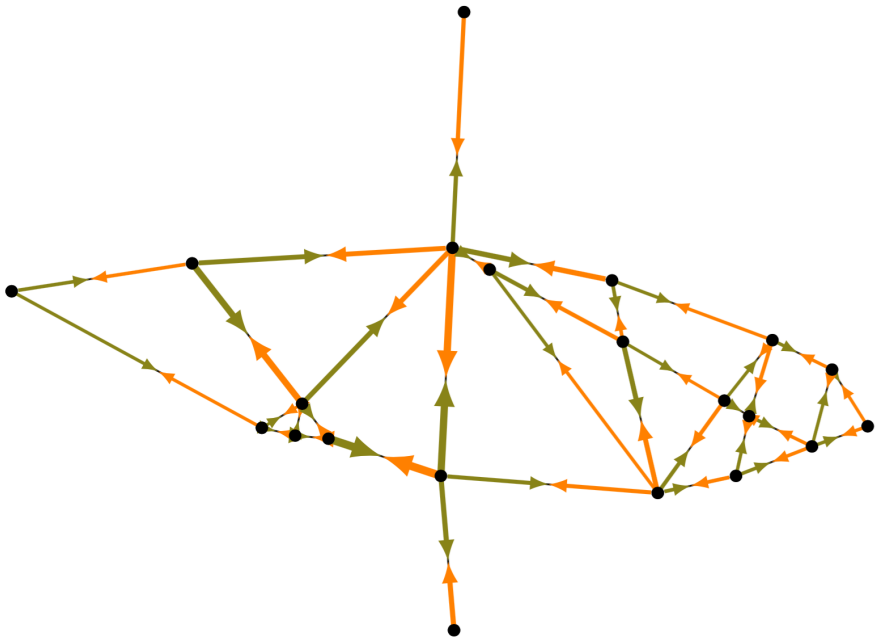


Figure 7.14: Scenario (1) at time $t = 17$

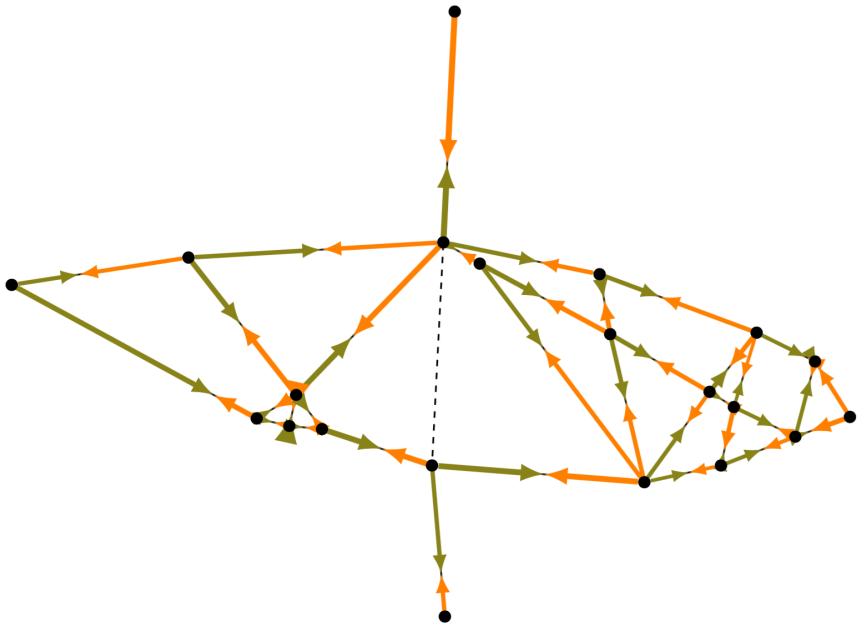


Figure 7.15: Scenario (2) at time $t = 17$

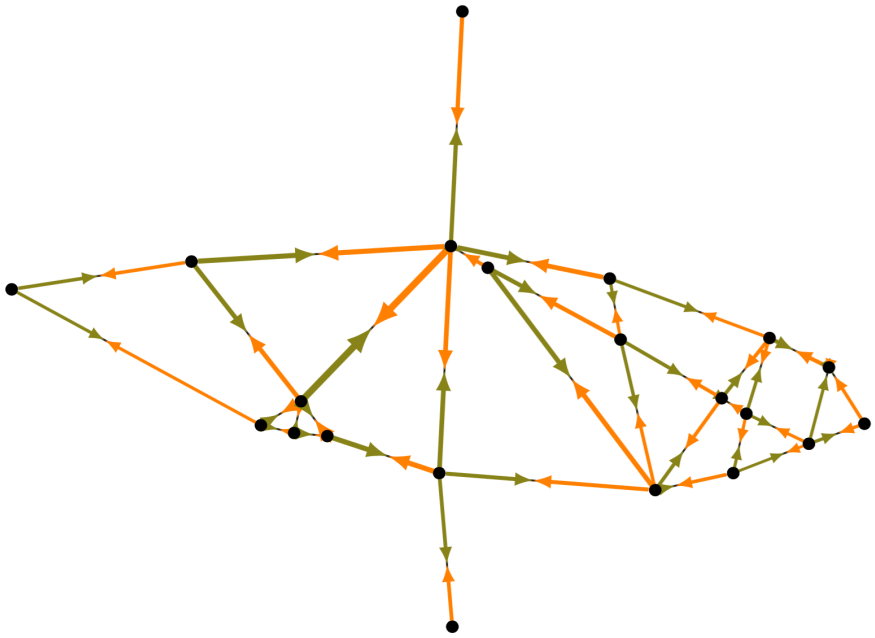


Figure 7.16: Scenario (1) at time $t = 21$

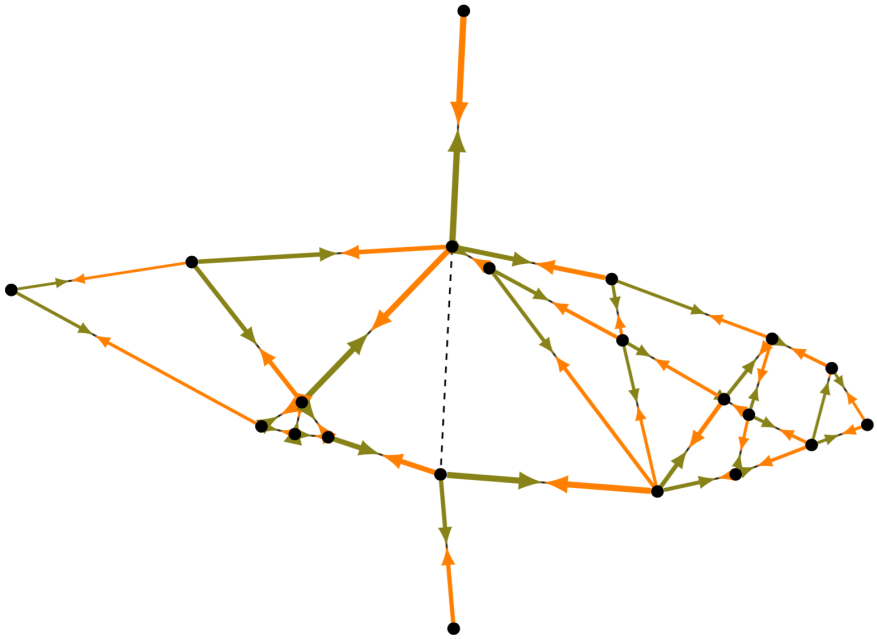


Figure 7.17: Scenario (2) at time $t = 21$

7.5 Conclusions

In this chapter we learnt how to apply a formal model in CARMA to a scenario which is to a certain degree based on real world data. Developing and analysing this model illustrates the usage of the CARMA language and its software suite for the purpose of developing models based on real data. At present, the only data on the Meadows city park available for analysis is the geographic data from the Google Maps API and the bike count data from the City Council, measured at a single point in the park. If real world data measuring pedestrian traffic at a number of locations around the park becomes available, it may be used to parametrise the presented model, which would allow us to evaluate the predictive power of the model.

Chapter 8

Data-driven modelling of urban transportation systems

A developed country is not a place where the poor have cars. It's where the rich use public transport.

paraphrased from Enrique Penalosa, former Mayor of Bogotá, Colombia

8.1 Introduction

In this chapter we present a model of an urban transportation system. We compare aspects of real data collected from a city bus system in the city of Edinburgh, UK, with the results of simulations of the model constructed in the CARMA language. The simulations show results which are in good agreement with the real-world data, leading us to believe that the model could have useful predictive powers and thus provide an environment for experimentation with possible changes to the design of the system.

This model builds upon the two simpler models presented in Chapters 5 and 7. It

takes advantage of lessons which have been learned from these models and provides the proof of concept of applied formal modelling by utilising a large and rich source of real world data. The main thing that differs this model from the one presented in Chapter 7 is how heavily based on data it is. We also for the first time in this thesis compare the results of the simulation with real data to assess how well the model represents the real-world system.

Deeper insights into the causes of various types of system behaviour can be obtained by combining data from several independent open data sources. This combination of data sets provides a different perspective on the use of the road network, allowing us to make a more detailed model which would not be possible if working from a single source of data. For the model in this chapter, we have combined Automatic Vehicle Location (AVL) data which was obtained from the Transport for Edinburgh company [18] with long-run average data on traffic intensity from the Tom Tom satellite navigation service [16]. These are two genuinely independent data sources, the real-time vehicle location system on the buses does not provide data to the Tom Tom network, which harvests data from their own propriety hardware installed in private vehicles. The data sources used in our research are described in detail in Chapter 6.

In order to analyse a real-world example of a problem in this domain, we constructed a formal model of the system of interest in the modelling language CARMA and studied it via simulation. In contrast to logic-based explicit state-space analysis approaches, such as probabilistic model-checking [65] with model-checkers such as PRISM [69] and Storm [34] (see 2.4), simulation provides no absolute guarantees of correct behaviour but it scales to allow the construction of very detailed spatial models of systems such as the location-accurate bus route which we have modelled here.

8.2 Related work

There is a number of existing approaches to stochastic modelling of urban transportation systems. In [105] the authors describe utilize Bio-PEPA in order to develop a method that allows for checking how well bus providers are fulfilling their agree-

ments set with respect to allowable delays in departures. They propose an alternative timetable the implementation of which would result in less potential delays. In [84], the authors use Monte Carlo simulation techniques as well as time series analysis in order to construct methods for evaluating specific metrics of overall bus sytem performance, based on the information about 'headways' (the time between subsequent bus arrivals). There is existing research relating to conceptually similar problems from the area of urban transport, mainly bike-sharing [83]. The paper talks about how to design optimal placement of bike sharing stations for such a system, and gives examples based on 11 cities worldwide.

The work presented in this chapter differs from these existing approaches in that it combines the spatial data about the locations of road nodes and bus stops with the temporal data from bus timetables and also factors in traffic information coming from a completely separate data source.

8.3 Locations

8.3.1 Coordinate system

In our model, we use locations extracted from the real data provided to us by Transport for Edinburgh. The easting and northing coordinates are expressed in the units of metres, and so are all the distances in the presented paper, unless stated otherwise. The detailed description of pre-processing of this data can be found in Chapter 6.

8.3.2 Data types

Each location in the model is represented by the tuple `(id,x,y)`. The `id` of a location is a unique identifier of a given location.

In the data from the Transport for Edinburgh API, locations are represented using latitude and longitude values from a continuous domain, as the buses move continuously and a given bus can be found using GPS at any location at the time of sampling. In our model, we represent movement as sequences of steps between discrete locations. These consist of locations of bus stops as well as any number of points on the way between two adjacent stops.

8.3.2.1 Looking up data inside the model

The location data within the model is usually requested in the same sequence as a bus would traverse these points on its journey. For example, if a given model contains only those locations that represent a stop, each Bus component would attempt to look them up one by one in the order in which they appear on that particular service's timetable. For this reason we map the original bus stop unique identifiers from the API sourced data into a different set of unique identifiers associated with a particular CARMA model name-space domain. In this way, we can ensure that for each service, the identifiers of locations which the bus is due to traverse, are represented by incrementing integers. This means that, conveniently, if a given bus is currently at the location with `id==i`, its next location has the identifier `id==i+1`.

Records in CARMA are indexed, and in our model the index represents the id of a given location. There are two records, x and y, for storing the x- and y- coordinate values of each location.

8.4 Departures

In our model, new Bus components need to be instantiated with a rate that reflects the departures of buses from the initial stop in the real data. A common pattern shared by the timetables shows buses departing less frequently in the morning and evening hours, and more frequently during the day, however the exact pattern of departures

differs from service to service.

In the CARMA system, we model the departure timetables using a function which returns a transition rate for triggering an action that results in instantiating a new Bus component at a starting location. This rate should be dependent on the time of the day in a way that reflects the data in the timetables.

In order to obtain the rate of transition for the Bus component instantiation, we will use a function that returns the period of bus departure occurrences given a certain time of the day. This function is calculated from the list of departure times on a timetable (there is a timetable for every given day of the week, service and destination).

One way to calculate an approximate value of the period of bus departure events, is to look at the differences between consecutive bus departures in a certain range of time around the time for which the period is being calculated. For example, when calculating the frequency of bus departures at 12 am, we take a mean of the time differences between consecutive buses departing between 9 am and 3 pm.

Another approach is to calculate a weighted average of the differences between bus departure times, with the weight being proportional to the time difference between the considered departures and the time of sampling. For example, when calculating the frequency of bus departures at 12 am, we take into account all of the time differences between consecutive buses that can be calculated from the available data. However the closer their time is to 12 am, the higher the weight associated with their value when calculating the average. In other words, this formula is a weighted average of differences between consecutive bus departure events and the weight is the distance [in the unit of time] between the average value of the two departures whose difference we are calculating, and the point of sampling.

The comparison of the two methods described above is presented in Fig. 8.1. The graph is based on the timetables of two stops which are the starting locations of service number 5 to Hunter's Tryst. The early morning departures start at Brunstane, while the rest of departures start at The Jewel. The vertical lines represent actual departure times, as sampled from the timetables.

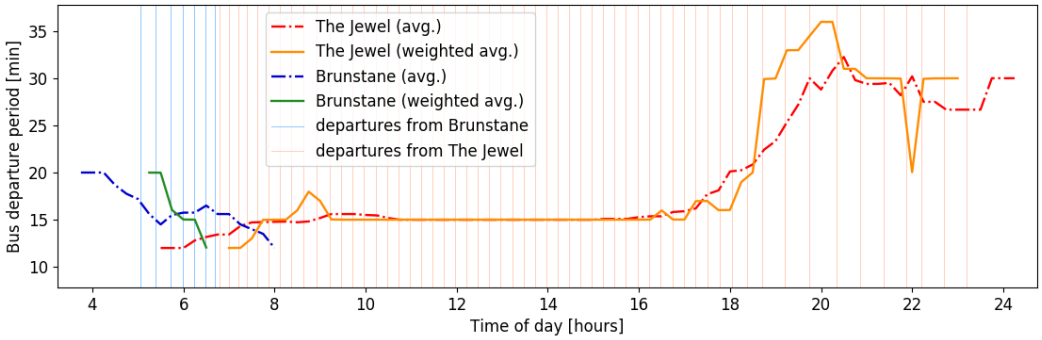


Figure 8.1: The period of bus departures per time of day calculated using a simple average over 3 hours, and using the weighted average method.

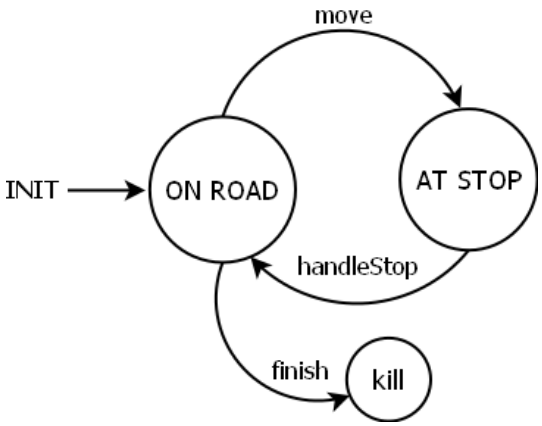


Figure 8.2: The internal state diagram of the Bus component.

In the simulation results presented later in this paper, we used a model based on the period function generated using the weighted average method.

8.5 Movement

8.5.1 Changing locations

In the presented model, a Bus is a CARMA component having two states, 'ON ROAD' and 'AT STOP'. The state graph representing its behaviour is shown in Fig. 8.2.

In the 'ON ROAD' state, the Bus component can perform a transition which, depending on the value of the predicate expression, has two possible outcomes:

- the Bus component updates its store to reflect its new location; or
- the Bus component is removed from the system.

In the CARMA model this is represented by two actions, $move^{\star}$ and $finish^{\star}$. Performing either of them results in exiting the 'ON ROAD' state. If the $move^{\star}$ action is triggered, the bus enters the 'AT STOP' state. In the other case, the bus has reached its destination, and the component is removed from the system.

In this manner, each of the Bus components traverses a number of locations on its route, until arriving at the destination. The continuous nature of movement of real-world vehicles is modelled using this step-based discrete approach.

When in the state 'AT STOP', the Bus component needs to perform the $handleStop^{\star}$ action in order to transition to the 'ON ROAD' state. This action simulates the additional waiting time a bus takes when passengers are leaving and boarding. This transition's rate depends on the kind of location the bus is currently at. When in this state, the bus component can be in one of the two situations:

- the current location in the store is a stop
- the current location in the store is a point on the route but not a stop.

If the location is just a point on the route and not a bus stop, the action has a fast rate, which means its delay time can be ignored and we may assume that it is triggered instantaneously. In the other case, the rate has a constant value which reflects the waiting time.

The elements of the model in CARMA, are shown in Figures 8.3 and 8.4.

Store of *Bus* component:

startL	(const) start location
startTime	(const) start time
goalL	(const) final location
currL	current location

Behaviour of *Bus* component:

$$\begin{aligned} MOVE &\stackrel{\text{def}}{=} [currL \neq goalL]\mathsf{move}^{\star}[\bot]\langle \mathsf{my}.currL\rangle\{\mathsf{my}.currL := \mathsf{my}.currL + 1\}.WAIT \\ &\quad + \ [currL == goalL]\mathsf{finish}^{\star}[\bot]\langle startTime\rangle.\mathbf{kill} \\ WAIT &\stackrel{\text{def}}{=} [isStop(currL)]\mathsf{handleStop}^{\star}[\bot]\langle\rangle.MOVE \\ &\quad + \ [isroutePoint(currL)]\mathsf{handleRoutePoint}^{\star}[\bot]\langle\rangle.MOVE \end{aligned}$$

Initial state of *Bus* component: $MOVE$
Values passed to *Bus* component's store on initialization: $startL, startTime, goalL$

Store of *BusGenerator* component:

startL	(const) start location for new Buses
goalL	(const) goal location for new Buses

Behaviour of *BusGenerator* component:

$$GEN \stackrel{\text{def}}{=} \mathsf{generate}^{\star}[\bot]\langle \mathsf{my}.startL, \mathsf{my}.goalL\rangle.GEN$$

Initial state of *Bus* component: GEN
Values passed to *Bus* component's store on initialization: $startL, goalL$

Figure 8.3: The *Bus* and *BusGenerator* components of the Lothian buses model. The *move*, *finish* and *generate* spontaneous actions broadcast a message even though no component will receive it - the content of the message can be accessed from the environment and is used to compute the rate for these actions as well as for updating the global store.

Evaluation context:

$$\mu_p(\gamma_s, \gamma_r, \alpha) = 1$$

$$\mu_w(\gamma_s, \gamma_r, \alpha) = 1$$

$$\mu_r(\gamma_s, \alpha) = \begin{cases} \mathbf{Rate_{move}}(now, \gamma_s(currL)) & \text{if } \alpha = \mathsf{move}^\star \\ \mathbf{Rate_{generate}}(now) & \text{if } \alpha = \mathsf{generate}^\star \\ \lambda_{\mathbf{stop}} & \text{if } \alpha = \mathsf{handleStop}^\star \\ \lambda_{\mathbf{routePoint}} & \text{if } \alpha = \mathsf{handleRoutePoint}^\star \\ 1 & \text{otherwise} \end{cases}$$

$$\mu_u(\gamma_s, \alpha) = \begin{cases} \emptyset, \left(Bus, \{startL \leftarrow \gamma_s(startL), startTime \leftarrow \text{now}, goalL \leftarrow \gamma_s(goalL), currL \leftarrow \gamma_s(startL)\}\right) & \\ \qquad \text{if } \alpha = \mathsf{generate}^\star & \\ \{\}, 0 \quad \text{otherwise} & \end{cases}$$

Initial collective:

$$Meadows \stackrel{\text{def}}{=} (BusGenerator, \emptyset)$$

Figure 8.4: Environment and evaluation context of the model of Lothian Buses. For routes with multiple initial stops, such as the one in our example, there are multiple BusGenerator components one for each stop at which buses start their journey. λ_{stop} and $\lambda_{routePoint}$ are constants chosen to reflect the fact that a bus takes longer time at a bus stop than at a regular route point.

8.5.2 Speed

The average speed at which a bus is moving is modelled by adjusting the rates of the $move^\star$ and $handleStop^\star$ actions of the Bus component in the CARMA system.

The rate of the $move^\star$ action depends on the distance between the current and next location as well as the current value of traffic. The $handleStop^\star$ action has a constant

rate, but it only results in delays in locations which are stops.

8.6 Limitations of the CARMA implementation

Because of the limitations that the syntax of CASL imposes on defining measures, the data obtained for the analysis could not be extracted directly from the CARMA simulation in the usual way. Each CARMA model for the purpose of simulation is translated to the Java programming language. This is done automatically, each time a CARMA file is saved in the Eclipse IDE.

One set of data that is not directly available through CASL, is the exact values of store variables of all components in the system, at any given time. In CASL, it is only possible to define measures that return the average, minimum or maximum from the set of sampled values. In order to obtain the complete data, sampled at each step of the simulation, the generated Java project was post-processed to save the following information, after each Bus component performs an action:

- bus identifier
- current time
- bus start time
- bus current location (x, y)

This post-processing was performed automatically using Python scripts.

The prototype versions of this model were developed using CGP and Spatial CASL. In order to make it feasible to run the simulation with the full data we needed to apply a number of optimizations to these models, mainly pre-computing most computationally heavy values such as distances between graph nodes. These were stored in

lookup tables indexed with location unique identifiers for fast lookup. We assigned these identifiers to the locations in such a way that the following always holds true: curentLoc $+ 1 =$ nextLoc. This was possible as the progression of a Bus through the system of nodes is always sequential and there are no decision points in the system, as to which node to move to next. This allowed us to simplify the implementation further by replacing the locations with an integer that was incremented every time the Bus changed location which had a great impact on the overall simulation time of the system.

8.7 Application of the model

8.7.1 Real world data source

The Transport for Edinburgh company [18] has provided us with access to the REST API that can be used to obtain data gathered from their bus system. The data used for the purpose of this research is relevant to the routes, timetables and live vehicle locations of bus services provided by Lothian Buses [14], a bus operator within the Transport for Edinburgh company. The data source is described in detail in Chapter 6 of this thesis.

8.7.1.1 Locations and points on routes

The objects ascribed location values are either bus stops or points on the route. The points on the route are included to preserve the shape of the bus route, which, when reconstructed using only stop locations may be missing important information such as road turns. In this thesis the term ‘points on routes’ is used to collectively refer to bus stops as well as other points the route whose locations are included in the Transport for Edinburgh API data. In the API data, each location is represented by its geographical latitude and longitude.

8.7.1.2 Bus stops

Apart from their locations, bus stops have been additionally given a unique identifier and a name, for example ‘Shandwick Place’.

8.7.1.3 Services

The `services` API endpoint provides information about all the existing services - including their names, destinations, and routes.

8.7.1.4 Live vehicle locations

The live vehicle locations endpoint of the Transport for Edinburgh Open Data API provides snapshots of the bus system at the time of request. This information includes the geographical location of all the currently active buses, as well as the next stop on their journey.

8.7.2 Traffic

We used data obtained from The Tom Tom Traffic Flow [16] service in order to estimate traffic values. This data source is described in detail in Chapter 6.

To lessen the probability that an incident (roadwork, closure) not included in the simulation would affect the results, we used data from a day in which all the causes of increased congestion were categorised as ‘jams’ in the Live Traffic Reports service.

Figure 8.5: The route of service number 5 to Hunter's Tryst, stops are indicated by white circles. Screenshot taken from Lothian Buses Network Maps webpage [15].

8.7.3 Instance of the model: Service number 5 to Hunter's Tryst

In order to create an instance of the model, we used data collected from buses on the route of Lothian Buses' service number 5 to Hunter's Tryst, using the weekday timetables (that is, excluding Saturdays, Sundays and Bank Holidays).

This route has 124 points on routes, 58 of which are stops.

We applied the same measures to the data obtained from the Lothian Buses API and from the simulation of our CARMA model.

We compare the real-world data with the model using the number of active buses per time of day. This measure depends on two aspects of the system:

- the frequency at which new buses are introduced into the system

- the length of time a bus remains in the system.

In this case, both of these characteristics are time-dependent and at the same time independent of each other.

The results obtained from the simulation correspond well to the real-world data. The greatest disagreement can be observed in the initial stage of the system evolution (morning hours). This discrepancy may be explained by the artefacts of the real data: many vehicles that are to start their service only later in the day appear as active in the system as soon as the GPS mechanism is switched on. In the presented graphs we removed the buses that appear as active before the first departure time of the timetable, however further and more selective data cleaning needs to be performed to eliminate this inconsistency in the later morning hours.

An interesting observation can be made about the influence of the traffic on the number of active buses. In the simulation instance without traffic (i.e. the speed depends only on the distances), shown in dark blue in Fig. 8.6, the number of active buses is underestimated in the time ranges 9:00 am - 11:30 am, 4:00 pm - 9:00 pm and overestimated between 2:00 pm and 3:00 pm. In the simulation with traffic (shown in dark red), the trend seems to be reversed for the time ranges 9:00 am - 11:30 am and 2:00 pm - 3:00 pm. This means that if the influence of traffic was smaller by a particular amount, the simulation would fit the data with greater accuracy. The reason for the traffic to have a smaller influence on buses, than it has on other vehicles (those equipped with Tom Tom GPS devices, which are the source of the traffic information) is probably the fact that buses can travel along privileged bus lanes. If we assume that a majority of vehicles used by Tom Tom for data gathering do not travel in such lanes, this discrepancy can be explained by the lower average speed of those vehicles than that of buses.

Between 7:45 pm and 12:00 am the inclusion of traffic seems to have a negligible effect on the simulation, and the active buses count is underestimated by both simulation instances in the time range 7:30 pm - 9:30 pm.

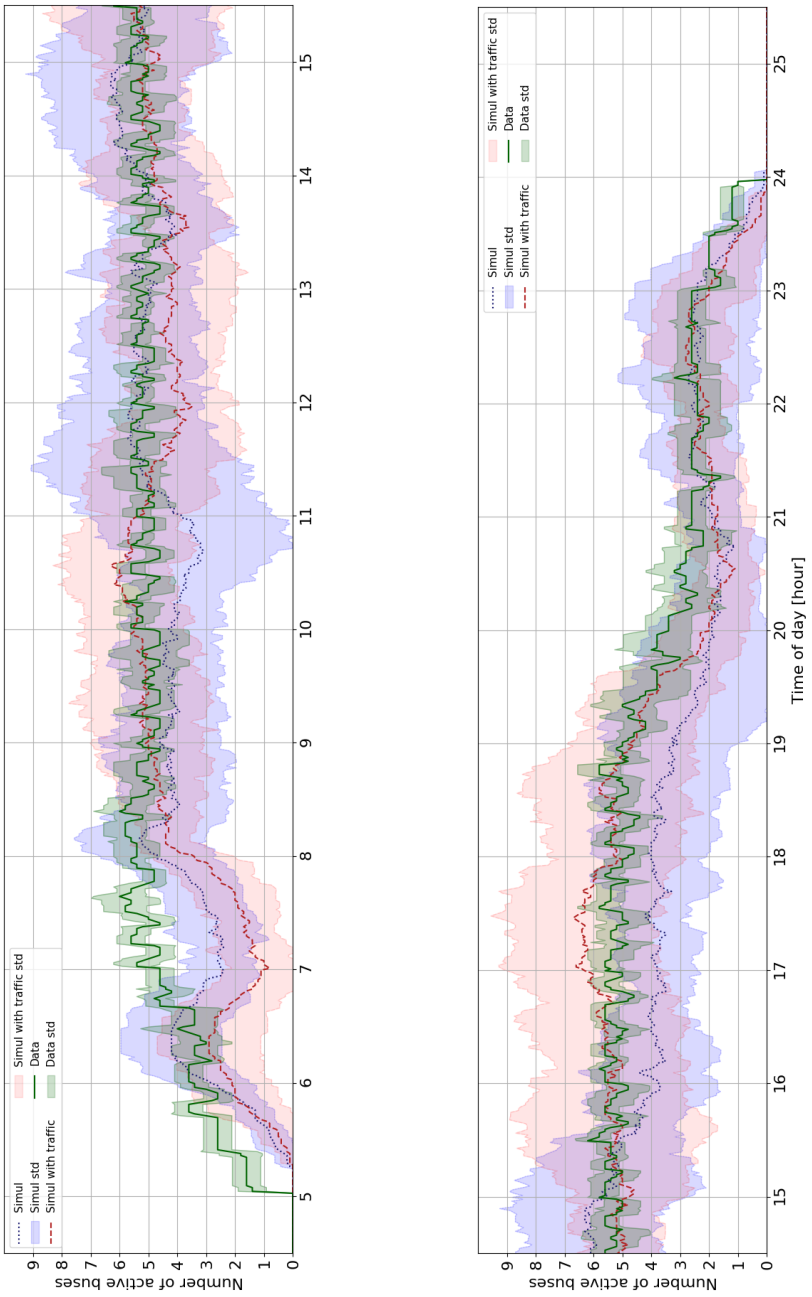


Figure 8.6: The number of active buses versus the time of day, observed in the real data, simulation with traffic and simulation without traffic.

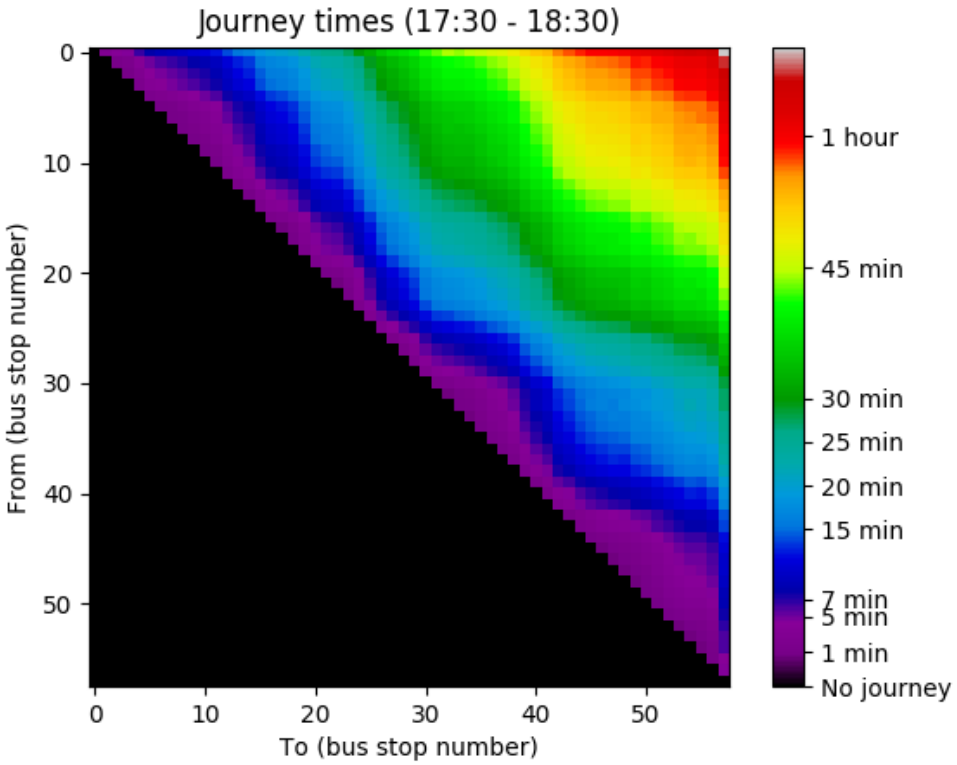


Figure 8.7: The average journey length for each pair of bus stops on the route arriving at the destination stop between 5:30 pm and 6:30 pm (data).

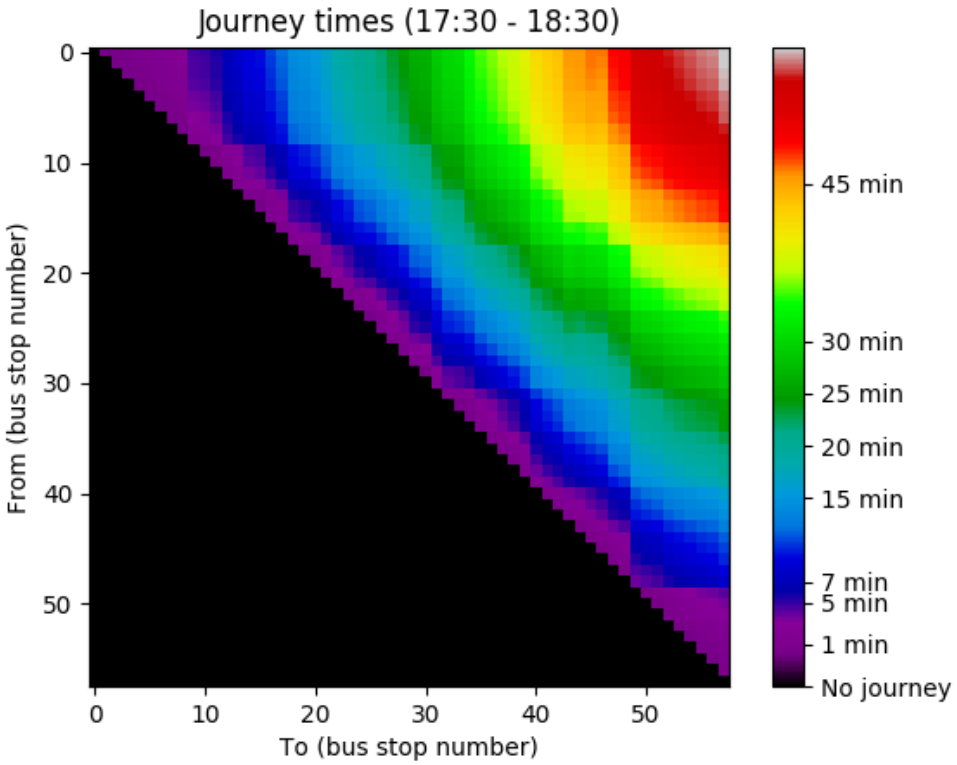


Figure 8.8: The average journey length for each pair of bus stops on the route arriving at the destination stop between 5:30 pm and 6:30 pm (simulation).

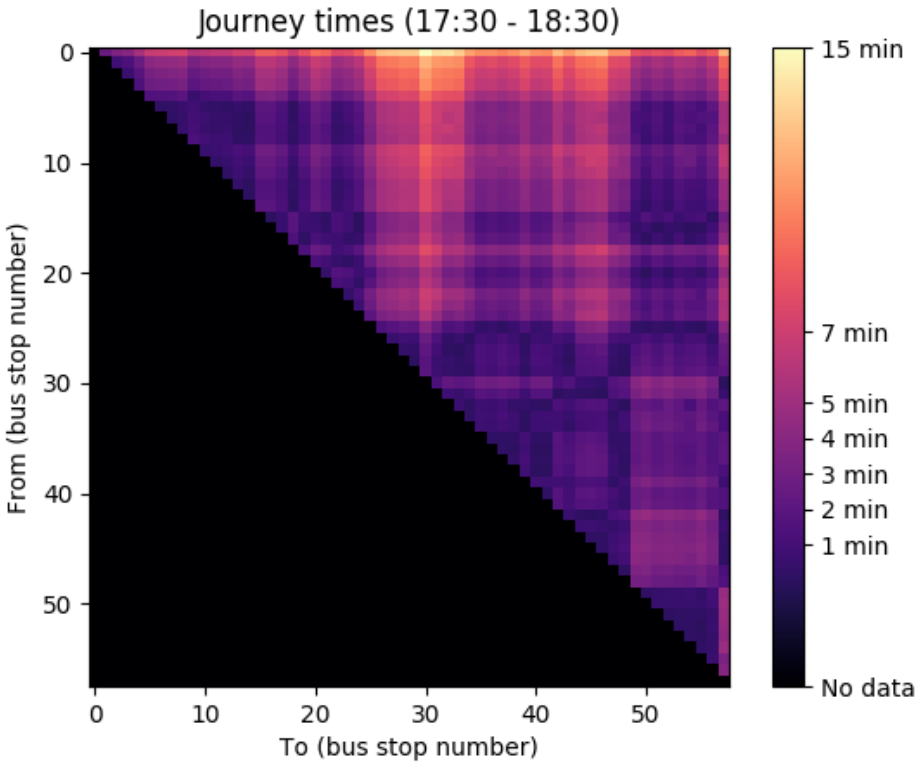


Figure 8.9: The absolute difference between data and simulation journey times for each pair of bus stops on the route arriving at the destination stop between 5:30 pm and 6:30 pm.

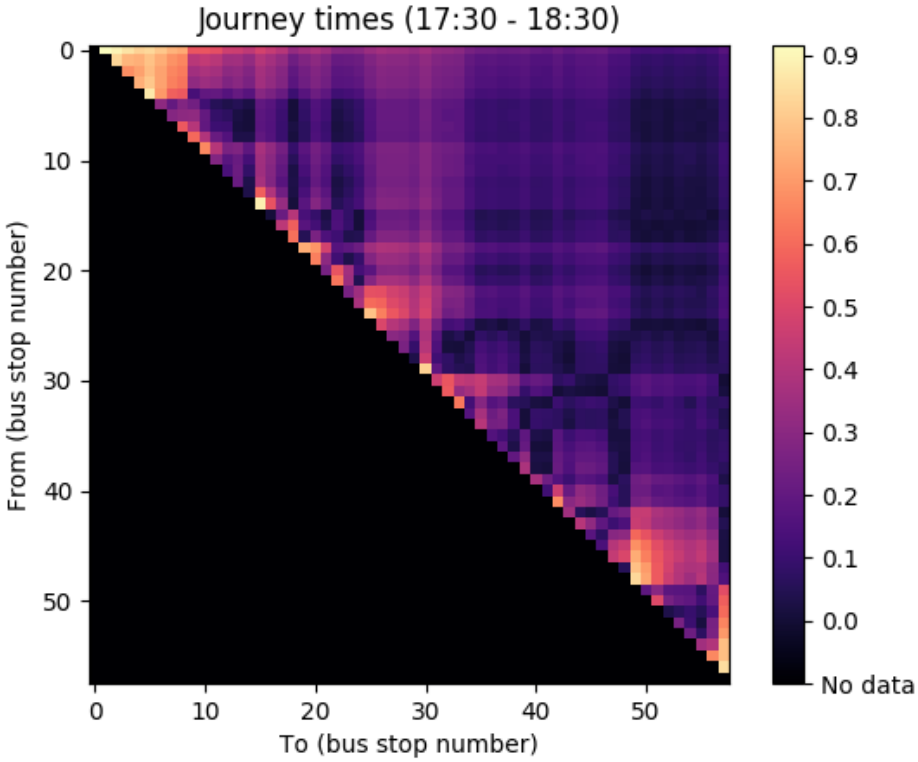


Figure 8.10: The relative difference between data and simulation journey times for each pair of bus stops on the route arriving at the destination stop between 5:30 pm and 6:30 pm. This was calculated using the formula $Diff_{a,b} = \frac{abs(a-b)}{max(a,b)}$.

In the last time segment, both simulation instances seem to align with the real-data results very well.

An hour long snapshot of the system taken between 5:30 pm and 6:30 pm can be seen in Fig. 8.7, 8.8, 8.9, and 8.10. The x and y axes represent bus stops, which are indexed with integers, and sorted by the order they appear along the route. The colour of each pixel represents the average journey time from the stop designated by the x-axis value to the stop designated by the y-axis value. The striped pattern of discontinuities, which can be observed on all plots, are more pronounced in the simulation results. This is because they are the results of accumulated delayed or early departures of a bus from consecutive bus stops on the journey. In real life, when a bus arrives at the stop too early, the driver waits until the timetable departure time before continuing, minimising the overall value of headway. In this simulated model that mechanism was not represented. For this reason, we can observe that on some journeys, the duration times have been shifted for all consecutive arrival stops.

8.8 Conclusions

In this chapter we presented a model of urban transportation scheme and compared the results obtained from analysing this model through means of simulation with the real-world data. We were able to demonstrate the expressive capabilities of the CARMA language and its suite of tools. We showed how various types of data obtained from multiple sources can be combined together in order to create a representation which yields simulation results that are in good agreement with measures applied to the real-data only.

In contrast to existing models from the literature that are conceptually similar [83, 84, 105] our model deals with a large and varied source of data and combines multiple types of it in a single model.

Chapter 9

Conclusions

In this thesis we explored various aspects of the process of modelling and analysis of Collective Adaptive Systems (CAS) in which the spatial location of components contributes in a significant way to the overall behaviour of the system.

We chose to work with the CARMA process calculus and its suite of software tools, which are specifically tailored for investigating CAS case studies.

The work described in this thesis applies formal and theoretical means to data and uses software-engineering and simulation in order to produce results that aim to broaden our understanding of existing real-world case studies.

9.1 The scientific contributions

The scientific contribution of this thesis consists of the creation of tools aiding the analysis of the properties of CAS in which space plays an important role, as well as utilizing real data in the models created using these novel tools together with pre-existing ones, in order to demonstrate their applicability to solving real world problems

from the area of CAS modelling.

Chapter 3 describes the work undertaken to extend CASL with a supplementary syntax designed specifically for working with spatial systems. The addition of spatially-oriented syntactic constructs allowed us to abstract the spatial aspects away from the models and tailor appropriate data structures suitable for representing systems in which they occur. This provides the modeller with the ability to create cleaner, highly concise and more readable models by avoiding the need of explicitly specifying many aspects of the spatial infrastructure.

The creation of CGP, described in Chapter 4 was another step towards creating more user friendly tools for specifying spatial systems. The contribution here consisted of designing a graphical representation for CAS with movement and implementing this approach in the form of a plug-in tool to the Eclipse IDE which can be used to automatically generate CASL code from graphical input.

The first model presented in this thesis, in Chapter 5, describes a crowd routing scenario assuming the flow of pedestrians moves from one designated node to another designated note, through networks of connections having various sizes and complexity. The analysis of this simple model allowed us to assess the expressiveness of CARMA for the purpose of describing similar types of real-world scenarios. The results obtained from simulating the model were in accordance with an intuitive understanding of the effects of congestion on the throughput of crowd movement – in all experiment cases the average times of travel were lowest when there was no congestion present. Networks having greater height showed better performance, which is the result of them having greater capacity, specifically in the context of the first step of a pedestrian's movement, where the pedestrian has one more node available when deciding which node to move to next.

In Chapter 7, building on top of the first basic model, we have formulated a data-driven model of a system of pedestrian movement as a collective adaptive system where the actions of some participants within the system are influenced by the actions (or even the presence) of others. We approached the creation of a more detailed and more realistic model by building on the experience which we gained in creating a basic model with

simple and regular symmetric structure. The insights obtained from this basic model directed our attention to likely issues with the more realistic model, and deepened our understanding of the effects of local decisions on global behaviour.

The degree of self-awareness in the system gained through the (local) recognition and attempted avoidance of congestion in the network by the participants within the system itself makes the problem different in character from a traditional network-flow problem where data is routed through a network without even local knowledge of congestion ahead. Adaptive systems naturally give rise to phenomena such as these where the decisions of one participant are influenced by the decisions of another seemingly entirely independent participant and the longer-term consequences of a decision are usually infeasible to predict at the time of making the decision, and it may even be difficult to understand the path from decision to consequence in retrospect.

The suite of experiments which we carried out on the models presented here, and the many variants of these which were created in the model development process, provided us with a convincing demonstration of the applicability of the CARMA modelling tools to a modelling problem with aspects of microscopic and macroscopic perspectives and evidence of the practical effectiveness of the CARMA process calculus as an intellectual vehicle for expressing intelligent density-dependent movement across an arbitrary network topology.

We made extensive use of the CGP together with the CASL language, which enriches the core CARMA process calculus with additional language features thereby making it possible to express large models cleanly by using data types and data structures to represent structured information in the model, allowing the model to be statically checked for model coherence at compile-time. This eliminated a raft of potential modelling errors, streamlining the process of model creation. CASL additionally supports arbitrary function definitions for the propensity functions which give rise to probability distributions over model actions, allowing us to model user behaviour accurately; and general rate functions, allowing us to model timing behaviour accurately. In addition, it provides spatial data structures, allowing us to represent physical separation and distance accurately.

We used the CGP for the purpose of automatically generating CASL code from graphical input, for the purpose of instantiating the four presented scenarios.

In Chapter 8 we applied the CARMA process calculus to model an urban transportation system. The challenges of this technique originate from the continuous nature of the real-world data being represented in a discrete and stochastic modelling environment. The patterns one might expect to observe when viewing the system from a high-level and simplified perspective, are often distorted by numerous factors that influence the real-world data. For this reason, a stochastic model will always be an abstract representation of the considered system. The challenge therefore is to extract a generic paradigm that can describe a given instance with a satisfiable accuracy, while at the same time remaining applicable to a range of other instances.

The simulations based on the presented modelling approach result in patterns which are in good agreement with those observed in the real data. An interesting feature of the Transport for Edinburgh transportation system which emerged from comparing the data with simulation, is the fact that traffic influences buses to a lesser extent than other vehicles (presumably because of the existence of bus lanes).

This good agreement between simulation results produced by the model and real-world behaviour means that experiments with the model can be used to effectively evaluate potential modifications to the real-world system, or to check the accuracy of other descriptions, as in [105].

The significance of the scientific contribution of this thesis lies first and foremost in demonstrating how the theoretical and software-based tools for modelling of CAS may be applied to real world data in order to obtain meaningful results and thus showing the promising potential of combining highly theoretical modalities such as formal modelling with substantial concrete problems based on real-world data.

9.2 Limitations

There are a number of limitations to consider with respect to the work described in this thesis.

We chose to work with the CARMA formal modelling language, which means that one must think of the modelled systems under the Markovian assumption, i.e. the future of the system's evolution depends only on its current state and the transitions have rates described by the exponential distribution.

Our extension of CASL, Spatial CASL, although providing structures much more suitable for representing spatial elements, is not capable of representing all aspects of physical space in a realistic way. One aspect that is not considered in Spatial CASL is the physical size of the elements. The locations in Spatial CASL are points on a plane and do not have a size of their own. They also have infinite capacity for co-locating agents (and there is also no concept of an agent's physical size). Areas in Spatial CASL are defined as collection of nodes and not continuous fields, and are only labelled with a name. There are a number of situations in which the ability to define a continuous area with features described by functions depending on the location within the area would be advantageous. For example, one can think of a spatial field with features whose values change smoothly over the whole area, in a gradient-like fashion.

Using Spatial CASL and CGP, one can construct a spatial graph that is non-planar, i.e. cannot be represented on a plane with no connections crossing (see graphs from Chapter 5 in Fig. 5.3). This is impossible in real-world scenarios without adding a third dimension representing elevation (one can think of tunnels and bridges). Such three dimentional spatial structures cannot currently be represented by CGP models.

The tool for working with CASL code was developed as an extension to the Eclipse IDE and the simulator was written in Java. Without a doubt the largest limitation of the CARMA Eclipse plugin we encountered is that at the time of writing, CASL does not support reading input data files and only implements a small subset of standard data structures such as vectors and arrays (but not, for example, dictionaries). This makes

working with models based on data extremely difficult, since all the data one wants to include in a model has to be hard-coded in the CASL file itself. With large data this yields large CASL code files, which the Eclipse parser was not designed to deal with. Most CASL files with the size in the order of magnitude of 10k+ lines are simply too large for the parser to handle and result in long periods of unresponsiveness or crashes of the software. In addition to that the implementation of CASL is of a rather preliminary nature and not robust in terms of translating CARMA models into Java, and will often result in Java programs that crash with exceptions without providing any logs or a way of attaching a debugger during a simulation run (in order to do that one would need to check out the repository of the CASL implementation itself, rather than install it as a Eclipse plugin).

The work presented in Chapter 7 brings this model closer to reality but it is still highly abstract and because of the lack of availability of data we could not compare its performance to that of the real-world system, in order to assess its feasibility as a predictive tool.

During the work on the model presented in Chapter 8 we pushed the CARMA Eclipse plugin to its limits, learning how much data it is feasible to encode in the model itself and still allow for the Java code to be generated and the simulation to finish within a reasonable amount of time. From the code generation perspective, it would have not been difficult to generate a CARMA model including all the buses and routes in the network, but it is simply not possible to simulate such a large and complex CARMA model currently.

9.3 Further work

There is a lot of potential for future work in relation to the work presented in this thesis. Specifically, the following future work projects could be identified:

- Extending Spatial CASL with the ability to handle more data structures (especially spatial ones, such as continuous gradient fields mentioned in the previous

section) and with the ability to read input data files.

- Making CGP more robust, adding the ability to import graphical arrangements of elements from data files and adding a graphical editor for modifying the behaviour of components.

- Creating a pedestrian routing model heavily based on a real-world scenario and comparing the simulation results with measures applied to real data.

- Creating a model of a bus network that includes all buses and routes in the system, and takes into account the existence of (or lack of) special privileged bus lanes, as well as traffic information based on location (as some areas are more likely to be congested than others).

Chapter 10

List of Abbreviations

Abbreviations

ADLs Architecture Description Languages

API Application Programming Interface

AVL Automatic Vehicle Location

BPD Business Process Diagram

BPMN Business Process Model and Notation

CARMA Collective Adaptive Resource-Sharing Markovian Agents

CAS Collective Adaptive Systems

CASL CARMA Specification Language

CGP CARMA Graphical Plugin

DERs Distributed Energy Resources

GPS Global Positioning System

GUI Graphical User Interface

IDEF Integration DEFinition

SIR Susceptible Infected Recovered

UML Unified Modelling Language

UTM Universal Transverse Mercator (projection)

WGS84 World Geodic System 1984

Bibliography

[1] *ABACUS — Avolution Software*. https://www.avolutionsoftware.com/. [Online; accessed 16-July-2018].

[2] *ISO 5807:1985 Information processing – Documentation symbols and conventions for data, program and system flowcharts, program network charts and system resources charts*. https://www.iso.org/standard/11955.html, 1985. [Online; accessed 16-July-2018].

[3] *NIMA Technical Report TR8350.2, Department of Defense World Geodetic System 1984, Its Definition and Relationships With Local Geodetic Systems*. http://www.earth-info.nga.mil/GandG/publications/tr8350.2/tr8350_2.html, 1997. [Online; accessed 16-July-2018].

[4] *Collective Adaptive Systems. Expert Consultation Workshop*. http://cordis.europa.eu/pub/fp7/ict/docs/fet-proactive/shapefetip-wp2011-12-02_en.pdf, 2009. [Online; accessed 02-March-2019].

[5] *Business Process Model and Notation Specification – Version 2.0*. https://www.omg.org/spec/BPMN/2.0/, 2011. [Online; accessed 16-July-2018].

[6] *ISO/IEC/IEEE 42010:2011 Systems and software engineering – Architecture description*. http://www.iso-architecture.org/42010/, 2011. [Online; accessed 16-July-2018].

[7] *Build Your Own ADL*.
http://byadl.di.univaq.it/, 2012. [Online; accessed 16-July-2018].

[8] *The Mindful Programmer: Geographic distance can be simple and fast*.
http://jonisalonen.com/2014/computing-distance-between-coordinates-can-be-simple-and-fast/, 2014. [Online; accessed 16-July-2018].

[9] *The City Of Edinburgh Council — Bike counter data cluster Midle Meadow Walk*.
https://data.edinburghopendata.info/dataset/bike-counter-data-set-cluster/resource/ffbeb785-a19f-4788-bc27-892b024a6750?view_id=6526a819-3578-43d7-afe1-71be4b7b5db5, 2018. [Online; accessed 16-July-2018].

[10] *Cycle Counter Statistics — Bicycle Counter*.
http://www.bicyclecounter.dk/BicycleCounter/BC_Statistics.jsp, 2018. [Online; accessed 16-July-2018].

[11] *Edinburgh 'overcrowded with tourists' during festivals. — STV*.
https://stv.tv/news/east-central/1406879-edinburgh-s-pavements-overcrowded-during-festivals/, 2018. [Online; accessed 16-July-2018].

[12] *Google maps*.
https://maps.google.com, 2018. [Online; accessed 16-July-2018].

[13] *Live Cycle Counter Data — Falco UK Ltd*.
http://www.falco.co.uk/products/live-cycle-counter-data/, 2018. [Online; accessed 16-July-2018].

[14] *Lothian Buses*.
http://lothianbuses.co.uk/, 2018. [Online; accessed 16-July-2018].

[15] *Lothian Buses Network Maps*.
https://www.lothianbuses.com/maps-and-times/network-maps/, 2018. [Online; accessed 15-March-2019].

[16] *Tom Tom Traffic Flow — Edinburgh*.

http://www.tomtom.com/en_gb/traffic-news/edinburgh-traffic/traffic-flow/, 2018. [Online; accessed 16-July-2018].

[17] *Tom Tom Traffic News — Edinburgh*. http://www.tomtom.com/en_gb/traffic-news/, 2018. [Online; accessed 16-July-2018].

[18] *Transport for Edinburgh Ltd.* http://transportforedinburgh.com/, 2018. [Online; accessed 16-July-2018].

[19] *HERE Traffic*. https://www.here.com/products/traffic-solutions, 2019. [Online; accessed 15-March-2019].

[20] M. BACKES, S. LORENZ, M. MAFFEI, AND K. PECINA, *The CASPA tool: Causality-based abstraction for security protocol analysis*, in Computer Aided Verification, A. Gupta and S. Malik, eds., Berlin, Heidelberg, 2008, Springer Berlin Heidelberg, pp. 419–422.

[21] C. BAIER, B. HAVERKORT, H. HERMANNS, AND J.-P. KATOEN, *Model-checking algorithms for continuous-time Markov chains*, IEEE Transactions on software engineering, 29 (2003), pp. 524–541.

[22] N. BELLOMO AND A. BELLOUQUID, *On multiscale models of pedestrian crowds from mesoscopic to macroscopic*, Communications in Mathematical Sciences, 13 (2015), pp. 1649–1664.

[23] N. BELLOMO AND C. DOGBE, *On the modeling of traffic and crowds: A survey of models, speculations, and perspectives*, SIAM review, 53 (2011), pp. 409–463.

[24] L. BORTOLUSSI, R. DE NICOLA, V. GALPIN, S. GILMORE, J. HILLSTON, D. LATELLA, M. LORETI, AND M. MASSINK, CARMA *Collective Adaptive Resource-sharing Markovian Agents*, in Proceedings of QAPL 2015, vol. 194 of EPTCS, 2015, pp. 16–31.

[25] E. CARSON, *10 rideshare apps to crowdsource your commute*.

https://www.techrepublic.com/article/10-rideshare-apps-to-crowdsource-your-commute/, 2014. [Online; accessed 02-March-2019].

[26] C. CHEN, D. ZHANG, Z. ZHOU, N. LI, T. ATMACA, AND S. LI, *B-planner: Night bus route planning using large-scale taxi GPS traces*, in 2013 IEEE International Conference on Pervasive Computing and Communications (PerCom), March 2013, pp. 225–233.

[27] F. CIOCCHETTA AND J. HILLSTON, *Bio-PEPA: A framework for the modelling and analysis of biological systems*, Theor. Comput. Sci., 410 (2009), pp. 3065–3084.

[28] K. C. CLARKE, *Advances in geographic information systems*, Computers, environment and urban systems, 10 (1986), pp. 175–184.

[29] A. CORBETTA, J. MEEUSEN, C. LEE, AND F. TOSCHI, *Continuous measurements of real-life bidirectional pedestrian flows on a wide walkway*, in Proceedings of Pedestrian and Evacuation Dynamics 2016, 2016, pp. 18–24.

[30] R. L. CREIGHTON, *Urban transportation planning*, tech. rep., 1970.

[31] E. DAVID MORGAN, *Trail pheromones of ants*, Physiological Entomology, 34 (2009), pp. 1–17.

[32] R. DE NICOLA, D. LATELLA, M. LORETI, AND M. MASSINK, *A uniform definition of stochastic process calculi*, ACM Computing Surveys, 46 (2013).

[33] M. J. DE SMITH, M. F. GOODCHILD, AND P. LONGLEY, *Geospatial analysis: a comprehensive guide to principles, techniques and software tools*, Troubador Publishing Ltd, 2007.

[34] C. DEHNERT, S. JUNGES, J.-P. KATOEN, AND M. VOLK, *A storm is coming: A modern probabilistic model checker*, in International Conference on Computer Aided Verification, Springer, 2017, pp. 592–600.

[35] P. DEMAIO, *Bike-sharing: History, impacts, models of provision, and future*, Journal of public transportation, 12 (2009), p. 3.

[36] G. DI CARO, *Ant colony optimization and its application to adaptive routing in telecommunication networks*, (2004).

[37] N. J. DINGLE AND W. J. KNOTTENBELT, *Automated customer-centric performance analysis of generalised stochastic Petri nets using tagged tokens*, Electronic Notes in Theoretical Computer Science, 232 (2009), pp. 75 – 88. Proceedings of the Third International Workshop on the Practical Application of Stochastic Modelling (PASM 2008).

[38] R. ERBAN AND S. CHAPMAN, *Stochastic modelling of reaction-diffusion processes: algorithms for bimolecular reactions*, Physical Biology, 6 (2009).

[39] D. FANGE, O. BERG, P. SJÖBERG, AND J. ELF, *Stochastic reaction-diffusion kinetics in the microscopic limit*, Proceedings of the National Academy of Sciences, 107 (2010), pp. 19820–19825.

[40] E. FERGUSON, *The rise and fall of the American carpool: 1970–1990*, Transportation, 24 (1997), pp. 349–376.

[41] R. FERNÁNDEZ, *Modelling public transport stops by microscopic simulation*, Transportation Research Part C: Emerging Technologies, 18 (2010), pp. 856 – 868. Special issue on Transportation Simulation Advances in Air Transportation Research.

[42] J. L. FERNANDEZ-MARQUEZ, G. DI MARZO SERUGENDO, S. MONTAGNA, M. VIROLI, AND J. L. ARCOS, *Description and composition of bio-inspired design patterns: a complete overview*, Natural Computing, 12 (2013), pp. 43–67.

[43] A. FERSCHA, *Collective Adaptive Systems*, in Adjunct Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2015 ACM International Symposium on Wearable Computers, UbiComp/ISWC'15 Adjunct, New York, NY, USA, 2015, ACM, pp. 893–895.

[44] M. FOWLER AND K. SCOTT, *UML Distilled (2nd Ed.): A Brief Guide to the Standard Object Modeling Language*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2000.

[45] C. FRICKER AND N. GAST, *Incentives and redistribution in homogeneous bike-sharing systems with stations of finite capacity*, Euro journal on transportation and logistics, 5 (2016), pp. 261–291.

[46] V. GALPIN, *Modelling residential smart energy schemes*, in Eighth IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshops, SASOW 2014, London, United Kingdom, September 8-12, 2014, IEEE Computer Society, 2014, pp. 49–54.

[47] ——, *Modelling ambulance deployment with* CARMA, in Coordination Models and Languages, A. Lafuente and J. Proenca, eds., Lecture Notes in Computer Science, Springer International Publishing, 2016, pp. 1–16.

[48] V. GALPIN, A. GEORGOULAS, S. GILMORE, J. HILLSTON, D. LATELLA, M. LORETI, M. MASSINK, AND N. ZOŃ, *Quanticol deliverable 4.3: CaSL at work*, 2017. http://blog.inf.ed.ac.uk/quanticol/files/2017/03/Deliverable-D43.pdf.

[49] V. GALPIN, A. GEORGOULAS, M. LORETI, AND A. VANDIN, *Statistical analysis of CARMA models: an advanced tutorial*, in 2018 Winter Simulation Conference (WSC), IEEE, 12 2018, pp. 395–409.

[50] V. GALPIN, N. ZOŃ, P. WILSDORF, AND S. GILMORE, *Mesoscopic modelling of pedestrian movement using* CARMA *and its tools*, ACM Trans. Model. Comput. Simul., 28 (2018), pp. 11:1–11:26.

[51] L. GARFIELD, *13 cities that are starting to ban cars — Business Insider*. https://www.businessinsider.com/cities-going-car-free-ban-2017-8?IR=T, 2018. [Online; accessed 16-July-2018].

[52] A. GEORGOULAS, J. HILLSTON, D. MILIOS, AND G. SANGUINETTI, *Probabilistic programming process algebra*, in Quantitative Evaluation of Systems: 11th International Conference, QEST 2014, Florence, Italy, September 8-10, 2014. Proceedings, G. Norman and W. Sanders, eds., Springer, 2014, pp. 249–264.

[53] S. GOSS, S. ARON, J. DENEUBOURG, AND J. PASTEELS, *Self-organized shortcuts in the argentine ant*, Naturwissenschaften, 76 (1989), pp. 579–581.

[54] Y. HAIRONG AND L. DAYONG, *Optimal regional bus timetables using improved genetic algorithm*, in Intelligent Computation Technology and Automation, 2009. ICICTA'09. Second International Conference on, vol. 3, IEEE, 2009, pp. 213–216.

[55] S. HASSOLD AND A. CEDER, *Multiobjective approach to creating bus timetables with multiple vehicle types*, Transportation Research Record: Journal of the Transportation Research Board, (2012), pp. 56–62.

[56] D. HELBING, L. BUZNA, A. JOHANSSON, AND T. WERNER, *Self-organized pedestrian crowd dynamics: Experiments, simulations, and design solutions*, Transportation Science, 39 (2005), pp. 1–24.

[57] D. HELBING, I. FARKÁS, P. MOLNAR, AND T. VICSEK, *Simulation of pedestrian crowds in normal and evacuation situations*, in Pedestrian and evacuation dynamics, M. Schreckenberg and S. D. Sharma, eds., Springer, Berlin, 2002, pp. 21–58.

[58] J. HILLSTON, *PEPA: Performance enhanced process algebra*, University of Edinburgh, Department of Computer Science, 1993.

[59] J. HILLSTON AND M. LORETI, *CARMA eclipse plug-in: A tool supporting design and analysis of collective adaptive systems*, in Quantitative Evaluation of Systems - 13th International Conference, QEST 2016, Quebec City, QC, Canada, August 23-25, 2016, Proceedings, G. Agha and B. V. Houdt, eds., vol. 9826 of Lecture Notes in Computer Science, Springer, 2016, pp. 167–171.

[60] J. HILLSTON, J. PITT, M. WIRSING, AND F. ZAMBONELLI, *Collective Adaptive Systems: Qualitative and Quantitative Modelling and Analysis (Dagstuhl Seminar 14512)*, Dagstuhl Reports, 4 (2015), pp. 68–113.

[61] M. HINCHEY, R. STERRITT, AND C. ROUFF, *Swarms and swarm intelligence*, Computer, 40 (2007), pp. 111–113.

[62] O. HOLLAND AND C. MELHUISH, *Stigmergy, self-organization, and sorting in collective robotics*, Artificial life, 5 (1999), pp. 173–202.

[63] Z. HUSSAIN, N. GROMOV, AND I. TODORAN, *SOA Integration Modeling: An Evaluation of How SoaML Completes UML Modeling*, in 2011 15th IEEE International Enterprise Distributed Object Computing Conference Workshops(EDOCW), vol. 00, 08 2011, pp. 57–66.

[64] A. IBEAS, L. DELLÂ OLIO, B. ALONSO, AND O. SAINZ, *Optimizing bus stop spacing in urban areas*, Transportation Research Part E: Logistics and Transportation Review, 46 (2010), pp. 446 – 458.

[65] J.-P. KATOEN, *The probabilistic model checking landscape*, in Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, ACM, 2016, pp. 31–45.

[66] S. KERNBACH, T. SCHMICKL, AND J. TIMMIS, *Collective Adaptive Systems: Challenges beyond evolvability*, CoRR, abs/1108.5643 (2011).

[67] L. KLEINROCK, *Queueing Systems*, vol. I: Theory, Wiley Interscience, 1975.

[68] M. KUNTZ, M. SIEGLE, AND E. WERNER, *Symbolic performance and dependability evaluation with the tool CASPA*, in Applying Formal Methods: Testing, Performance and M/ECommerce, FORTE 2004 Workshops The FormEMC, EPEW, ITM, Toledo, Spain, October 1-2, 2004, M. Núñez, Z. Maamar, F. Pelayo, K. Pousttchi, and F. Rubio, eds., vol. 3236 of LNCS, Springer, 2004, pp. 293–307.

[69] M. KWIATKOWSKA, G. NORMAN, AND D. PARKER, *Prism 4.0: Verification of probabilistic real-time systems*, in International conference on computer aided verification, Springer, 2011, pp. 585–591.

[70] J. LARSEN, *Bike-sharing programs hit the streets in over 500 cities worldwide*, Earth Policy Institute, (2013).

[71] A. LEGAY, B. DELAHAYE, AND S. BENSALEM, *Statistical model checking: An overview*, in International conference on runtime verification, Springer, 2010, pp. 122–135.

[72] F. LIBRINO, M. E. RENDA, G. RESTA, P. SANTI, F. DUARTE, C. RATTI, AND J. ZHAO, *Social mixing and home–work carpooling*, tech. rep., 2018.

[73] W. LIU AND A. F. WINFIELD, *Modeling and optimization of adaptive foraging in swarm robotic systems*, The International Journal of Robotics Research, 29 (2010), pp. 1743–1760.

[74] M. LORETI AND J. HILLSTON, *Modelling and Analysis of Collective Adaptive Systems with* CARMA *and its Tools*, in Formal Methods for the Quantitative Evaluation of Collective Adaptive Systems - 16th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM 2016, Bertinoro, Italy, June 20-24, 2016, Advanced Lectures,

M. Bernardo, R. D. Nicola, and J. Hillston, eds., vol. 9700 of Lecture Notes in Computer Science, Springer, 2016, pp. 83–119.

[75] I. MAHMOOD, M. HARIS, AND H. SARJOUGHIAN, *Analyzing emergency evacuation strategies for mass gatherings using crowd simulation and analysis framework: Hajj scenario*, in Proceedings of the 2017 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation, SIGSIM-PADS '17, New York, NY, USA, 2017, ACM, pp. 231–240.

[76] M. MASSINK AND D. LATELLA, *Fluid analysis of foraging ants*, Lecture Notes in Computer Science, 7274 (2012), pp. 152–165.

[77] P. MINETT AND J. PEARCE, *Estimating the energy consumption impact of casual carpooling*, Energies, 4 (2011), pp. 126–139.

[78] M. MOUSSAÏD, D. HELBING, AND G. THERAULAZ, *How simple rules determine pedestrian behavior and crowd disasters*, Proceedings of the National Academy of Sciences, 108 (2011), pp. 6884–6888.

[79] O. NORAN, *UML vs IDEF: An Ontology-oriented Comparative Study in View of Business Modelling*, (2005).

[80] E. PAPADIMITRIOU, G. YANNIS, AND J. GOLIAS, *A critical assessment of pedestrian behaviour models*, Transportation research part F: Traffic Psychology and Behaviour, 12 (2009), pp. 242–255.

[81] J. L. PETERSON, *Petri nets*, ACM Comput. Surv., 9 (1977), pp. 223–252.

[82] S. D. REICHER, *The psychology of crowd dynamics*, vol. 44, Blackwell handbook of social psychology: Group processes, 2001.

[83] D. REIJSBERGEN, *Probabilistic modelling of station locations in bicycle-sharing systems*, in Software Technologies: Applications and Foundations, P. Milazzo, D. Varró, and M. Wimmer, eds., Cham, 2016, Springer International Publishing, pp. 83–97.

[84] D. REIJSBERGEN AND S. GILMORE, *Formal punctuality analysis of frequent bus services using headway data*, in Computer Performance Engineering - 11th European Workshop, EPEW 2014, Florence, Italy, 1112 September 2014, A. Horváth and K. Wolter, eds., Springer International Publishing, pp. 164–178.

[85] C. W. REYNOLDS, *Flocks, herds and schools: A distributed behavioral model*, in ACM SIGGRAPH computer graphics, vol. 21, ACM, 1987, pp. 25–34.

[86] J. M. RIBÓ AND X. FRANCH, *Building Expressive and Flexible Process Models Using a UML-Based Approach*, in Software Process Technology, V. Ambriola, ed., Berlin, Heidelberg, 2001, Springer Berlin Heidelberg.

[87] C. ROBERT, L. GRUNSKE, AND W. KIRSTEN, *Probabilistic Timed Behavior Trees*, Integrated Formal Methods. IFM 2007. Lecture Notes in Computer Science, vol 4591. Springer, Berlin, Heidelberg, (2007).

[88] C. SHETH, K. TRIANTIS, AND D. TEODOROVĆ, *Performance evaluation of bus routes: A provider and passenger perspective*, Transportation Research Part E: Logistics and Transportation Review, 43 (2007), pp. 453 – 478.

[89] F. SIMPSON, *Tourist Impact in the Historic Centre of Prague: Resident and Visitor Perceptions of the Historic Built Environment*, The Geographical Journal, 165 (1999), pp. 173–183.

[90] C. F. SNOOK AND M. J. BUTLER, *Using a graphical design tool for formal specification*, in Proceedings of the 13th Annual Workshop of the Psychology of Programming Interest Group, PPIG 2001, Bournemouth, UK, April 17-20, 2001, Psychology of Programming Interest Group, 2001, p. 24.

[91] J. SNYDER, *Map Projections: A Working Manual*, Professional paper: United States Geological Survey, U.S. Government Printing Office, 1994.

[92] M. SOUTHWORTH, *Designing the walkable city*, Journal of urban planning and development, 131 (2005), pp. 246–257.

[93] F. STEIMANN AND H. VOLLMER, *Exploiting practical limitations of UML diagrams for model validation and execution*, Software & Systems Modeling, 5 (2006), pp. 26–47.

[94] D. P. STORMONT, *Autonomous rescue robot swarms for first responders*, in CIHSPS 2005. Proceedings of the 2005 IEEE International Conference on Computational Intelligence for Homeland Security and Personal Safety, 2005., March 2005, pp. 151–157.

[95] M. TER BEEK, A. FANTECHI, AND S. GNESI, *Challenges in Modelling and*

Analyzing Quantitative Aspects of Bike-Sharing Systems, Lecture Notes in Computer Science, 8802 (2014), pp. 351–367.

[96] D. THALMANN, *Crowd Simulation*, American Cancer Society, 2007.

[97] P. A. THOMPSON AND E. W. MARCHANT, *A computer model for the evacuation of large building populations*, Fire safety journal, 24 (1995), pp. 131–148.

[98] J. TOLUJEW AND F. ALCALÁ, *A mesoscopic approach to modeling and simulation of pedestrian traffic flows*, in Proceedings of the 18th European Simulation Multiconference (ESM2004), 2004, pp. 123–128.

[99] A. TOVEY AND M. LEWIS, *High-frequency trading: when milliseconds mean millions — The Telegraph*. `https://www.telegraph.co.uk/finance/newsbysector/banksandfinance/10736960/High-frequency-trading-when-milliseconds-mean-millions.html`, 2014. [Online; accessed 16-July-2018].

[100] M. TRIBASTONE, A. DUGUID, AND S. GILMORE, *The PEPA Eclipse Plugin*, Performance Evaluation Review, 36(4) (2009), pp. 28–33.

[101] A. TRIVEDI AND M. PANDEY, *Agent-based modelling and simulation of religious crowd gatherings in India*, in Advanced Computational and Communication Paradigms, S. Bhattacharyya, N. Chaki, D. Konar, U. K. Chakraborty, and C. T. Singh, eds., Singapore, 2018, Springer Singapore, pp. 465–472.

[102] A. E. TURGUT, H. ÇELIKKANAT, F. GÖKÇE, AND E. ŞAHIN, *Self-organized flocking in mobile robot swarms*, Swarm Intelligence, 2 (2008), pp. 97–120.

[103] U. C. B. U.S. DEPARTMENT OF COMMERCE, *2009 American community survey (washington, dc)*, tech. rep., 2010.

[104] A. VANDIN AND S. SEBASTIO, *MultiVeStA: Statistical model checking for discrete event simulators*, 2014.

[105] L. L. VISSAT, A. CLARK, AND S. GILMORE, *Finding optimal timetables for Edinburgh bus routes*, Electronic Notes in Theoretical Computer Science, 310

(2015), pp. 179 – 199. Proceedings of the Seventh International Workshop on the Practical Application of Stochastic Modelling (PASM).

[106] A. WALKER, *Pedestrian power to shape future cities — BBC*. http://www.bbc.com/future/story/20131018-walk-to-work-transform-your-city, 2013. [Online; accessed 16-July-2018].

[107] K. WINTER, *Formalising Behaviour Trees with CSP*, in Integrated Formal Methods, E. A. Boiten, J. Derrick, and G. Smith, eds., Berlin, Heidelberg, 2004, Springer Berlin Heidelberg, pp. 148–167.

[108] W. YUAN AND K. H. TAN, *A model for simulation of crowd behaviour in the evacuation from a smoke-filled compartment*, Physica A: Statistical Mechanics and its Applications, 390 (2011), pp. 4210–4218.

[109] F. ZAMBONELLI, *Engineering self-organizing urban superorganisms*, Engineering Applications of Artificial Intelligence, 41 (2015), pp. 325–332.

[110] B. P. ZEIGLER, T. G. KIM, AND H. PRAEHOFER, *Theory of Modeling and Simulation*, Academic Press, Inc., Orlando, FL, USA, 2nd ed., 2000.

[111] D. ZHANG, T. HE, Y. LIU, AND J. A. STANKOVIC, *CallCab: A Unified Recommendation System for Carpooling and Regular Taxicab Services*, in 2013 IEEE International Conference on Big Data, Oct 2013, pp. 439–447.

[112] N. ZON, V. GALPIN, AND S. GILMORE, *Modelling movement for collective adaptive systems with* CARMA, in Proceedings of the Workshop on FORmal methods for the quantitative Evaluation of Collective Adaptive SysTems, FORECAST@STAF 2016, Vienna, Austria, 8 July 2016., M. ter Beek and M. Loreti, eds., vol. 217 of EPTCS, 2016, pp. 43–52.

[113] N. ZON AND S. GILMORE, *Data-driven modelling and simulation of urban transportation systems using Carma*, in Proceedings of International Symposium On Leveraging Applications of Formal Methods, Verification and Validation (ISOLA) 2018, Springer, 2018.

[114] N. ZOŃ, S. GILMORE, AND J. HILLSTON, *Rigorous graphical modelling of movement in collective adaptive systems*, in Leveraging Applications of Formal Methods, Verification and Validation: Foundational Techniques - 7th

International Symposium, ISoLA 2016, Imperial, Corfu, Greece, October 10-14, 2016, Proceedings, Part I, T. Margaria and B. Steffen, eds., vol. 9952 of Lecture Notes in Computer Science, 2016, pp. 674–688.

[115] V. ZYRYANOV AND A. MIRONCHUK, *Simulation study of intermittent bus lane and bus signal priority strategy*, Procedia-Social and Behavioral Sciences, 48 (2012), pp. 1464–1471.