A software framework for the microscopic modelling of pedestrian movement

Robert Kukla

A thesis submitted in partial fulfilment of the requirements of Napier University for the degree of Doctor of Philosophy

This research programme was carried out in collaboration with the Transport Research Institute

June 2007
Abstract

A town planner, faced with the task of designing attractive walking spaces, needs a tool that will allow different designs to be compared in terms of their attractiveness as well as their effectiveness. PEDFLOW is an attempt to create such a tool. It is an agent-based, microscopic model of pedestrian flow where virtual pedestrians navigate a virtual environment. On their way towards a goal the agents, representing pedestrians, interact with features of the environment and with other agents. The microscopic, rule-based actions result in an emergent behaviour that mimics that of real pedestrians.

Pedestrians are subjected to a multitude of influences when walking. The majority of existing models only focus on a single aspect, typically the avoidance of obstructions or other pedestrians. PEDFLOW uses an implementation of context-mediated behaviour to enable the agents to deal with multiple cause-effect relations in a well-defined and flexible yet highly efficient manner. A variety of mobile and immobile entities can be modelled by objects in an object-oriented environment. The model is informed by an empirical study of pedestrian behaviour and the parameters of the agents are derived from measures of observed pedestrian movement.

PEDFLOW’s suitability for pedestrian modelling in the described context is evaluated in both qualitative and quantitative terms. Typical macroscopic movement patterns from the real world such as "platooning" and "walking with a partner" are selected and the corresponding emergent model behaviours investigated. Measures of service (MOS) are defined and extracted from the model for comparison with real world measures. As PEDFLOW was created as an interactive tool to be used in an office environment rather than in a high performance lab, the scalability and performance limitations are explored with regards to the size of the modelled area, the number of modelled pedestrians and the complexity of the interactions between them. It is shown that PEDFLOW can be a useful tool in the urban design process.
Contents

Abstract...............................................................................................................................i
Contents.................................................................................................................................. ii
List of Figures......................................................................................................................... iv
List of Tables......................................................................................................................... vi
Acknowledgements ............................................................................................................... vii
Contributing papers.............................................................................................................. viii
1 Introduction .......................................................................................................................1
2 Modelling pedestrian movement.....................................................................................6
  2.1 Computer models and computer simulations.............................................................6
  2.2 Autonomous agents in modelling..............................................................................9
  2.3 Evaluation of existing pedestrian models.................................................................15
    2.3.1 Cellular Automata model (Blue and Adler).............................................................15
    2.3.2 Agent based CA model (Dijkstra)...............................................................................16
    2.3.3 STREETS (Schelhorn, O’Sullivan)............................................................................17
    2.3.4 ROBO SOCCER (Stone)............................................................................................19
    2.3.5 Discrete Force Model for pedestrian motion (Helbing)..........................................21
    2.3.6 SimPed (Daamen).....................................................................................................22
    2.3.7 Nomad (Hoogendoorn).............................................................................................23
    2.3.8 PEDFLOW1 (Kerridge).............................................................................................25
    2.3.9 Legion (Kagarlis).......................................................................................................27
    2.3.10 PedGo (Kuepfe1).....................................................................................................29
    2.3.11 EXODUS (Galea).....................................................................................................30
    2.3.12 Model feature overview.........................................................................................32
3 Creating a flexible and efficient design............................................................................35
  3.1 Environment modelling...............................................................................................35
    3.1.1 Entities.....................................................................................................................35
    3.1.2 Movement...............................................................................................................37
  3.2 Behaviour modelling.....................................................................................................40
    3.2.1 Agent activation......................................................................................................40
    3.2.2 Cause-effect rules..................................................................................................42
    3.2.3 Context-mediated behaviour..................................................................................43
4 Building the prototype application....................................................................................50
  4.1 Implementation aspects...............................................................................................50
    4.1.1 Programming language............................................................................................50
    4.1.2 Modelling framework..............................................................................................52
    4.1.3 Concurrency............................................................................................................53
    4.1.4 Model dynamics......................................................................................................55
    4.1.5 Modelled entities.....................................................................................................58
    4.1.6 Data storage............................................................................................................59
  4.2 Agent characteristics of PEDFLOW entities...............................................................66
  4.3 Obtaining input values for the PEDFLOW model.......................................................70
    4.3.1 Filming, sampling and calibration..........................................................................72
    4.3.2 Tracking individual pedestrians............................................................................73
    4.3.3 Informing the PEDFLOW model..........................................................................75
List of Figures

Figure 2.1 Bi-directional pedestrian flows visualised in a Java applet at [32] ......................... 15
Figure 2.2 Cellular Automata grid and visualised pedestrian movement in [36] ..................... 16
Figure 2.3 Buildings, way-points and pedestrians in STREETS in [37] ............................. 17
Figure 2.4 RoboCup Simulator in [40] ............................................................................. 19
Figure 2.5 Modelling pedestrian interaction at a crossing with the DFM. ...................... 21
Figure 2.6 Screenshot from the NOMAD model (Schiphol Plaza simulation) in [47] .......... 24
Figure 2.7 Offline visualiser for PEDFLOW1. ................................................................. 27
Figure 2.8 Example of Legion output for average delay (London Docklands Light Rail) [54] . 28
Figure 2.9 Screenshot of PedGo, simulating the evacuation on a cruise ship [57] .......... 29
Figure 2.10 buildingEXODUS: density display during the early stages of evacuation. [59] ... 31
Figure 3.1 Levels of abstraction in a scenario with building, two pedestrians and a bin .... 37
Figure 3.2 Using footsteps to identify the route of a person. ........................................... 38
Figure 3.3 Agent affected by two contexts .................................................................. 45
Figure 3.4 Adding an additional context (attractor). ..................................................... 46
Figure 3.5 "Walking with a partner" ............................................................................. 49
Figure 4.1 Modular structure of the PEDFLOW Model .................................................. 52
Figure 4.2 Classes in PEDFLOW (dashed boxes indicate derived sub classes) ................. 53
Figure 4.3 Timing of activation and display thread ....................................................... 57
Figure 4.4 Model time compression with timeslots ..................................................... 57
Figure 4.5 Class inheritance for entities in PEDFLOW ................................................. 58
Figure 4.6 Use of the parameter object to store information in PEDFLOW .................... 62
Figure 4.7 Data Flow in PEDFLOW ......................................................................... 66
Figure 4.8 Calibrating the tracking software by matching real-world to image co-ordinates. .. 72
Figure 4.9 Semi-automatic tracking of pedestrians using image analysis software .......... 73
Figure 4.10 Software to visualise the trajectory of pedestrians and extract measures ......... 74
Figure 4.11 The application’s main window ................................................................. 77
Figure 4.12 File menu (left) and directory structure (right) ............................................ 77
Figure 4.13 Grid editor interface ............................................................................... 78
Figure 4.14 Entity editor while editing a simple entity (blockage) .................................... 79
Figure 4.15 Entity selector of the grid editor .............................................................. 80
Figure 4.16 Preference window .............................................................................. 81
Figure 4.17 Visualisation of 'hotspots' .................................................................. 82
Figure 4.18 Tracking features in PEDFLOW: path, rubber band and observation area ....... 82
Figure 4.19 Simulation toolbar (left) and simulation menu (right) ............................... 83
Figure 4.20 RunTo dialog box to control timed model execution ................................. 83
Figure 5.1 Example of clustered movement of pedestrians (platooning) ...................... 86
Figure 5.2 Emergent platooning in PEDFLOW ........................................................... 87
Figure 5.3 Local, short platooning in high agent density .......................................... 88
Figure 5.4 Couple walking towards a lamp post. .................................................................89
Figure 5.5 Agent pairs rejoining after having circumvented two obstacles...................90
Figure 5.6 PEDFLOW agents at a modelled pedestrian crossing. .....................................91
Figure 5.7 Pedestrian crossing in PEDFLOW: a) build-up, b) crossing, c) dispersion ......92
Figure 5.8 Artist with audience (C), attracted people (B) and non-interested passer-by (A)...94
Figure 5.9 Simulation of an attractor in PEDFLOW. .........................................................95
Figure 5.10 Orderly queue at a British bus stop. .............................................................96
Figure 5.11 PEDFLOW application with a bus stop simulation. .......................................97
Figure 5.12 Queuing behaviour at an ATM: a) milling b) environmental and c) straight [106].98
Figure 5.13 Setting up the model for the collection of microscopic measures. .............102
Figure 5.14 Average relative delay over density for different numbers of obstructions ....104
Figure 5.15 Distribution of relative delay for different agent densities.............................104
Figure 5.16 Speed/density diagram for High Petergate simulation. .................................106
Figure 5.17 Speed/density diagrams in the literature [109]. ............................................106
Figure 5.18 Basic experiment setup to determine performance in a worst case scenario. ...113
Figure 5.19 Collecting internal measure from model runs. .............................................114
Figure 5.20 Average activation time in relation to agent influx. ......................................116
List of Tables

Table 2.1 Typical agent properties from [23] .................................................................................. 10
Table 2.2 Overview over the reviewed models classified according to section 2.2 ....................... 34
Table 3.1 Possible speed representations for different space/time resolutions ....................... 40
Table 4.1 Simplified example for entity storage as flat file.......................................................... 60
Table 4.2 Example PEDFLOW XML file with embedded DTD. .................................................... 64
Table 4.3 Macro commands and corresponding menu options.................................................. 84
Table 5.1 Possible interpretation of microscopic measures. ......................................................... 101
Table 5.2 Java version used during the performance evaluation. .............................................. 109
Table 5.3 Example macro for a single simulation run................................................................. 110
Table 5.4 Total execution time, number of activations and derived average execution time. 116
Acknowledgements

The PEDFLOW research project was funded by the UK-EPSRC grant reference GQ/M59792. It was undertaken in collaboration with the City Councils of York and Edinburgh and the consultancy Halcrow Fox, UK. I would like to express my gratitude to my colleagues on the project: Jon Kerridge, Alexandra Willis, Nathalia Gjersoe and Catriona Havard.

The PhD was financially supported by the School of Computing of Napier University. Many thanks go to my supervisors Jon Kerridge and Jessie Kennedy for their support and continuous encouragement.

I would also like to thank Mareike Bethge and Sara Northcott for helping with the proof-reading of the thesis.
Contributing papers


1 Introduction

Strategic town planning has become a requirement in today’s urban design process in order to avoid the creation of routes that will become bottlenecks in the future. With the increase of population in urban areas it becomes more and more important to make effective use of available space. The planning and building of urban areas needs to take into account not only economic and aesthetic aspects but also the logistical problems associated with the concentration of working, living, shopping and recreational activities [1; 2]. Capacity requirement will need to be predicted and possible designs compared with regard to their efficiency.

One particular aspect of traffic has gained increased interest in recent years - pedestrian traffic. This is due to the fact that vehicular traffic has become overwhelming and cities are losing their attractiveness as places to live [3]. The increased pollution as the result of increased car traffic has a negative impact on peoples health as has been shown by research carried out as part of, for example, the DAPPLE project [4; 5]. Particularly in old towns with insufficient road capacity, the already narrow pavements are obstructed by parked cars, traffic signs, bus stops, barriers and other items of street furniture. Crossing the road can be a hazard at unregulated places, but even road crossings can have negative impact on pedestrian traffic if queues build up and impede pedestrian movement along the pavement. As a result people get discouraged from walking as a means of transport. Formerly local activities such as shopping and entertainment are performed at centres in remote locations or online over the internet. As a result of all these issues people tend to walk less and less. For example [6] shows “that the online grocery market is an expanding market segment” and records diminishing foot traffic as a consequence.

Several measures have been suggested to target this decline in walking activity, especially when designing new urban areas (e.g. [7]) but also in existing places (e.g. improving the interchange between pedestrian and other modes of transport to provide a ‘seamless journey’) and many governments have addressed this problem by creating walking strategies or similar guidelines to encourage walking. In the United Kingdom, the Department of the Environment,
Transport and the Regions (DETR) expresses concern that little is known about which measures are most effective in encouraging walking [8].

One way to achieve this is to provide attractive walking spaces that allow pedestrians to reach their destination efficiently and without interference. On the other hand the design should allow people to walk in their own way without hindering others. This includes walking together in groups, walking at individual speeds and even stopping to watch a street musician or shop display. Interfacing with other means of transport is another requirement, be it public transport like a bus stop or access to a car park.

Design tools (CAD, design guidelines, rule checking) need to be extended to include a utility to predict the resulting pedestrian traffic and verify that the suggested design can cope with demand before it is realised. The town planner, faced with such a task, needs a tool that will allow different designs to be compared in terms of their attractiveness as well as their effectiveness for the purpose. A shopping area has different influences on pedestrian movement from a public park and hence different layout requirements. Ideally a pedestrian model should allow describing the purpose and size of the place, the expected population of people and their purpose and destination to allow experiments for different layout designs to be evaluated on different criteria. If there was a way to expose modelled pedestrians to a choice of pre-defined virtual environments which reflect possible designs for an urban area and if it was possible to extract measurements from the model that can be used as an expression of ‘effectiveness’, then the option that provides the highest level of service could be selected and implemented.

Ideally such a tool would allow the representation of any possible place layout by either drawing it within the application or importing it from existing blueprints or CAD applications. Facilities to change the layout quickly by, for example, inserting additional pieces of street furniture or widening/narrowing the pavement would be beneficial. This virtual environment would then be populated with virtual pedestrians which would behave/move according to parameters that mimic behaviour typical for the currently modelled environment (e.g. it would not be helpful to model the behaviour of people on their way to work with behaviour patterns observed in a shopping centre). The urban
planner would be able to observe the overall flow and identify bottlenecks in the design as well as regions that are not walked on and other areas of interest. In addition to these qualitative observations, quantitative measures could be extracted such as density in certain areas, distribution of journey time etc.

For such a model to be useful it needs to be realistic enough (e.g. based on information from the real world or ‘calibrated’) as well as fast enough. While complex models often run in a server environment with large processing capacity (space and computational), the application area for a pedestrian model as described would be the typical office computer. And while PCs are getting more powerful every year, it is important to make sure that such a model can be used by the target audience without further investment in computer infrastructure. A related requirement is that the software must be compatible with typical desktop operating systems such as Microsoft Windows or Linux and not require specialist hardware.

When investigating if such a tool already existed at the beginning of the project (see also section 2.3) it became obvious that pedestrian modelling was still at an infant stage. Network or route-choice models [9] which have historically dominated modelling of vehicular traffic in transport research [10] have been adapted for use with pedestrians. However, network approaches are of limited value in modelling pedestrian traffic because they use a predefined number of routes (roads) from which the person/vehicle can choose. By giving the routes or parts of the routes an associated cost one can assess the flow of traffic. Interaction between vehicles (congestion) is captured in the cost factor and as the reasoning is on a statistical basis; nothing can be concluded about individual vehicles. Pedestrians in open, unobstructed spaces however are not restricted to specific routes. Their movement seems almost chaotic and can be determined by many different factors, including ability, journey purpose and the interactions with other individuals and features of the built environment.

New kinds of pedestrian models were proposed based on empirical studies which suggest that pedestrian movement can be correctly predicted if the number of environmental features and other relevant factors are kept small [11]. Models have been created based on those assumptions and work well within their constraints [12]. These models have however limited usefulness in
dynamic situations like pedestrianised shopping areas or pavements with bus stops, where factors like journey goal, obstructions and position of other pedestrians change suddenly. Models of pedestrians have also been developed which seek to identify pedestrians at a strategic level [13], but they only represent routes approximately.

Others represent pedestrians as a continuous flow [14] in order to estimate congestion in situations like the London Underground [15] or use a force model to simulate the attraction and repulsion between pedestrians [16]. But even these models are not suitable for dynamic situations where journey goals change, obstructions become visible and other pedestrians move around (such as in shopping areas or bus stops).

Most of these models have in common that they exist only as “proof of concept” implementations. They have gone through only a limited calibration process and use only a minimal set of information from the real world.

Having identified the need for a pedestrian model to aid urban planners, the following research objectives have been formulated:

- Review the existing literature (“state of the art”) to determine if such a model already exists and refine its requirements.
- Investigate which design features, implementation details or calibration mechanisms of existing models are suitable to fulfil the defined requirements.
- If necessary develop new designs, implementation details or calibration mechanisms.
- Validate the model with regards to its realism.
- Evaluate the computational performance and limits of the model.

This thesis presents the results of the research and is structured as follows: after the brief introduction into the subject and motivation for the project in the current chapter, the second chapter introduces modelling in general with specific emphasis on microscopic and agent-based models in the context of the modelling of pedestrian movements. Chapter 3 describes the design of the PEDFLOW model and justifies design decisions based on the experience from other models and original research. Chapter 4 shows how the model ideas are transformed into a prototype software application. Selected implementation
details are discussed and decisions justified. The chapter also describes and explains the visual appearance and usage aspects of the program. It explains the methodology by which parameters used in the PEDFLOW model are derived from observational research. The working model prototype is then evaluated in Chapter 5 by comparing its outputs against observations of the behaviour of real pedestrians. Also computational requirements of the model are explored and the scalability limits established. Chapter 6 summarises the conclusion from the research, states the contribution to knowledge and gives an outlook towards potential further work exploiting the findings and further refining the model.
2 Modelling pedestrian movement

This chapter reviews existing literature on modelling and evaluates existing mechanisms for modelling pedestrians and related entities. It identifies the need for a pedestrian model that allows an urban planner to evaluate different layouts with regards to their efficiency. Existing models are compared with regards to the used approaches and techniques and a summary of useful features and shortcomings is derived.

2.1 Computer models and computer simulations

The term model stands for “example”, “pattern” or “template” and is generally defined as a man-made representation of a part of the world or a system. A system is a part of reality with well-defined boundaries. Interaction with the outside happens over a limited number of channels. With the design of a model, an ideal or typical representation of reality is created in order to show only the specific features of reality (and/or their changes) which are of interest. Models are required because of the complexity of reality. They are aids for thinking and a source for hypothesis in research. Models do not show the whole truth, but a useful and understandable part of the truth. They are typically used to conduct experiments that can not be conducted in the original system, because that would be too expensive, too dangerous or simply because it is more convenient to do it in a laboratory. Other reasons include speed (a simulation can run faster or slower than the original system) and the possibility of extracting values not measurable in the real system.

Models can be concerned with the microscopic (relative to system size) building blocks of the system or with macroscopic properties and their distribution in the system. Examples of the former are models of particle movements in gases such as the “Particle-motion-resolved discrete model” described in [17] and for the latter stress models for shapes and materials used by designers like the “Damage model for hardening and softening material” described in [18].

Models are often (but not always) concerned with the change of properties over time, for example climate predictions. They are based on physical laws or empirical evidence. Models can sometimes be represented by a mathematical system of equations that in some form allows the calculation of the model output in relation to a number of input values. An example for the former would
be temperature with regards to certain times and locations in a meteorological climate model. Other models have to run through a number of incremental stages before an output can be extracted. An example would be the generations in a model of the growth of a colony of bacteria. Common for all models is that they need to be empirically validated (confirm the validity of results by comparing them to observations for a sufficient number of representative scenarios) before predictions based on the models can be transferred back into reality.

Although models exist in a variety of shapes and forms, in the last three decades computer models have established their place in engineering, medicine and many other fields of research [19]. Here the representation of the object or process modelled is virtual – no material exhibition exists. In that way it is very similar to the “mental” models on which people base their decisions in everyday life and which only exist as part of people’s consciousness. Mental models are based on people’s mental images of the world, of the relationships between their parts. By expressing these mental models as algorithms that can be performed by a computer their shortcomings, such as limited computational capacity or difficulty in communicating them to other people, can be reduced.

Advantages of computer models are speed, flexibility and the possibility of easily post-processing and analysing the data gained during the simulation. Well-designed model frameworks and toolkits make it possible for specialists with little computer knowledge to conduct experiments in their field of expertise and display the results in a way that is useful for their research. With the decreasing cost and increasing computational power of today’s computer systems, process modelling is penetrating more and more markets and helps to provide scientifically justified solutions where otherwise there would just be experience or guesswork. User-friendly modelling software can run on the office desktop computer of a designer and is no longer a tool for the use of only the highly qualified engineer.

There are many different and quite distinct types of models, however only a few are relevant to pedestrian modelling. The following attempts to place these types of modelling in the PEDFLOW context (i.e. the modelling of pedestrians to support the urban planning process) according to a number of criteria:
Static versus dynamic
A static model attempts to explain a particular phenomenon at a given point in time, whereas a dynamic model is concerned with the processes by which a particular phenomenon occurs. PEDFLOW is concerned with the modelling of the movement behaviour of pedestrians and as such requires a dynamic model.

Deterministic versus stochastic
Deterministic models are based on the doctrine that there is a reason for everything and as such are useful in situations where a limited number of well defined input variables lead to predictable outcome, whereas stochastic models use random variables (based on a distribution) for parameters that are uncertain. While pedestrian behaviour in the real world is (in theory) deterministic, the number and variety of inputs makes it impossible to model (or even understand) the process deterministically. Also, the purpose of a model in the context of the thesis is not to model individual people, but a representative group for a certain situation. This group will be represented by model parameters that are random, but vary according to statistical distribution – hence stochastic models are most appropriate.

Simulating versus optimising
For dynamic models the model process can either attempt to mirror the existing real world process or it can attempt to ‘improve’ it by dynamically varying input parameters in such a way that one or more output variables are maximised. For example, if we consider the (modelled) pedestrian’s parameters to be inputs to the model, one could envisage a program where these parameters (for example speed) could be optimised to achieve maximum throughput through a given walkway. However, these parameters would no longer reflect real pedestrians (such algorithms would still be useful in other application areas, e.g. games). PEDFLOW is purely a simulation model. The optimisation of the street layout is done interactively by the urban planner and there is no inherent support required in the model to help with it.

Macroscopic versus microscopic
The phenomenon of interest can be modelled as a single entity or as the combined results of contributing effects. In the context of pedestrian modelling, the macroscopic phenomenon could be the flow, with attributes like, for
example, density, and average speed. However, to model flow directly would require knowledge of its dependency on place layout and other external factors – the issue PEDFLOW wants in fact to help to investigate in the first place. Consequently PEDFLOW needs to be a microscopic model, where the modelling of the behaviour of individual pedestrians is used to create the macroscopic effect as emergent behaviour (i.e. it is not programmed into the algorithm). It needs to be noted that some recently developed models feature a hybrid of microscopic and macroscopic aspects [20] where strategic aspects (such as route planning) are modelled macroscopically and the actual movement is microscopic.

**Time advanced versus event driven**

This category describes the circumstances under which a change in the state of the model is possible. In time advanced models the model’s state is updated at regular time intervals, whereas in event driven dynamic models model state changes are triggered by condition changes (events) at irregular intervals. Both methods can be found in implementations of pedestrian models. A discussion how it is implemented can be found in section 0.

A pedestrian model to serve the purpose outlined in Chapter 1 would therefore be a dynamic, stochastic, microscopic simulation that it is either time advanced or event driven. What is not yet clear, is how the actions of the individual microscopic components (modelled pedestrians) are determined. Considering that pedestrians in the real world are individuals that are not controlled by an external influence, the logical consequence is to represent them as individuals in the model as well. An established computing paradigm for such individuals is an autonomous agent [21; 22].

### 2.2 Autonomous agents in modelling

The term ‘agent’ is an overloaded term that is used in a wide variety of contexts. There is no generally acknowledged definition but instead features can be named that are common in varying extents to all agent implementations. In general it can be said that agents represent ‘living behaviour’. They are objects capable of independent actions within a system. This system, including the agents who ‘live’ in it, can be considered as the agents’ environment or their ‘world’. The latter is used in order to emphasise the similarity of the concepts.
This world can be ‘observed’ by the agents via sensors, which, just like sensory organs on living beings, collect information about the world either passively or actively. In the context of this ‘world view’, agents have goals they are trying to achieve by modifying their world with the help of effectors. Consequently this requires that agents are adapted to a specific type of world with regards to the type of ‘organs’. Agents can communicate with other agents in the shared environment by use of additional ‘organs’ to exchange information. Combined with the aforementioned re-activity, group behaviour can evolve. This group behaviour can be so complex that it becomes unpredictable from the observation of a single agent.

A comprehensive attempt at defining agents can be found in [23]. Franklin et al. analyse agent definitions from a number of sources and formulate the essence as: “An autonomous agent is a system situated within and part of an environment that senses that environment and acts on it, over time, in pursuit of its own agenda and so as to effect what it senses in the future.”

They then name a number of typical agent properties as shown in Table 2.1.

<table>
<thead>
<tr>
<th>Property</th>
<th>Other Names</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>reactive</td>
<td>sensing and acting</td>
<td>responds in a timely fashion to changes in the environment</td>
</tr>
<tr>
<td>autonomous</td>
<td></td>
<td>exercises control over its own actions</td>
</tr>
<tr>
<td>Goal-oriented</td>
<td>pro-active purposeful</td>
<td>does not simply act in response to the environment</td>
</tr>
<tr>
<td>temporally</td>
<td>continuous</td>
<td>is a continuously running process</td>
</tr>
<tr>
<td>communicative</td>
<td>socially able</td>
<td>communicates with other agents, perhaps including people</td>
</tr>
<tr>
<td>learning</td>
<td>adaptive</td>
<td>changes its behaviour based on its previous experience</td>
</tr>
<tr>
<td>mobile</td>
<td></td>
<td>able to transport itself from one machine to another</td>
</tr>
<tr>
<td>flexible</td>
<td></td>
<td>actions are not scripted</td>
</tr>
<tr>
<td>character</td>
<td></td>
<td>believable &quot;personality&quot; and emotional state</td>
</tr>
</tbody>
</table>

Table 2.1 Typical agent properties from [23].

In many agent implementations, agent perception and behaviour mechanisms are described using standard programming languages. This has the advantage of being very flexible, but the disadvantage that the perception, behaviour and interaction aspects are difficult to separate, even within an agent's description, and are therefore confusing. In a more formal approach, the functionality is completely separated and encapsulated in separate ‘functions’ or other appropriate concept of the programming language.

Autonomous agents in computing have a number of features that make them well suited for modelling people. With a term as open to interpretation as the
term agent, it is difficult to find a structure in which to organise all agent types. At the conference Agents ’97 [24], a linear classification of agent issues was suggested as follows:

- synthetic agents
- knowledge acquisition and accumulation
- action selection and planning
- models of personality
- sensing and perception
- modelling the environment
- learning
- expert assistant architecture
- user modelling
- multi-agent interaction
- world wide web agents
- autonomous robots
- people-agent interaction

This scheme, although a useful grouping for organisational purposes, is merely a random selection of agent-related issues. It is insufficient for the classification of agents because of the overlap and the lack of structure. In order to help with the evaluation of existing pedestrian models, an alternative classification has been developed as part of this dissertation that distinguishes the variety of agents in existence with regards to a number of criteria that are particularly relevant in the context of agent-based modelling:

**The environment they live in**

Certain kinds of agents are created to exist within ubiquitous computer systems and their inputs are the real live values of that system. It could be the personal computer on someone’s desk, the control unit of a numerically controlled industrial robot or even the nodes of a computer network within which the agent is able to travel. Examples are indirect management agents or fault detection agents [25]. Another group of agents lives in a purpose-built, virtual environment, usually an abstract model of a part of the reality or an artificial world. Agents in computer models are examples of this type.
Their primary task

The majority of agents are generally concerned with information gathering. They occupy a pre-existing environment and unobtrusively either collect information about the environment itself, extract information stored in the environment or analyse the interaction of external entities (e.g. humans) with the environment. The accumulated information is either periodical or on-demand output for use outside the environment. Examples are www agents for data collection, e.g. ARA [26] or agents to gather customer data for targeted advertising [27]. A growing application for agents is user assistance. With the increasing influx of computers in everyday activities, the untrained user is often overwhelmed by the great number of options presented by unfamiliar and non-intuitive user-interfaces. Here agents work as personal assistants and guide the user in unfamiliar terrain but otherwise behave unobtrusively. A well-known example is the ‘paper clip’ agent in Microsoft applications. Agents are also used in entertainment software, specifically games. Here the real-time strategy (RTS) games [28] are the best example for use of agent technology, where a number of independently acting opponents, controlled by an artificial intelligence, move in a virtual world and react to the actions of the player. A scientific application area is simulation and modelling, where agents are used to model real-life objects in an attempt to predict outcomes of environmental changes in ‘what-if’ scenarios. Pedestrian models fall into that category.

Source of knowledge used in the decision-making process

In order to react to the environment, algorithms are required that control the behaviour of the agent. These algorithms can be static and built into the agent or they can dynamically change during the lifetime of the agent. In the case of changing knowledge, further distinction is necessary in pre-programmed changes, self-improvement/adaptation or user-induced modification. Typically agents will improve their interaction over time as a function of the knowledge they collect. This usually happens under the control of the user in order to prevent unexpected side effects. But there are also application areas for static knowledge or pre-programmed changes.
Extent of spatial awareness

For many purposes agents do not require a sense of location. An example is the ‘personal assistant’ type, described by P. Maes in [29]. Some agents will move around in their environment, e.g. a computer network like the one described by B. Aoun in [27], but the physical location is inconsequential for their task. For another kind, spatial awareness is the main requirement as their task is concerned with moving between locations. All games and many models fall into this category.

Inter-agent communication

In addition to being passively perceived by other agents, some agent implementations have the ability to actively communicate as a means to changing other agent’s internal state. Communication can happen on several levels - unreliable, reliable randomly, reliable periodically.

Creation, termination

Typically agents go through phases in what could be considered as their ‘life’. They are created, have parameters assigned, are activated, and eventually die. The activation phase might be further structured to contain periods of inactivity and/or objective changes. Optionally a post-mortem phase may exist, where information is extracted. Phase changes such as creation or death can be user-initiated or event-initiated. In the latter case the events can be environment changes or agent-caused. The latter is of great interest in some application areas (i.e. biological research), as it suggests the possibility of ‘procreation’, with the implication of concepts like ‘generations’ and ‘inheritance’.

Implementation/programming language

Agents could in theory be implemented in every imaginable programming language. In practice, however, the choice of implementation is usually influenced by a number of factors like performance, ease of implementation and interfacing. In particular, agents designed to ‘live’ in a pre-existing environment require careful consideration with regard to their implementation in order to be unobtrusively integrated. If the environment is built alongside the agents, the choice is less critical.
Agent representation/visualisation

The majority of agents provide some form of user interface to allow for the setting up of specific parameters or the presentation of results. Depending on the kind of agent, this can be implemented offline, online or a combination of both. The online version is used if it is required to observe the change of the internal state over time. Usually, colour, shape and/or location are used to indicate that a value is within a certain range, where the exact values can be extracted on demand.

With this proposed scheme it is possible to categorise any example of agent implementation, and in particular agent-based pedestrian models, with regards to its criteria. While these pedestrian models share a number of common characteristics which sets them apart from other agent based models, they differ in details. The list summarises the common characteristics and points out the potential differences by which existing models are compared and evaluated in section 2.3.

- The environment agents live in is a purpose-built environment. Pedestrian models differ in their representation of space.
- Their general, primary task is for simulation and modelling. Pedestrian models differ in how agents determine their goals.
- The source of knowledge used in the decision-making process is static and pre-determined. Pedestrian models don’t differ from each other in that aspect.
- The extent of spatial awareness required. Pedestrian models differ in their ability to react to multiple different concurrent influences in different ways.
- Inter-agent communication. Pedestrian models differ in the extend to which they model pedestrian communication.
- Agent representation/visualisation. Pedestrian models differ in whether the visualisation is online (during the run or the experiment) or offline (via post-processing). The level of visualisation also differs.
- Creation and termination are usually user-initiated as part of well-defined experiments. Pedestrian models don’t differ from each other in that aspect.
- Implementation. Pedestrian models are implemented on different hard and software platforms.
Differences are usually caused by the fact that the specialised application areas have different requirements, which lead to different approaches in the model creation. They are discussed in the section below, which reviews the cross section of pedestrian modelling tools available at the time of writing.

2.3 Evaluation of existing pedestrian models

2.3.1 Cellular Automata model (Blue and Adler)

Blue and Adler [30; 31] use a basic cellular automata (CA) approach to model pedestrian movement. A cellular automaton consists of a regular two-dimensional lattice, where each cell of this lattice has a discrete state. It can be empty or occupied. The dynamic behaviour of the CA is described by a formula describing the state of a cell for the next time step depending on the states of its neighbouring cells. Movement is modelled as the vacation of one cell and the occupation of a new cell. The model is implemented in Java and can run in real time with online visualisation as an applet.

![Figure 2.1 Bi-directional pedestrian flows visualised in a Java applet at [32].](image)

The basic CA concept was expanded by Blue and Adler to include a directional component [33]. If the destination cell is already occupied, a random neighbouring cell is chosen. With the intended aim to create a model of minimal complexity, they have achieved some interesting results in creating emergent, collective behaviour of modelled pedestrians. The modelled environment however is limited to moving pedestrians only; no interactions with other objects exist, nor can they be easily added. The authors state in [34], page 1: “By ‘designing’ the CA-based pedestrian from the bottom-up at the interface with one another, higher-level functions, like route selection and trip behaviour, can be added later without fundamentally changing the inter-pedestrian dynamics.”. This seems to imply that extension to include more elaborate rules is possible. However, there do not seem to be any specific facilities to achieve that. In later versions they extended the model to include four-directional movement [35]. An additional problem with the model is the fact that parameters like walking speed...
are assigned randomly and do not relate to measurable attributes of the modelled environment. These disadvantages make it unsuitable for

With respect to the categories from section 2.2, the Cellular Automata Model by Blue and Adler, although microscopic, is not an agent-based model. The purpose-built environment the modelled pedestrians live in is represented by a lattice. The modelled pedestrians work towards a pre-defined goal. The movement is determined by a global, static rule-set. There is no ability to react to multiple concurrent influences. Modelled pedestrians do not communicate with each other. The location and movements of modelled pedestrians is visualised online. The demo software is implemented in Java and runs as an applet in Java enabled browsers.

2.3.2 Agent based CA model (Dijkstra)

The Dijkstra model [36] has extended the CA model further and has combined it with a network approach. The aim is to create a three-dimensional visualisation of simulated pedestrian activity in the retail environment.

Figure 2.2 Cellular Automata grid and visualised pedestrian movement in [36].

They use agent technology to better simulate autonomous individuals and the interactions between them. Agents have an “activity agenda”, which is updated according to rescheduling of activity decisions, perceptions of the environment and adaptation of time-budget. The update however can only occur at network nodes or on completion of an action. Between nodes the activity is restricted to obstruction avoidance. While the software is fit for the purpose of evaluating retail layouts, the design makes it hard to include more rapid responses to other influences in the environment or even to deal with multiple influences concurrently.
With respect to the categories from section 2.2, Dijkstra’s model can be described as an agent-based model. The purpose-built environment they live in is represented by a lattice. The agents work towards a pre-defined goal. The source of knowledge used in the decision-making process is a static rule-set. There is no ability to react to multiple concurrent influences. Agents do not communicate with each other. The location and movements of agents is visualised online. The extended CA model is implemented in an unspecified modelling framework and runs on a desktop PC.

2.3.3 STREETS (Schelhorn, O'Sullivan)

The STREETS model [37] is an agent-based model of large urban areas. Features of the environment (e.g. buildings or pavements) and pedestrians are modelled as agents and their attributes are automatically derived from several Geographical Information System (GIS) data sets (e.g. socio-economic data, street networks). Pedestrian agents emanate from gateways and move between pre-assigned way-points. On their way they are ‘distracted’ by other agents and their route is modified by attributes such as ‘walkability’ or ‘fixation’.

The STREETS model consists of a modelling stage, which populates the agent-based model and the simulation stage where experiments are run to predict the pedestrian movements.

Agents in streets are described by their:

- intended routes
- behavioural parameters (e.g. maximum walking speed, visual acuity)
- preferences (currently expressed as a meaningless number)
The environment is characterised by:

- a vector representation of the buildings (buildings have attributes representing functions - retail, commercial, residential, various categories of each of these - which describe their attractiveness for ‘matching’ types of agents)
- a grid (approx. 1m) representing the ‘walkability’ of different areas
- a network of ‘way-points’ areas describing the route of the agents

During the simulation phase agents move from way-point to way-point using a mechanism referred to as ‘high-level navigation’ while choosing their route according to the ‘walkability’ of the grid elements - the existence of another agent in the grid-square makes it less ‘walkable’. They might be attracted by buildings of a matching type and this attraction causes a delay - agents enter the building - which is compensated by an increased ‘fixation’.

The level of detail suggests that STREETS is essentially a mesoscopic [38] model. It is not aimed at modelling the pedestrian at a step-by-step level, but is concerned with a more global view. As such it is not suited to the evaluation of the quality of service provided by urban areas - in fact one could argue that the level of service is an input to the model and agents chose their route according to the ‘quality’ of the environment. Its usefulness lies in a different domain. If the methodology that extracts the model parameter from GIS data can be proven valid, it would provide a way to reason about potential bottlenecks in the street layout, rather than the design of streets or places.

With respect to the categories from section 2.2, the STREETS model can be described as an agent-based model. The purpose-built environment the agents live in is represented by a lattice. The agents work towards a sequence of multiple, pre-defined goals. The source of knowledge used in the decision-making process is a static rule-set. The spatial awareness is limited but there is ability to react to multiple concurrent influences. Agents do not communicate with each other. The location and movements of agents is visualised online. The STREETS model is implemented in C using an Agent model framework called ‘SWARM’ [39] and runs on multiple desktop PCs.
2.3.4 ROBO SOCCER (Stone)

Strictly speaking not a pedestrian model, this project was included in the review as it shows some aspects of an autonomous agent based simulation that contrast with agents as used in a pedestrian flow model while still maintaining a large number of similarities.

A project at Carnegie Mellon University [40], it explores the operation of a society of robots that operate co-operatively to achieve a common goal, in this case a game of ‘robot soccer’. The algorithms of the physical robots are based on an agent-based simulation model.

![Figure 2.4 RoboCup Simulator in [40].](image)

In the simulation model, agents are capable of perception (observation), cognition (decision), and action. They create an internal model of the environment’s current state from their observation of (part of) that world. Then, based on a set of behaviours, they choose an action appropriate for the current world state. The simulator operates in fixed cycles of length 0.1s. During that cycle, the agent will store all information that results from a sensation (see, hear, or sense/body). On a cycle change it uses all of the information available (temporary information from sensations and predicted effects of past actions) to update the internal world model to match the server's world state (the “real world state”). Based on that state the agent chooses and sends an action to the server. Although agents can theoretically perform a number of actions per cycle, only one is accepted by the simulator which updates the world state at the end of the cycle with the combined results from all agents. If no action command is sent, an action opportunity has been lost.
One of the biggest challenges in the model is to keep the internal model up to date with the limited input data. The agents use sophisticated algorithms to predict the position of the ball and other players from previous position, velocity and levels of uncertainty to represent the results. This is a feature that is not commonly used in agent models, where the action is typically only based on the observation.

Another important issue is co-operation. In earlier versions robots from the same team would work more or less independently by dynamically taking on a ‘role’. Examples are: handle ball (used when the ball is kickable), active offence (go to the ball as quickly as possible; used when no team-mate could get there more quickly), auxiliary offence (get open for a pass; used when a nearby team-mate has the ball) etc. Later implementations have improved on this concept by providing co-operation by means of ‘Locker-Room Agreements’ and ‘Communication’. Locker-room agreements are based on the assumption that agents can periodically meet in safe, full-communication environments and specify how they should act when in low-communication, time-critical, adversarial environments. In the latter communication is limited to short messages with no guarantee that the messages will reach their destination.

The main difference between soccer players and pedestrians, as far as their modelling is concerned, is the fact that pedestrians typically don’t work co-operatively in changing roles towards a common goal. It could also be argued that this is more an optimising model than a simulating one, and that even though certain behaviours of real players are simulated, the goal is to find the best strategy to win within the limits of the hardware.

With respect to the categories from section 2.2, the ROBO SOCCER model can be described as an agent-based model. The purpose-built environment they live in is a continuous space. The agents work towards dynamically changing goals. The source of knowledge used in the decision-making is a dynamic knowledge base with self-learning features. The spatial awareness is limited but there is ability to react to multiple concurrent influences. Agents do communicate with each other. The location and movements of agents is visualised online. The ROBO SOCCER model is implemented in client-server implementation using UDP/IP for communication on multiple desktop PCs.
2.3.5 Discrete Force Model for pedestrian motion (Helbing)

Although not an agent model, the Discrete Force Model (DFM) for pedestrian motion [16] is mentioned here because it features concepts also found in agent models of pedestrian flow. Pedestrians are modelled as individual entities, but their behaviour is not controlled internally but by external ‘forces’. Pedestrians behave as if they are subjected to an acceleration force towards their goal and to repulsive forces describing the reaction to borders and other pedestrians. As a result self-organisation and collective behavioural patterns emerge.

![Figure 2.5 Modelling pedestrian interaction at a crossing with the DFM.](image)

Unlike the STREETS model, DFM is concerned with microscopic behaviour on a street level. The environment is geometrically simple and is represented as a path, a junction or a square. Entities experience an accelerating force from a destination point that determines their primary direction and speed and corresponds to their desire/motivation to get to the destination. There are no intermediate goals, although the model could be extended to include a mechanism to insert a new goal once the first destination has been reached. In addition to the primary force other pedestrians assert a repulsive force that corresponds to their territorial requirements. Similarly walls or items of street furniture have a repulsive or attractive force attached. The sum of all directed forces dictates the final movement.

The Discrete Force Model has been extended to include the localised effects of pedestrians on their environment, specifically the development of trail systems on deformable ground (‘Active Walker’ [41]). Here changes of features of the walkable ground produced by one pedestrian will attract other pedestrians until
the trail becomes less walkable and pedestrians will walk next to the original trail, effectively moving the trail. The direction of the movement is determined by other factors (e.g. goal location) and leads eventually to commonly found trail layouts, like y-shapes. The DFM is able to replicate these shapes with high accuracy.

With respect to the categories from section 2.2, the DFM model is not an agent-based model. The purpose-built environment is continuous. The modelled pedestrians move towards a pre-defined goal. The location and movements of the modelled pedestrians can be visualised online. DFM is implemented in Java and runs on a desktop PC.

2.3.6 SimPed (Daamen)

SimPed [42] is a large scale (at least 100,000 m² area, about 100,000 persons at any time) pedestrian simulation tool especially tailored for the modelling of transport interchanges such as train stations. The model layout is drawn in a SimInput module which behaves like a drawing tool. In addition to the infrastructure, the origins and destinations of the pedestrians can be specified, as well as temporal aspects of the model such as timetables. Additional parameters can be defined in the SimControl part of the application. The Simulation is run as several applications (SimPed, SimAnimation, and SimArchive). As the name suggests, SimPed handles the actual simulation, while SimAnimation will display an online visualisation and SimArchive will collect and store the model output.

While the SimPed model [43] is microscopic in the way in which it presents the pedestrian flow as individual pedestrians, the behavioural rules that describe the flow are macroscopic [44]. This has benefits such as low computation time and reduced calibration and validation effort, but the applicability of the model is restricted to situations that have been investigated already and for which the macroscopic measures such as speed-density curves or capacities are available. For example, a bottleneck in the design can only be modelled correctly if its capacity is known, since the capacity will depend on both the geometric shape of the bottleneck and the individual behaviour of the pedestrians. Thus it is unsuitable for the intended purpose of trying and evaluating different designs.
With respect to the categories from section 2.2, the SimPed model is not an agent-based model, although pedestrians are represented microscopically. The purpose-built environment they live in is represented by a continuous space. The modelled pedestrians work towards visiting multiple pre-defined goals. The behavioural rules that describe the flow are macroscopic. Modelled pedestrians can react to multiple concurrent influences. They do not communicate with each other. The location and movements of agents is visualised offline. SimPed is implemented in MODSIM III [45] and VC++ (supporting tools) as a multi-computer network application.

2.3.7 Nomad (Hoogendoorn)

Nomad [44; 46] is a microscopic pedestrian flow simulation tool intended for the modelling of public pedestrian facilities such as airports, public transit transfer stations and shopping malls. It consists of two mutually dependent models. The tactical model predicts pedestrian activity scheduling, activity area choice and route choice in public spaces. Rather than routes just being sets of links, they can be continuous curves in space and time. The operational (and in the context of this review more interesting) part of the model concerns the walking behaviour of the pedestrians. It is based on optimal predictive feedback controllers with limited prediction horizon, so it is by no means an autonomous agents model. Given an ‘optimal velocity’ (speed and direction) from the route-choice part of the model, the movement is determined as the solution of a differential equation that takes into account physical interactions with other pedestrians, obstacles but also special infrastructure such as grade of walking surface or stairs. Multiple activities can be chained together, for example as a sequence of multiple destinations.

The prototype implementation accepts all input parameters in a text file:

- Run-time parameters (e.g. length of simulation period)
- Network topology (e.g. map of the walking area)
- Parameters describing walking behaviour (e.g. free speed, size of pedestrians)
- Activity scheduling and route choice parameters (e.g. origin areas)
- Location of virtual detector loops.
The program will generate a number of outputs such as:

- Animations of the pedestrian simulation
- Snapshots of the situation at predefined time instants
- Density and speed contour plots
- Output of the virtual detector
- Trajectories of each pedestrian
- Individual pedestrian characteristics

The strength of the model is the combined approach of a tactical (route-choice) model with an operational, microscopic model. As the microscopic model is not agent based, it doesn’t benefit from the flexibility this provides (e.g. use real world parameters directly).

With respect to the categories from section 2.2, the NOMAD model is not an agent-based model, although pedestrians are represented microscopically. The purpose-built environment they live in is represented by a continuous space. The modelled pedestrians work towards visiting multiple pre-defined goals which are determined dynamically by a route-choice model. The behavioural
rules that describe the flow are macroscopic. Modelled pedestrians can react to multiple concurrent influences. They do not communicate with each other. The location and movements of agents is visualised online and/or offline. The implementation of NOMAD could not be determined.

2.3.8 PEDFLOW1 (Kerridge)

PEDFLOW1 [48] is the project (with prototype implementation) that prompted the development of the subject of this thesis, the PEDFLOW model. In PEDFLOW1, pedestrians are represented by agents (i.e. software objects) which can move in a virtual world representing an urban environment. Their behaviour is controlled by a set of rules applied to a snapshot of the environment. The rule-set can be considered as a function where input variables (observation of the virtual environment) combined with parameters specific to a ‘type’ of person in a ‘type’ of situation that are transformed into output variables (a change of location in a specific time). This cycle is repeated continuously. The sum of the individual decisions and their results make up what can be observed as movement. Multiple agents’ movements form a flow that is also observable and to which measurements can be applied. Although it can be interesting to observe individual agents, the primary output is the resulting flow. The properties of the flow are emergent by nature [49] because the microscopic interactions between agents only use local knowledge. No agent has access to all of the globally available data.

Agents (and static entities) are mapped to a grid with a grid size that corresponds to the agent size, which in turn is a rough representation of the size of the ‘average’ person. In PEDFLOW1 it was set to 0.75m. Agents move on the grid in the way the king moves in the game of chess, but their moves are dictated by a set of rules that transforms their perception of the grid-occupation in front of the agent into a change of position over a certain time. More details about the rule-evaluation and other implementation details can be found in [50].

A prototype of the model was first implemented in the parallel programming language OCCAM. Making use of the parallel processing design features, the agents are modelled as processes and the shared data structure is used to represent the environment. The data structure holds limited information about the existence of an agent in a grid (type, direction and speed) but not the agent
itself. This has advantages and disadvantages. The grid needs to be updated and be kept synchronised with part of the internal state of the agent but the observation part is simplified as the features to be observed are stored in the grid (no interactions between agents are required).

Concurrently active agents see the same environment; they make their decision based on what they see but the update of their position occurs in sequence. The environment change might require a roll-back of the action (similar to a database) in certain cases. Alternatively agents could lock all potential destination fields in a deadlock-safe way. This would imply sequential processing for neighbouring agents only at the cost of the additional locking-overhead. The computational approach does not necessarily conform to pedestrian behaviour.

Simulations are run as batch processes and the resulting files are then analysed or visualised. This means that users can only see the result after a complete run which deprives them of the possibility to abort an experiment that obviously goes wrong in the beginning. It might also be desirable to allow the operator to store an intermediate scenario which can be used without re-running the whole experiment, an action which is not possible with this implementation.

The program is executed on a dedicated DEC Alpha processor on a PC add-on board. The requirement for specific hardware makes it unsuitable for commercial use. The need for portability also dictates the move away from the proprietary OCCAM. Although emulators exist to run OCCAM programmes on a variety of platforms, for example [51], the program would need to be recompiled and also experience a performance drop with respect to running it on the native hardware.

Provisions for a graphical user interface are extremely minimal in OCCAM, making representation and interaction difficult to implement. To work around that shortfall, an offline visualiser would read the output of the OCCAM simulator and display the movement of agents. However, the problem with this solution is that there is no mechanism to influence the model at simulation time. All experiments have to run to completion, even if there is a problem that could have been discovered at an early stage with an online visualiser.
With respect to the categories from section 2.2, the PEDFLOW1 model can be described as an agent-based model. The purpose-built environment they live in is represented by a lattice. The agents work towards a pre-defined goal. The source of knowledge used in the decision-making process is a static rule-set. There is no ability to react to multiple concurrent influences. Agents do not communicate with each other. The location and movement of agents is not immediately displayed, but can be visualised offline. PEDFLOW1 is implemented in OCCAM and runs on a dedicated DEC alpha board.

2.3.9 Legion (Kagarlis)

The Legion model [52], a commercial software product, is described in USA patent 10237178. It is a multi-agent pedestrian simulation, where every pedestrian is modelled as a two-dimensional entity that can move in multiple 2D virtual environments that are connected via circulation elements. Circulation elements represent real-life objects such as stairs, lifts, ramps that allow the movement between different levels. The learning-adaptive agents use cost-minimisation function, where cost is calculated from the following factors [53]:

- Inconvenience: Work to move, in excess of the amount which is necessary to reach one’s destination
- Frustration: Energetic cost equivalent of violating the speed preference time expenditure,
- (Spatial) Discomfort: Energetic cost equivalent of violating the preferred clearance from neighbours and obstacles
The weights in the cost functions are derived from information gained in empirical studies [53] and are adapted based on recent experience (learning). It is questionable how the machine learning can represent the learning process of real pedestrians (see also section 2.1, simulation versus optimising), unless it is explicitly modelled using empirical data from real pedestrian observations for which no evidence was found. The agents distinguish other agents moving in the same direction as well as cross- and counter flow, but their perception is imperfect and they often have to predict the future position.

All agents move every 0.6s, regardless of speed or other factors. Since the time period is constant, speed is determined by the movement distance. It also allows to model individual sizes for pedestrians as well as personal space. However, it also implies the requirement for a high spatial resolution (i.e. grid lattice is not sufficient) and a performance overhead when calculating distances to objects.

What makes Legion very attractive is the availability of a large number of analyzes and visualisation tools. As a drawback, the resulting complexity makes it difficult for untrained personnel to use the product which suggests the use of consultancy services [54].

Figure 2.8 Example of Legion output for average delay (London Docklands Light Rail) [54]

With respect to the categories from section 2.2, the Legion model can be described as an agent-based model. The purpose-built environment they live in consists of multiple 2-dimensional planes connected by special circulation objects (representing stairs, lifts etc.). The agents work towards a pre-defined target. The source of knowledge used in the decision-making process is a pre-determined cost-function, but agents also learn from recent experience. Multiple concurrent influences can be incorporated into the cost function. Agents do communicate with each other to resolve local blockages. There are extensive
facilities to visualise the location and movement of agents on the map as well as through 3D animations (Legion 3D companion product). Legion is implemented in C and runs on powerful desktop computers.

2.3.10 PedGo (Kuepfel)

PedGo [55] is a CA model with a lattice size of 40cm$^2$. Elements can be either empty or occupied by a person or object (wall, furniture). Time is measured in discrete steps which correspond to 1s real time. The relatively coarse resolution in time and space determines the speed resolution. In [56], PedGo is claimed to be a Multi-Agent model, however the description does not correspond with the definition of autonomous agents as presented in section 2.2. PedGo is a stochastic model, meaning that the behaviour of the modelled people is determined by a probability function. Each cell has a ‘potential’ and transition occurs if the potential difference reaches a certain level. The model's parameter are based on the “effective (phenomenological) behaviour of and interaction between individuals.” [56]. There are six parameters: free walking speed ($v_{max}$), time after which a strategy change occurs ($t_{patience}$), reaction time ($t_{react}$), ($S$), orientation frequency ($p_{orient}$) and inertia ($\Theta$).

Figure 2.9 Screenshot of PedGo, simulating the evacuation on a cruise ship [57].
Given the limits above, the model is unsuitable for the requirements of the PEDFLOW model, but it is still very useful for the purpose for which it was developed: evacuation simulation. Due to its relative simplicity, the computational requirements are low: 90 MHz CPU, 128MB RAM, XGA graphics. Not only does that mean it will run on dated hardware, but it will scale effectively with numbers of people, so large simulations are possible. The User interface allows the interactive setting of parameters, gives control over the running of the experiments and supports the analysis of the results by providing a number of standardised diagrams, e.g. density plot and wait plot.

With respect to the categories from section 2.2, the PedGo model, although microscopic, is not an agent-based model. It is a stochastic CA model. The purpose-built environment the modelled pedestrians live in, is represented by a lattice. There are multiple exit routes, which are not pre-selected for a specific agent. The movement is determined by a parameterised probability function. There is no ability to react to multiple concurrent influences. Modelled pedestrians do not communicate with each other. The location and movements of modelled pedestrians is visualised online.

2.3.11 EXODUS (Galea)

The EXODUS [58] [59] is a sophisticated model that was developed to simulate the evacuation of large numbers of people from a variety of enclosure types such as planes, ships and trains. These environments are modelled as a two-dimensional grid. It maps out the geometry of the structure, the location of the exits, internal compartments, obstructions, etc. Multiple floors can be represented as multiple grids connected by staircases. The grid consists of nodes and arcs where nodes represent a small region of space and arcs represent the distance between the nodes. Modelled people travel from node to node along the arcs. The escape route chosen by each individual is the result of their interactions with the enclosure, other occupants and any fire hazards present.

The attributes of the modelled person are: gender, age, maximum running speed, maximum walking speed, response time, agility, etc. While some of the attributes are constant, others are changing as a result of inputs from other sub-models (e.g. toxicity). The behaviour sub-model has two aspects: global and
local. The local behaviour determines an individual's (microscopic) response to their local situation, while the global behaviour represents the strategic exit plan (the goal). Depending on the information from the behavioural sub-model, the movement module will change the person’s position, a manoeuvre that may involve overtaking, side stepping or other evasive actions.

The model consists of five sub-models [59]:

- Hazard (controls the atmospheric and physical environment.)
- Toxicity (determines the effects on an individual exposed to toxic products distributed by the hazard sub-model.)
- Occupant (describes an individual as a collection of defining attributes)
- Behaviour (determines an individual's response to the current situation on the basis of their personal attributes.)
- Movement (controls the physical movement of individual occupants )

Figure 2.10 buildingEXODUS: density display during the early stages of evacuation. [59]
The EXODUS application suite is implemented in C++ using oo programming techniques and consists of 5 specialised packages:

- buildingEXODUS for applications in the built environment
- airEXODUS for aviation applications
- maritimeEXODUS for marine applications
- railEXODUS for train applications
- vrEXODUS to view EXODUS generated evacuations in 3D

With respect to the categories from section 2.2, the EXODUS model shows aspects of an agent-based model. The purpose-built environment in which they live consists of multiple 2-dimensional planes connected by objects representing staircases. The agents work according to a global exit strategy. The source of knowledge used in the decision-making process is pre-determined. Multiple concurrent influences can be dealt with, but are limited to danger (hazard, toxicity) in accordance with the purpose of the product. Agents do not communicate with each other. There are extensive facilities to visualise the location and movement of agents on the map as well as through 3D animations (vrExodus). EXODUS is implemented in C++ and runs on desktop computers.

### 2.3.12 Model feature discussion

Table 2.2 summarises and compares selected features of the models discussed above to show the diversity of the existing implementations.

The table shows how different approaches have been used depending on the purpose for which the model has been developed. While the simple CA model is sufficient to generate speed/density curves for high-density “pedestrian flows that compare favourably with established curves as described in the Highway Capacity Manual” [32; 60], it would be unsuitable for simulating football/soccer games. Similarly the hardware requirements for STREETS would rule out use in a single-PC office environment. Models for an environment were a single factor is predominant, such as flight from a hazard (Pedgo), are not sophisticated enough to allow modelling of situations with multiple influences. Long turn-around times as required by models with off-line visualisation (PEDFLOW1) are not acceptable for an effective use.
Despite their disadvantages, all of the models contain aspects worth preserving/re-using in the Pedflow model. The spatial resolution by the lattice used as the agent environment by the CA models (Blue, Dijkstra) is sufficient and computationally effective. The use of agent-technology, as pioneered by STREETS has become the predominant technology for the microscopic modelling of pedestrians due to its intuitivity and flexibility. Lessons can also be learned from unlikely candidates like RoboSoccer when it comes to inter-agent communication. While early models often concentrated on modelling one single aspect (DFM), newer models employ a 2-level strategy, distinguishing between route finding and micro-navigation (SimPed). Online visualisation of the modelling process has become a standard, although off-line visualisation (EXODUS) or the use of additional hardware (SimPed) is still required for more sophisticated visualisation techniques as 3D presentations. The use of a universal modelling framework is rare (SimPed), most applications are programmed in a programming language such as C (STREETS), often object oriented (EXODUS), because of the flexibility offered.

Models that have been calibrated with data from empirical research (Nomad, Legion) show the highest level of realism. In fact, more recent models such as Nomad and Legion fulfil most of the requirements of PEDFLOW, however they didn’t exist during the development phase of the project. They were designed and calibrated with their specialised area (pedestrian behaviour at traffic interchanges, evacuation of objects) in mind, so there is still justification for a pedestrian model targeting the town planner as a user.

The next two chapters will show how the PEDFLOW has been created re-using and adapting features from existing models, as well as integrating novel solutions in order to create a model suitable for the specific purpose of aiding the urban design process.
<table>
<thead>
<tr>
<th>Model</th>
<th>Purpose</th>
<th>Realism/accuracy</th>
<th>Agent-based</th>
<th>Space representation</th>
<th>Route</th>
<th>Multi-influence</th>
<th>Communication</th>
<th>Visualisation</th>
<th>Implementation</th>
<th>Hardware</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cellular Automata Model (Blue, Adler)</td>
<td>study of speed/density dependency in cross and counter flow, no practical application</td>
<td>low, limited to specific scenarios</td>
<td>no</td>
<td>lattice</td>
<td>start-goal predefined</td>
<td>no</td>
<td>no</td>
<td>online</td>
<td>Java applet</td>
<td>desktop PC</td>
</tr>
<tr>
<td>Agent based CA Model (Dijkstra)</td>
<td>simulated pedestrian activity in the retail environment</td>
<td>Medium, limited number of influences</td>
<td>yes</td>
<td>lattice</td>
<td>start-goal predefined</td>
<td>no</td>
<td>no</td>
<td>online</td>
<td>modelling frame work</td>
<td>desktop PC</td>
</tr>
<tr>
<td>STREETs (Schelhorn, O’Sullivan)</td>
<td>modelling of large urban areas</td>
<td>medium due to its mesoscopic nature</td>
<td>yes</td>
<td>lattice</td>
<td>multiple start-goal predefined</td>
<td>yes</td>
<td>no</td>
<td>online</td>
<td>C using SWARM classes</td>
<td>multiple desktop PCs</td>
</tr>
<tr>
<td>Robo Soccer (Stone)</td>
<td>game, application of agent technology</td>
<td>n/a</td>
<td>yes</td>
<td>continuous</td>
<td>dynamic</td>
<td>yes</td>
<td>yes</td>
<td>online</td>
<td>C, client-server</td>
<td>desktop PC</td>
</tr>
<tr>
<td>Discrete Force Model for Pedestrian Motion</td>
<td>study of self-organisation</td>
<td>limited to specific scenarios</td>
<td>no</td>
<td>continuous</td>
<td>start-goal predefined</td>
<td>no</td>
<td>no</td>
<td>online</td>
<td>Java</td>
<td>desktop PC</td>
</tr>
<tr>
<td>SimPed (Daamen)</td>
<td>modelling of transport interchanges</td>
<td>High</td>
<td>no</td>
<td>continuous</td>
<td>multiple start-goal</td>
<td>yes</td>
<td>no</td>
<td>online</td>
<td>MODSIM III, VC++</td>
<td>One or more desktop PCs</td>
</tr>
<tr>
<td>Nomad (Hoogendoorn)</td>
<td>modelling of transport interchanges</td>
<td>High</td>
<td>no</td>
<td>continuous</td>
<td>route-choice</td>
<td>yes</td>
<td>no</td>
<td>online and offline</td>
<td>Unknown</td>
<td>desktop PC</td>
</tr>
<tr>
<td>PEDFLOW1 (Kerridge)</td>
<td>proof of concept for PEDFLOW, no practical application</td>
<td>Low</td>
<td>yes</td>
<td>lattice</td>
<td>start-goal predefined</td>
<td>no</td>
<td>no</td>
<td>offline</td>
<td>OCCAM add-on board</td>
<td>Unknown</td>
</tr>
<tr>
<td>Legion (Kagarlis)</td>
<td>wide variety of scenarios (shopping areas, traffic interchange, evacuation)</td>
<td>High</td>
<td>yes</td>
<td>continuous</td>
<td>start-goal predefined</td>
<td>yes</td>
<td>yes</td>
<td>online and offline</td>
<td>unknown</td>
<td>desktop PC</td>
</tr>
<tr>
<td>PedGo</td>
<td>Crowd Movement and Egress Simulation</td>
<td>low, but sufficient for purpose</td>
<td>no</td>
<td>lattice</td>
<td>multiple possible exit routes</td>
<td>limited</td>
<td>no</td>
<td>Online</td>
<td>unknown</td>
<td>desktop PC</td>
</tr>
<tr>
<td>EXODUS</td>
<td>Evacuation simulation</td>
<td>sufficient</td>
<td>no</td>
<td>network of nodes</td>
<td>multiple possible exit routes</td>
<td>yes, limited</td>
<td>No</td>
<td>online and offline</td>
<td>C++</td>
<td>desktop PC</td>
</tr>
</tbody>
</table>

Table 2.2 Overview over the reviewed models classified according to section 2.2.
3 Creating a flexible and efficient design

Based on the conclusions from the literature and the requirements outlined in the introduction, the chapter describes the design of the developed PEDFLOW model. There are two aspects, namely the virtual environment and the agent behaviour within that environment. The way the environment is represented determines the complexity of the required algorithms and therefore the computational demands. On the other hand, the algorithmic rules can only work with information that is captured in that virtual environment. By describing the abstraction process the reasons for the chosen representation is explained, based on the purpose of the model. The architecture is a compromise between the two opposing aims of being able to express sufficient information from the real world in a manner that is accessible to agents and making sure the agents are able to process this information within the constraints of the memory size and processing power available in typical desktop machines.

3.1 Environment modelling

3.1.1 Entities

PEDFLOW is concerned with the modelling of objects of the real world such as pedestrians, but also of street furniture and other items they may encounter, and specifically their change of location over time. In order to avoid confusion with objects in the context of object-oriented programming, they will be called ‘entities’. These entities have a shape and position. Both aspects need to be transferred into the digital domain, where measures can only take on discrete values.

The scenarios to be modelled in PEDFLOW will be restricted to a planar area (street, public place). Movement up and down staircases, escalators and lifts is significantly different to walking at a same level and are not considered. However, slight changes of level (such as ramps, single steps) can be ignored (i.e. considered level) for the purpose of the model. Another consequence of this modelling approach is that the height of an entity (pedestrian, street furniture) is not implicitly captured in the model. While height is not part of the position description, there will be cases where it can be critical in the decision-making process of the agents. An attribute that describes the height as a category can be used for that purpose, but there is no
The shape of entities in the real world is of similar significance. Details are important only if they contribute to the decision-making process of the agents. This is the case for large obstructions, but also if the shape has a meaning. While the latter can be more easily conveyed via an attribute of the entity, the former will need to be modelled. Looking at other models, it is noted that a circle is used as an abstracted shape for a pedestrian, e.g. [16], whereas layout information is always presented by blocks or lines. The circle representation is useful as it implicitly defines how close pedestrians get to each other. While Helbing [16] utilises circles of different size to represent differently sized people, others use circles of uniform size. The disadvantage of the circle shape is that it ignores the body form and has no directional expression, which would be useful interpreting freeze-frame scenarios from the model output.

When modelling irregular shapes in the digital domain, they are depicted as polygons with sufficient numbers of lines/corners. To describe the position of the shape it is enough to capture the co-ordinates of one corner and the relative co-ordinates of the others. In PEDFLOW simplified shapes are used to represent any entities in the modelled environment like items of street furniture and pedestrians. They correspond to the parallel projection of the entity onto the plane (Figure 3.1a).

A Cartesian co-ordinate system provides an intuitive means of recording an entity’s position. Although other systems are conceivable, Cartesian co-ordinates are used most widely [61]. In an urban environment, the majority of features are based on 90-degree angles and are thus easily lined up with the axes.

The location of an entity is specified by the x and y co-ordinates of a fixed point, usually the centre of mass. In case of mirror-symmetrical shapes this is the centre of the shape. Looking towards a computer model implementation, where efficient collision detection in a Cartesian co-ordinate system is required, all objects are modelled as rectangles (Figure 3.1b). The location of the rectangles can again be identified through the co-ordinates of a selected/unified point within the square, e.g. the lower left corner of the shape or the centre. The smallest unit of interest in the model is a pedestrian, hence the size of the atomic entity is chosen to be the size of an average pedestrian.
To accommodate more complex shapes, squares can be combined to form compound entities (Figure 3.1c). This simplification implies that all objects are a multiple size of the atomic object (smallest modelled entity). If the smallest unit in the co-ordinate system is now defined to be length of one side of the ‘atomic’ square, all entities are automatically aligned (Figure 3.1d).

As a result, the modelling environment can be visualised as a grid, where grid elements can be occupied by entities that might be mobile (pedestrians) or form structures (street furniture, buildings).

![Figure 3.1](image)

Figure 3.1 Levels of abstraction in a scenario with building, two pedestrians and a waste bin: (a) stylised shapes, (b) transformed into rectangular shapes, (c) segmented into squares (d) aligned to grid.

The disadvantage of the coarse representation with its impact on distance measurement and derived measures such as density scales is balanced with a simplified representation of rules and better computational performance.

### 3.1.2 Movement

In order to validate the assumption that pedestrian movements can be approximated as movements between grid elements, we look at the way people move. For bipedal beings, movement consists of a sequence of steps. Every step is a routine involving acceleration, side-movement and other properties that are irrelevant for modelling the overall movement in the context of this model. It is therefore desirable to filter them out. One approach is to connect the virtual footprints in sequence. This results in a series of straight lines that describes the route of the person and the time it took for each step. Turns can only occur when a foot is down, as it is the pivot point for the movement. The method fails to adequately deal with time measurement and the zigzag effect resulting from the two feet. A more suitable technique is to consider the moment when both feet are next to each other and consequently the body is located
directly above. Over a journey, the point between the feet at this moment gives a similar trajectory and provides a better means of identifying the time for each ‘step’. Hoogendorn et al [62] show that such sub-microscopic modelling is a concern for high pedestrian densities as they appear in evacuation situations, but they are not within the scope of PEDFLOW.

Figure 3.2 Using footsteps to identify the route of a person.

The result is a number of segments, where the speed is determined by the length of stride and the time it took to make a step. For modelling purposes it is desirable to unify either time or space for all entities in the model and adjust the other to compensate the resulting speed. With the former, the result is a time-advanced system, where during every time unit all modelled pedestrians move a distance that is proportional to their speed. This is the method typically used in cellular automata (CA) based systems, e.g. [30]. The disadvantages here are that all entities will move with every modelling cycle and that the granularity of the grid has to be sufficiently small to distinguish between different speeds. Both result in high computational requirements. A space-based system would work well in a 1D environment. Here the modelled pedestrians would move a unified distance and then ‘rest’ for a time that corresponds to their speed. If the origins of the entities are aligned to multiples of that distance, collision detection becomes trivial. In a 2D world, however, the system can only work for movements in x or y direction; for all other manoeuvres entities would deviate from the grid positions. Diagonal movements can be compensated for by adjusting the time for the extended distance by square root of 2. All other directions need to be approximated as a sequence of straight and/or diagonal moves.
If the size of the modelled pedestrian is chosen to be a multiple of or equal to the step size in order to simplify collision detection, the result is the same environment model as the one resulting from the discussion in section 3.1.1. The environment model can thus be reasonably represented as a 2D grid. A variety of other models use this approach (e.g. [35]) although a number of mesoscopic models (e.g. [37]) employ a system where one grid element can hold a number of pedestrians. However, this method is unsuitable in this context as the microscopic position of entities is not captured. In PEDFLOW, grid elements can be occupied by single entities that are either stationary or mobile. Movement in the grid is limited to neighbouring grid elements giving a maximum of eight degrees of freedom. This is further limited to five directions with the assumption that people will not walk backwards. “Standing still” as a pseudo movement increases the number of options to six again.

Movement speed is modelled as delay between position changes. Thus the resolution of time (a tick is the smallest time unit in the model) and space (the size of a single grid element) directly determines the speeds that can be represented. Table 3.1 gives an overview of this dependency for movements between neighbouring grid elements with a selected choice of temporal and spatial resolutions. Based on empirical measurement, 0.75m was the grid size chosen in PEDFLOW1. This is reasonable as the step size of an average person is in this range. A time resolution of 0.1s gives a usable spread of speeds (Option A): with a typical average speed of 1.4m/s it is possible to model 4 faster speeds and unlimited slower ones (due to the fact that speed and time are inverse, the dependency is not linear). Studies of the passing distances of real pedestrians [63; 64] however, indicate that smaller distances than 0.75m between people are conceivable. Fruin [65] even contemplates distances as small as 0.4m for densely populated areas. When adjusting the space resolution to 0.5m (Option B), the speed spread becomes insufficient and the time resolution needs to be increased as well (Option C). Reducing it even further (Option D) is unjustified as it would merely increase the computational requirements.
<table>
<thead>
<tr>
<th>Option</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>space (m)</td>
<td>0.75</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
</tr>
<tr>
<td>tick time (s)</td>
<td>0.1</td>
<td>0.1</td>
<td>0.05</td>
<td>0.025</td>
</tr>
<tr>
<td>delay (ticks)</td>
<td>speed (m/s)</td>
<td>speed (m/s)</td>
<td>speed (m/s)</td>
<td>speed (m/s)</td>
</tr>
<tr>
<td>1</td>
<td>7.50</td>
<td>5.00</td>
<td>10.00</td>
<td>20.00</td>
</tr>
<tr>
<td>2</td>
<td>3.75</td>
<td>2.50</td>
<td>5.00</td>
<td>10.00</td>
</tr>
<tr>
<td>3</td>
<td>2.50</td>
<td>1.67</td>
<td>3.33</td>
<td>6.67</td>
</tr>
<tr>
<td>4</td>
<td>1.88</td>
<td>1.25</td>
<td>2.50</td>
<td>5.00</td>
</tr>
<tr>
<td>5</td>
<td>1.50</td>
<td>1.00</td>
<td>2.00</td>
<td>4.00</td>
</tr>
<tr>
<td>6</td>
<td>1.25</td>
<td>0.83</td>
<td>1.67</td>
<td>3.33</td>
</tr>
<tr>
<td>7</td>
<td>1.07</td>
<td>0.71</td>
<td>1.43</td>
<td>2.86</td>
</tr>
<tr>
<td>8</td>
<td>0.94</td>
<td>0.63</td>
<td>1.25</td>
<td>2.50</td>
</tr>
<tr>
<td>9</td>
<td>0.83</td>
<td>0.56</td>
<td>1.11</td>
<td>2.22</td>
</tr>
<tr>
<td>10</td>
<td>0.75</td>
<td>0.50</td>
<td>1.00</td>
<td>2.00</td>
</tr>
<tr>
<td>11</td>
<td>0.68</td>
<td>0.45</td>
<td>0.91</td>
<td>1.82</td>
</tr>
<tr>
<td>12</td>
<td>0.63</td>
<td>0.42</td>
<td>0.83</td>
<td>1.67</td>
</tr>
<tr>
<td>13</td>
<td>0.58</td>
<td>0.38</td>
<td>0.77</td>
<td>1.54</td>
</tr>
<tr>
<td>14</td>
<td>0.54</td>
<td>0.36</td>
<td>0.71</td>
<td>1.43</td>
</tr>
<tr>
<td>15</td>
<td>0.50</td>
<td>0.33</td>
<td>0.67</td>
<td>1.33</td>
</tr>
<tr>
<td>16</td>
<td>0.47</td>
<td>0.31</td>
<td>0.63</td>
<td>1.25</td>
</tr>
<tr>
<td>17</td>
<td>0.44</td>
<td>0.29</td>
<td>0.59</td>
<td>1.18</td>
</tr>
<tr>
<td>18</td>
<td>0.42</td>
<td>0.28</td>
<td>0.56</td>
<td>1.11</td>
</tr>
<tr>
<td>19</td>
<td>0.39</td>
<td>0.26</td>
<td>0.53</td>
<td>1.05</td>
</tr>
<tr>
<td>20</td>
<td>0.38</td>
<td>0.25</td>
<td>0.50</td>
<td>1.00</td>
</tr>
<tr>
<td>21</td>
<td>0.36</td>
<td>0.24</td>
<td>0.48</td>
<td>0.95</td>
</tr>
</tbody>
</table>

Table 3.1 Possible speed representations for different space/time resolutions (typical average speeds highlighted)

In summary, entities in PEDFLOW are an abstract, atomic representation of objects of the real world. Their location is determined by co-ordinates in a grid of 0.5m size. There can only be a single entity per grid element, which is then considered occupied or blocked. Large objects are modelled as compound entities consisting of multiple atomic entities. Mobile entities move by changing the occupied grid position to a neighbouring one. Speed is expressed as a delay time associated with a move. It is measured in multiples of 0.05s.

### 3.2 Behaviour modelling

#### 3.2.1 Agent activation

In section 3.1, the movement of mobile entities (and more specifically of modelled pedestrians) is described as a change of position within the grid. Since PEDFLOW is an agent-based model, this movement is initiated by the agent rather than performed by a global algorithm, as is the case with CA models [30] or flow models [66]. Agents in PEDFLOW will move towards their goal in the most direct manner on a path approximated through steps between
adjacent grid elements (including diagonal moves). A straight route, however, will not always be possible. Fixed objects or moving pedestrians might obstruct the path. They need to be observed and a well-defined change of direction might be required as the result. In the model, agents make that decision with every step. They do not plan a certain manoeuvre but conceive the change in an environment as a new situation with every move. The programmatic realisation of this concept is a cycle of activations. An agent activation consists of a sequence of direction determination, observation, parameterisation, rule evaluation and movement, and is followed by a pause of inactivity. The length of the inactivity period corresponds to the time the modelled pedestrian takes to cover the distance between the two grid elements.

Direction determination is the process of finding the shortest way to the goal. Mapped onto the grid and normalised to a single unit length, it represents the straight direction for the agent. It is the only absolute direction used. Observations, positions and movements are considered relative to it.

PEDFLOW agents, very much like the pedestrians they represent, ‘observe’ their environment only in the forward direction. This, however, takes place not only in the grid elements straight ahead, but also in ‘lanes’ of grid-element width on the left and right. Currently one lane to each side is considered sufficient, but future extensions are possible. In each of those lanes an agent will look ahead until it encounters an entity in its path. The results of the observation are the category of the object and its distance for each of the lanes. Values like distance and speed are categorised according to the profile of the individual agent. The parameterised observations, together with further individual agent parameters, are used as input for a set of rules. The rules evaluate a possible change of direction and a change of speed. The agent will move one step in the adjusted, determined direction, freeing the current grid element and blocking the new.

It is important to understand the distinction between ‘rules’ and ‘parameters’ in PEDFLOW. While the rules capture a general concept, such as the avoidance of an obstruction, the parameters modify or adjust the rules according to the individual agent’s profile. Examples of parameters are measures like the personal space measure and the yield distance (which describes the way
pedestrians perceive the distance of entities). While these can take on absolute measures (e.g. in metres) the actual rules are based on concepts like “close” or “far”. The translation is realised via agent parameters. There is only one set of rules in the model that all agents use, but their individual behaviour will vary as the result of their different parameters. The calibration of these parameters is explained in section 4.3.

3.2.2 Cause-effect rules

Algorithms representing rules need to be able to evaluate any possible combination of parameterised observation values for all available lanes. In other words, they map an output (typically movement direction and movement speed) to every possible set of inputs (observation categories). There are several approaches by which this can be achieved which differ in complexity, abstraction level, intuitive and computational requirements.

They range from AI scripts that are interpreted through an Expert Systems Shell class (e.g. Jess [67]), and manually constructed decision trees, to lookup tables that hold all possible input combinations, together with their result values. Implementing behavioural rules as interpreted scripts is too inefficient. An expert system implementation that pre-compiles rules to code (which can be efficiently parsed or even directly executed) would be the best balance between design-friendliness and speed. In the design phase, new rules could be easily inserted in the source code and re-compiled for fast execution. Development of such a compiler, however, is an extensive task and is outside the scope of this thesis. Decision trees, as an alternative low-level approach, are difficult to manually construct and maintain and a complete lookup table is too big in volume.

The PEDFLOW1 model [48] uses a sequential rule table, where the number of rules is reduced through either the use of wildcards for input variables that do not contribute to the result (“don’t care” inputs) or where similar input configurations are merged. Here input variables, that have the same effect on the result of the rule, are replaced with an alias that covers them all, resulting in a smaller number of rules.

The disadvantage is that such a table is difficult to maintain. The order at which rules appear is significant, making it difficult to change rules individually without
affecting others. As the rule table is subject to change in the development phase, the optimisations need to be reversible. Secondary, unwanted rules are likely to occur. Consequently, while it is sufficient to test individual rules, it is impossible to cover all possible input combinations.

These shortcomings can be addressed by having a complete rule table. However, with only very few categories and combinations it becomes too large to handle, populate and maintain (range of $10^6$). A solution is shown in [68]. By eliminating impossible combinations of input values, efficient set arithmetic and intelligent rule-merging, the size can be reduced to about $10^2$ rules. This is supported by an editor with built-in mechanisms for consistency check and rule manipulation.

During the course of the project it became clear that although a complete rule-table would be able to capture every possible situation, the problem was to determine what the rules actually are and to set up the table accordingly. In a prototype implementation the rules were chosen pragmatically (according to the programmers’ personal understanding and experience) but due to their complex interconnections they were impossible to validate against real world measures. For example: walking speed can be measured for unobstructed walking pedestrians and it can be assumed that this is their desired walking speed. Other measures such as personal space measure (which had been used as a parameter in the rule set), already associated with pedestrian movement by Fruin [65], can not be easily obtained by video analysis or other empirical studies. For these reasons the rule system had to be revised. This lead to the idea of having a context dependant rule system, where simple rules are evaluated and the executed action is determined by a transparent selection mechanism: context-mediated behaviour to be dealt with in more detail in the next section.

3.2.3 Context-mediated behaviour

Consider a basic mode of moving towards a goal while avoiding obstructions. The agents “observe” a rectangular area in the movement direction, similar to real pedestrians looking forward. The observation area is three grid elements wide (a left, a straight and a right lane) and DD grid elements long. DD is the deviation distance (specific to each agent) which refers to the maximum
distance to an object at which the agent will start to deviate. A statistical
distribution for such a value can be easily obtained from the empirical study of
video data [69]. If another entity is positioned in the middle lane, the agent will
deviate to the left or to the right, choosing a lane that is unobstructed.

It is of course possible that both lanes are free of obstructions and the agent
has a 50% chance of choosing either one. Observation of pedestrians in the
urban environment shows however that this split is not at random; in fact,
pedestrians will choose directions for any number of reasons. For example, if
they walk with a partner they might consider stepping in front or behind the
person in order to not get separated; if they determine the intention of an
oncoming person to pass them on one side they will chose the other; or if one
choice would bring them closer to a dangerous area (e.g. a road) they will opt
for the safer direction. The model requires a way to capture these influences in
a well-defined and flexible manner.

As a consequence the concept of context-mediated behaviour was introduced
into PEDFLOW. The idea is to consider selected, crucial influences
independently and treat each one as a context that contributes to the decision
of a pedestrian. Every context will be evaluated in isolation and a weight
assigned to the outcome that describes the importance of the influence and the
likelihood of the particular action occurring. Contexts can result in multiple
possible actions - in the example above the basic context can result in moving
straight ahead or a 50/50 likelihood of moving left or right. Different contexts can
have actions in common. Figure 3.3 illustrates this case. The “basic” context (a)
has two possible actions (left or right), the “walking with a partner” context (b)
has one (right, towards partner). The action to be performed is chosen
according to the sum of weights from the contribution contexts (c). Assuming
the same weight, the move to the left has a probability of 33.3% to be executed
and the one to the right a probability of 66.6%. This method of managing
contexts has the advantage that influences can be added or removed easily
during model development and their effects investigated.
So far only movements have been considered as possible actions, but pedestrians can also “act” in other ways. A special case of a movement would be a pause – the agent does not change position, but re-evaluates its options again after a certain time has passed. Other non-movement actions are possible as well. Typically they have an amount of time associated with them and therefore are a special case of a pause (e.g. listening to a street musician for a while). Some actions are instantaneous (signal to another agent) or include waiting for an external event (e.g. green pedestrian signal). The former is trivial to implement as it is the action followed by an immediate re-activation, the latter can be realised either by repeatedly checking (polling) of the entity responsible for the event or by means of a re-activation queue within the event-entity. The choice of implementation for a specific context will depend on the type and frequency of the event and the number of entities involved.

When selecting an action from all possible alternatives according to weight, one option is to always choose the one with the highest weight. This makes the model deterministic. Experience shows that this is not always desirable as it leads to some actions dominating and others being ignored. Additionally, it may occur that actions have very similar weight totals which results in an unfair disregard for the runner-up. One solution is to select an action randomly so that actions with a higher weight have a greater chance of being selected. Linear dependency appears to be sufficient and there is no indication that other transformations which lead to a preference of higher weighted actions (e.g. quadratic) would lead to more realistic results.
There is no universal way to derive the values for the weights of the different contexts using observational methods; the method will always be case dependent. The suggested method is to start with few contexts where the importance can be roughly determined by experience and/or deduced from empirical study of pedestrian behaviour. In an iterative process of modifying the weights, a modelled behaviour that approximates to the behaviour of real pedestrians can be achieved. The process can be illustrated using the example of an attractor context that is to be added to the contexts illustrated in Figure 3.3. The new situation is shown in Figure 3.4 and it appears that there is now a 50% chance for the agent to move either left or right (indicated by the two arrows pointing to either side). By adjusting the weights this relation can now be changed to increase the likelihood for the person to move towards the partner. With an assigned weight of 0.1 for the attractor context, there is now a $(1+0.1)/(1+1+1+0.1)^*100 = 0.35$ chance of the agent to move towards the attractor and a 0.65 chance of it moving towards the partner.

![Figure 3.4 Adding an additional context (attractor).](image)

When introducing new contexts, they will be validated with a small set of contexts first before combining them with the whole set. The iteration process can be shortened by prioritising the contexts in groups, e.g. essential, alternative, irrelevant contexts, and adjusting the weights within these groups more precisely.

Two special cases need to be considered separately: absolute actions and disallowed actions. Absolute actions are actions that take absolute priority over any other possible outcome. For example, if the agent has reached its goal, there is no reason to consider other possible actions - the action of goal update is the only choice. In this case and in similar contexts the context evaluation can
be aborted and the action selection process omitted. Similarly certain contexts do not result in an action, but rather block other actions. For example, a red light should prevent an agent stepping onto the road by disallowing actions from other contexts that would require just that. All other contexts still need to be evaluated (i.e. it is not an absolute action). All evaluated actions will then be post-processed to eliminate actions that must not apply. From the remainder an action is determined according to its weight. If no possible action remains the default action of pause will be applied.

Every context requires up to two independent activities by the agent: context detection and context evaluation. Context detection is the process where the agent determines whether a certain context is applicable or not. This can take the form of an observation of the virtual environment, an interrogation of an attribute of another entity, the check of an internal flag or a combination of two or more conditions. The internal flag can be set by an external entity or be a counter field that is updated under certain conditions. An internal flag is also used to keep track of recurring conditions. Let’s consider for example the modelling of a street musician, or (more abstract) an attractor. The attractor should only attract an entity once. When the entity activity associated with the attraction (move towards attractor and wait) has elapsed, the entity mustn’t be attracted again.

Once it has been determined that the context applies, possible actions will need to be derived. Typical actions are: move to adjacent grid element, swap position with entity at adjacent grid element (to model passing in narrow corridors), pause (un-schedule), wait for event (queue), send signal to agent, insert new sub goal, adjust sub goal, remove sub goal. While “move”, “swap” and “pause” incorporate a time period after which a new activation is scheduled (and thus contain the concept of speed of the action) the last three actions can happen instantaneously. Additional actions can be added as required without breaking the design.

Contexts can be classified as external (set from the outside of the entity) or internal (detected from the value of an internal flag or timer), and persistent (lasting over a number of activation cycles until turned off) or transient (only valid for the current activation cycle).
To illustrate this concept, the “walking with a partner” scenario will be looked at in a little more detail. Research [70] shows that the majority of non-commuting journeys are performed not by a single person but by groups of people and that group behaviours are significantly different from those of singletons. These differences are difficult to capture in the model. Some features, like reduced average walking speed, can be realised with adjusted parameters; others require elaborate algorithms. One typical group feature is the desire of the group to stay together - even if that means individual disadvantages like pauses or detours are required. For pairs of pedestrians (the dominant group size) we try to capture this behaviour in the context “walking with a partner”. For bigger groups, more complex algorithms are required to implement “leader-followers” behaviour.

The prerequisite for the scenario walking with a partner is that both agents have the same desired speed and the same goal. Each agent in the group contains a reference to the other agent. Under normal circumstances (basic context) they will walk next to each other. If, however, an additional context requires the agent to pause, slow down or deviate from the common path (e.g. it is faced with an obstruction), special action is required. One part of the solution is to reduce the likelihood of such situations occurring. Each agent flagged as walking with a partner will reduce the likelihood of actions that move it away from its partner. If such an action does happen, either one or both partners will enter the context “lost partner” and initiate actions to come together again. Depending on the situation the partner will pause or (more likely) move towards the other agent. This does not necessarily happen immediately, as it will take a certain minimum distance between the entities (upper limit) to become aware of losing the partner. Once it does happen, the agent will only stop moving towards the partner once a distance is under a certain maximum range (lower limit). The limits can be different for both partners and reflect the affinity of the pedestrian to its partner. This is illustrated in Figure 3.5.
In situation (a), agent A has detected the loss of its partner (outside outer circle) while B has not (still inside outer circle). Agent A will therefore initiate action to bring itself close to B. If it doesn’t succeed and the distance increases, B will detect the loss as well at some point (outside outer circle) as illustrated in (b). Now both agents will attempt to close in on each other. In situation c), agent B will stop any activities (A is within inner circle) while agent A still needs to get closer. Section 5.1.2 investigates how realistic the emergent behaviour from this context model is in comparison to real world pedestrian behaviour.

Context-mediated behaviour is a novel approach to dealing with multiple concurrent influences and as such a thorough evaluation and validation is required. The next chapter describes how the mechanism and the other design decision laid out so far are implemented in a software prototype to enable such an evaluation.
4 Building the prototype application

The implementation of the architecture presented in Chapter 3 must preserve the extensibility of the design while at the same time be fast enough to run on an office desktop computer. As a prototype it has to provide sufficient means of interaction to support set up, data input, data output and experiment control. This chapter also includes details on how the data used as input to the model was obtained and how the output from the model can be used.

4.1 Implementation aspects

4.1.1 Programming language

Since computer modelling has become ubiquitous in recent years, a great variety of tools have been developed that allow the modelling of dynamic processes in several ways. Examples for such modelling toolkits are SWARM [71], CYBELE [72] or ARA [73]. They either provide a specialised programming language [74] or extend the functionality of an existing one by adding a set of function libraries [39]. Distributed computing support makes large-scale simulations possible.

The advantage of such specialised software products is that they allow fast prototyping in standardised environments. They are typically optimised for performance and/or ease of programming and are supported by a large user community.

On the other hand, modern universal programming languages are becoming more and more powerful. They provide all the necessary constructs to quickly create prototype products. Library functions simplify routine tasks and interactive tools help with the design of user interfaces.

Universal modelling software can greatly improve the speed of model development and implementation because many facilities already exist which only need to be adapted to the specific model requirements. However, such a model framework shows its limits at later stages, for instance when algorithms which are outside the original concept can’t be easily realised. An example would be event driven activities in a cycle-based modelling system or hierarchical data structures in a flat file system. They typically also provide
some means for user interactions, but on a technical level not suitable for use by an untrained person such as an urban planner.

The ownership of such modelling frameworks is another concern. The model becomes dependent on the manufacturer and potential users are forced to pay for an extra license as is the case with PEDROUTE [75] which requires a licence for EMME/2 to run. There are open-source models available [71] which might lack the commercial support but have a broad academic user-base.

Commercial software is often written for a single platform – for example, UNIX clusters are required for CYBELE to work [72], which could pose a problem if the model is to be used in a PC environment.

This dilemma is reflected in the many solutions other researchers have used to implement their model: Streets [37] uses SWARM [71], a software package for the multi-agent simulation of complex systems. Modsim III [76], an object-oriented, discrete event simulation language is used for example in SIMPED [77]. On the other side, the behavioural force model by Helbing [66] is implemented in Java and so are the CA based models (e.g. [78]).

Fortunately the difference between specialised modelling products and universal programming languages has become smaller and smaller as toolkits provide more and more flexibility and language has more and more specialised facilities. Increases in computing power remove the performance advantage of specialised modelling packages. The decision on what environment to use for the implementation can be based on other factors such as availability, portability and cost. It is a good compromise to use a general programming language with libraries to support specific tasks in the modelling process.

Java was chosen mainly because of its popularity and the consequent available support. The Java white paper by Sun Microsystems [79] states that "Java is …simple, object-oriented, distributed, interpreted, robust, secure, architecture neutral, portable, multithreaded, and dynamic." Several features make it especially suited for the modelling task.
Simple, single inheritance object model

In a microscopic model, many similar independent entities need to be represented. The object approach lends itself to this purpose as it allows easy re-use of common features and methods between entities.

Multi-threading

The multi-threading model of Java, even though it is not as efficient as that of more specialised parallel processing languages like OCCAM [80], is easy to use and capable of handling simple concurrent tasks independently.

Portability

Associated with the phrase “Program once, run anywhere”, the platform independence of Java is exceptional. Run-time environments exist for virtually any hardware platform and operating system. This is essential for an application that is to run in an office environment.

These reasons make Java the programming language of choice for PEDFLOW.

4.1.2 Modelling framework

The PEDFLOW software has a modular design as outlined in Figure 4.1. The heart of the program is the simulation module. All of its functionality is exposed via an interface definition. To modify its behaviour the preference module contains a number of settings, which can be altered by a user interface. The output of the model can be displayed, manipulated or simply saved to the data storage.

Figure 4.1 Modular structure of the PEDFLOW Model.
The modularity allows for many different configurations simply by ex-changing classes for classes with different functionality but identical interface. The basic stand-alone configuration would contain elaborate versions of all four modules allowing for maximum access to all features. For batch-runs of already designed simulations a version without user-interface or display is possible. Web presentations for demo purposes with limited interaction only require a simplified user interface and don’t need not write to the local file system. The display module could pass on the output of the model to a 3D rendering engine for a 3D representation of the simulation, in which case the output model would contain a format-converter and some means of data transmission.

Figure 4.2 Classes in PEDFLOW (dashed boxes indicate derived sub classes).

Since the PEDFLOW model is an object-oriented program, it can be best understood by looking at the various classes, their use and relationships. Figure 4.2 gives an overview. The most important are the entity classes used to represent real life objects ranging from street furniture to pedestrians. To keep track of them within the software model, a set of modelling classes (Grid and
TimeSlot) are used. The user interface classes (within the MainFrame class) allow control of the model. They use, and are mainly based on, Java Swing classes [81]. Swing is a graphical user-interface toolkit, consisting of a set of classes and interfaces to standardise interface implementation and to provide a consistent look and feel. Last but not least there are many utility classes used to encapsulate and pass information within the model or just to generally make the workings more transparent.

4.1.3 Concurrency

People in the real world move concurrently, therefore the natural way of implementing a computer model of pedestrians would appear to be to represent them by a set of parallel processes. This was the approach taken in the OCCAM prototype implementation [48] and led to an intuitive design due to OCCAM’s unique parallel programming constructs. A parallel approach can also be found in other agent based models (e.g. Swarm [39]). However, most traditional CA based models (e.g. Blue and Adler [78]) use a sequential approach. Parallel computer programs are more difficult to create and to debug than sequential ones [82], unless sufficient support is provided by the programming language.

Multi-threading is natively supported in Java, but its realisation varies depending on the Java run-time environment and operating system [83]. Native threads come as one-to-one or many-to-many implementations depending on operating system and java development kit (JDK) version. ‘One-to-one’ means that each thread is basically a clone of its parent process, the threads have higher overhead for context switching and creation, and a larger memory footprint. The number of threads is limited by the number of processes/tasks built into the OS kernel, whereas many-to-many doesn’t have this limitation. At the start of the project only the latest versions of the SUN JDK implemented native threads, but now most current Java run-time environments provide excellent multi-threading support [84].

With JCSP [85] a set of java classes exist that implement concurrent sequential processes (CSP) [86], the programming approach used by OCCAM. It is based on concurrent processes that communicate via synchronising channels and thus is more intuitive than the native Java thread model. The programming
overhead however cannot be ignored. Mixing paradigms (CSP processes and pure Java) will make the resulting code hard to read and understand.

Looking at the proposed design of the PEDFLOW model, it will be noted that the potential for concurrent actions of the pedestrians is not as high as expected. Early tests indicated that for an average scenario typically only 1/5 of the agent population will be active at a certain time. Since the selection of this fifth from the agent population is arbitrary (depending on speed and starting time), there is no reason to provide special provisions (scheduling overhead, locking of grid elements) for a subset of agents only. In order to exploit the advantages of a parallel implementation while avoiding the potential problems, a result a hybrid version was developed which is described in the next section.

4.1.4 Model dynamics

Models of dynamic processes are usually either event-driven (event advanced) or cycle-based (time advanced). In an event-driven simulation model, activity occurs as a series of asynchronous events happening at irregular intervals. In cycle-based simulations, all system changes are synchronized to a single clock. The two approaches can be combined in a simulation that includes clock ticks as a simulation event.

PEDFLOW uses a modified version of such a combination. In this hybrid approach, a clock tick will generate an event only for an individual entity that is scheduled for activation. It follows that not all entities are activated with every clock tick. Also clock ticks do not need to be at regular intervals – it is sufficient that all scheduled activity has finished before the next tick. Apart from clock ticks, an activated entity can also generate events for other entities. Further events can be generated through the user interface.

Tick-based activation is managed through instances of the TimeSlot class: handleInitTimeSlot (entities not in the simulation) and handleGridTimeSlot (entities currently in the simulation). The objects are part of simulation.Java. A TimeSlot is a ring-buffer where every slot contains a list of entities to be processed at a given time. The handle functions do just that - they go through all the entities in the current slot, performing the following actions: check the entities loopcount variable, and if it has expired either drop the entity on the grid (handleInitTimeSlot) or call its activate method (handleGridTimeSlot). Typically
the entity is then re-added to the appropriate TimeSlot at a new position that corresponds to its activation delay.

In order to progress the simulation one just needs to advance the buffer pointer and call the two handleTimeSlot methods (as implemented in the step() method). For continuous simulation call step() in a loop. In reality, however, there are additional requirements to this ‘run as fast as possible’ approach, concerning dataflow and user interaction.

Through this mechanism a flexible scheduling is achieved, as the entities control themselves, the pause between their activations and thus their speed (see also section 3.1.2). The design guarantees that the next position in the ring-buffer will only be processed once the current position is exhausted.

If a timeSlot position is empty, it will be skipped. The gap between two non-empty slots is called a time-step. Since all the activations of one time-step happen at the (conceptually) same time, there is only one display update per time-step. If the step time is equal the buffer length it means no active entities are scheduled for activation and the simulation has finished. It can be halted by stopping the calls to the step method.

Methods of the simulation class that control the scheduling are:

runToChange() start simulation thread to run until change in display
runToEnd() start simulation thread to run (until aborted or out of entities)
runToTime() start simulation thread to run until time has passed
runToTimeJoin() blocking version of runToTime()
stop() is used to abort any of the previous methods prematurely.

As part of the user interface, a concurrently updated display can be provided. The Swing mechanism for displaying is based on a call-back paradigm. Every object that has a visual representation provides a method to draw itself. The method is called under various circumstances such as window move, window resize etc., but can also be requested by the object itself if its representation has changed. For this to work, it requires the activation and the display process to run concurrently. The two threads (activation and display/user interaction) can be visualised as shown in Figure 4.3.
**Figure 4.3 Timing of activation and display thread.**

*Timeslot* classes are used to keep track of model time. They are loosely based on the bucket construct [87] but work on a higher level. They account for the fact that with a growing number of modelled entities, the maintenance of timer queues to control their periodic activation becomes computationally expensive. *Timeslots* get round this by clustering entities with the same activation time in a collection class or *slot*. These collection classes, implemented as *ArrayList*, are arranged in a circular buffer, where two neighbouring slots are activated sequentially with a delay that represents the smallest possible time delay in the model. If the computation time required for that cycle is in fact smaller that the modelled time delay, experiments can run faster than real time. Figure 4.4 illustrates this dependency. It shows four time periods (cycles) modelled in real time and faster then real time; bold lines indicate actual processing time.

**Figure 4.4 Model time compression with timeslots.**

The necessary size of the ring is therefore the maximum required delay divided by the minimum delay time. The *Timeslot* class maintains an internal position pointer with a method for its advancement, so the wrap-around is encapsulated. There are methods to retrieve entities from the current slot and methods to
deposit entities at a positive offset to the current slot. The size of the offset gives the delay that will be experienced by the entity. Currently there are two uses for the *timeslot* class: the activation of entities already in the model and the insertion of entities at specified time during the model run. The insertion at model start is a special case of the latter.

### 4.1.5 Modelled entities

Static and mobile entities populate the grid and represent buildings, various items of street furniture and other objects in the real world. They can be divided into static passive entities (e.g. obstructions), static active entities (e.g. attractors), mobile entities (e.g. vehicles), compound entities (e.g. buildings) and quality of area (e.g. pavement). Entities have associated parameters that describe them and their behaviour (if applicable). The simplest entities in the model are obstructions with only the co-ordinates as parameters and the most complex entities are the agents used to represent pedestrians with a multitude of parameters describing their behaviour. Since parameters are often applicable to more than one entity type, entities are implemented as a class hierarchy (Figure 4.5) where entities inherit the parameters from all classes that are higher up in the tree. For example a Person inherits x and y co-ordinates from Obstruction, start time from Active and goal co-ordinates and speed from Mobile. A more detailed description of the implemented entity classes and their parameters can be found in the appendix.

![Figure 4.5 Class inheritance for entities in PEDFLOW.](image-url)
Typically all entities of a particular type have the same set of parameters, but some parameters are optional or dependent on the value of other parameters. All parameters are of type integer and can be represented by a single, constant number or a distribution. The distribution class is used to implement the concept of a number that is chosen according to a statistical distribution. The simplest form is the even distribution where all numbers in a certain specified range would be selected with equal probability. Other forms currently implemented are normal distribution (mean, standard distribution and range) and an arbitrary list distribution where every value has an associated probability value.

Apart from having properties, entities can also have relationships with each other. Examples are two people walking together or the effect of a set of traffic lights. Such relations are captured as object references in PEDFLOW. If an entity is part of many relations these references become quite complex.

4.1.6 Data storage

The information that needs to be stored for a scenario consists of general/global information describing the modelled scenario (such as area dimensions, but not origin/destination information) and information specific to entities (local, temporal and behavioural). The complete data set describing a specific scenario can be interactively generated using the editing facilities of the software. All the necessary objects will be instantiated and parameters populated. This process is labour intensive and since experiments are typically run several times, it is desirable to be able to store the information on a mass-storage device. The storage format needs to be flexible and expandable to accommodate future additions. Ideally it would be readable by software other than the PEDFLOW user interface for debugging and batch processing. It should allow the reuse of data subsets to easily model the same street layout for different populations of pedestrians or investigate layout changes with a particular set of pedestrians.

There is very little information available in the literature, about the type of data storage used for microscopic pedestrian models. Researchers have used whatever mechanisms their platform provided and not considered data storage an important enough part of their model for publication. Fortunately similar storage requirements exist in other application areas and, as a result, methods and standards have been created that can be adapted for use here.
Originally a flat text file was used in PEDFLOW to store the information, where every line represented an entity and every column a specific parameter (Table 4.1). Parameters that weren’t applicable to a particular entity would contain a null value as a placeholder. Additional information that didn’t fit into this scheme (e.g. global and control data) was held in a separate file in the form `<parameter> = <value>`. This worked reasonably well with few different entities and a limited number of scalar parameters in the early prototypes of the PEDFLOW model. The fact that the tables were editable with a simple text editor was a great benefit in the initial stages of the project, when a built-in editor didn’t exist or was very limited in functionality. However, it became unworkable in the later versions of PEDFLOW, when more and different entity types were introduced and parameters could contain not only single values but a variety of expressions.

<table>
<thead>
<tr>
<th>entity</th>
<th>x-pos.</th>
<th>y-pos.</th>
<th>x-goal</th>
<th>y-goal</th>
<th>speed</th>
<th>....</th>
</tr>
</thead>
<tbody>
<tr>
<td>person</td>
<td>10</td>
<td>5</td>
<td>10</td>
<td>15</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>obstruction</td>
<td>10</td>
<td>10</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.1 Simplified example for entity storage as flat file.

A different solution was required. The literature suggests various methods of achieving object persistence [88] such as object databases and binary files that don’t rely on table structures. They are based on implementing the java.io.Serializable interface introduced in JDK 1.1 which declares the methods writeObject and readObject. Nomad [44] uses object serialisation to store its state.

The writeObject method is responsible for writing the state of the object such that the corresponding readObject method can restore it by writing the individual fields to the ObjectOutputStream using the writeObject method for object data types or by using the methods for primitive data types supported by DataOutput. Conversely, the readObject method is responsible for reading from the stream and restoring the fields of the class. The method uses information in the stream to assign the fields of the object saved in the stream with the correspondingly named fields in the current object.
The object serialisation solutions differ, however, in where the resulting data stream is stored and how this storage area is accessed. Interfaces of varying functionality and complexity are supplied to organise the data and often they rely on back-end object databases for storage. This, however, is an inherent disadvantage for a project like PEDFLOW where portability and the ability to run independently of additional software is a requirement and data needs to be exchanged in a simple way. For that reason all approaches involving external databases and interfaces like java database connector (JDBC) are unsuitable. All the data pertinent to an experiment needs to be stored in a single file. This is also possible using the java.io.serializable interface but will result in a binary data file that is very difficult to edit without a dedicated application.

Only a text file could provide the flexibility needed. Instead of using a data format where data is accessed by its position in the file, a label-based concept was selected. Given the choice of designing a new format or employing an existing and proven concept, the increasingly popular XML (Extensible Mark-up Language) was adopted.

Marking up text is a methodology for encoding data with information about itself [89]. Rather than providing a set of pre-defined tags, XML specifies the standards with which you can define your own mark-up languages with their own sets of tags. Thus XML is a meta-mark-up language which allows the definition of other mark-up languages based upon the standards defined by XML. XML is not limited to text mark-up. For an XML document to be useful (processable by a computer application) it needs to be well formed (i.e. the syntax needs to be correct) and valid (i.e. it conforms to specified structure) [90]. In XML, the definition of a valid mark-up can, for example, be handled by a Document Type Definition (DTD) which defines the structure of the data.

As a first step in mapping the PEDFLOW class structure to a XML structure, it needs to be transformed into a tree. Figure 4.6 shows the object relationships. A grid object with parameters describing its properties contains a number of entities. Each entity in turn is described by its parameters. Parameters can be of single value or have a distribution, where a distribution has many (single-value) parameters. Entities can have relationships with each other.
The obvious problem here is in expressing the relation between entities (e.g. two people walking together) which is implemented by means of an object reference in the program. While an object ID could be used, this would introduce additional parsing requirements and potential integrity problems. For that reason it is more desirable to capture such relations within the XML structure. In the adopted solution, so-called phantom entities are introduced. They are entities without an equivalent in the real world and they act as wrappers to encapsulate a group of entities that are related to each other. As a simplification, the wrapper functionality can be merged into one of the entities of the group. This becomes more intuitive when one entity is different already, for example a group leader and a group. The phantom entities solution has the disadvantage that an entity can only be part of one relationship. As this is typically the case for PEDFLOW, it is not currently a problem.

Once a tree structure has been established it can be transformed into XML by adding the mark-up to its text representation, which is a way of labelling the elements. The boundaries of elements are delimited by descriptive start-tags and end-tags. The start-tag consists of the element name surrounded by ‘<’ and ‘>’ while the end-tag has an additional ‘/’ immediately in front of the name.

Applying this to the PEDFLOW data means that single parameter values are enclosed in tags containing the name of the parameter. For parameters based on a distribution, the actual number is replaced by a tag that indicates the distribution type. This tag encapsulates all the tagged, scalar parameters required to define the distribution. A similar approach is used for entities. However instead of using an individual tag for each entity type, there is only a
generic tag (<entity>). The type of the entity is specified by means of a <label> parameter. This alternative solution provides for easier processing of the data with regards to parameters that are common between different types of entities.

The sum of all valid XML data structures resulting from this procedure is described by means of a Document Type Definition (DTD). The purpose of a DTD is to define the legal building blocks of an XML document. It defines the document structure with a list of legal elements and thus provides a template against which a parser can validate an XML document. A DTD contains a list of all valid tags for an XML document and a description of what elements they can contain. Unfortunately the syntax of a DTD is quite restrictive. There is no data type definition apart from text (#pcdata) and for complex nested structures with many optional elements a DTD is very hard to write and read. There is also no way to describe dependencies other than by grouping items together, which is not always possible. An alternative to DTD is XML Schema Definition (XSD), which is XML-based itself. As a result it is richer in expression and more expandable. It also supports data types. At the time of writing, however, it is not supported by the parser. Consequently a DTD is used for PEDFLOW, even though it requires additional data validation in the code. But it is a good way to describe and document the XML structure for humans and a transition to XSD is possible in the future with little effort.

To illustrate the use of XML for PEDFLOW, the following is a simplified example scenario. A narrow strip of pavement is 1.5m wide and 10m long. A pedestrian, starting on one side (x, y co-ordinates), is trying to go to the other end of the pavement (goalx, goaly). Its path is blocked by an obstruction in the middle of the pavement. In order to describe this situation for the PEDFLOW model, the grid must be defined. Its parameters are x = 20 units and y = 3 units. Space and time units have the default value and can be omitted. The grid contains a person entity and an obstruction entity. The parameters of the obstruction are simply x = 10 and y = 1 whereas the person entity has the parameters of x = 0, y = 1, goalx = 20, goaly = 1. It also contains a variety of parameters that determine its behaviour. In order to keep it simple only one of them, desired walking speed (DWS), is considered. DWS, however, is not a constant, but is to be chosen from a distribution. Again for the sake of simplicity it is assumed that DWS is equally distributed between 100 cm/s and 160 cm/s.
The resulting XML including the simplified DTD (embedded for illustration purposes) is shown in Table 4.2.

```
<?xml version="1.0" ?>
<!DOCTYPE grid [
<!ELEMENT grid (x,y,space?,time?,entity*)>
<!ELEMENT equal_dist (minvalue,maxvalue)>
<!ELEMENT minvalue (#PCDATA)>
<!ELEMENT maxvalue (#PCDATA)>
<!ELEMENT label (#PCDATA)>
<!ELEMENT x (#PCDATA |equal_dist)>
<!ELEMENT y (#PCDATA |equal_dist)>
<!ELEMENT goalx (#PCDATA |equal_dist)>
<!ELEMENT goaly (#PCDATA |equal_dist)>
<!ELEMENT dws (#PCDATA |equal_dist)>
<!ELEMENT entity (label,x,y, (goalx, goaly, dws)?)>]
<grid>
  <x>20</x><y>3</y>
  <entity>
    <label>blockage</label>
    <x>10</x><y>1</y>
  </entity>
  <entity>
    <label>person</label>
    <x>0</x><y>1</y>
    <goalx>0</goalx><goaly>1</goaly>
    <dws><equaldist><minvalue>100</minvalue><maxvalue>160</maxvalue></equaldist></dws>
  </entity>
</grid>
```

Table 4.2 Example PEDFLOW XML file with embedded DTD.

Having found a convenient way to express the PEDFLOW data structure in XML, it is now important to integrate it seamlessly into the existing software. Since the existing program structure has to be maintained, it is a matter of reading the XML file, creating the appropriate grid class and populating it with the required entities and parameters according to the XML. Additionally, a mechanism is required to export all data from the program as XML text.

There are two established approaches for reading and parsing XML files, the Simple API for XML (SAX) and the Document Object Model (DOM). Implementations for both APIs exist in a variety of languages, including Java. The SAX API serially reads the XML document and fires events for each element tag encountered. The one-pass sequential parsing makes it a highly efficient approach for data transformations or one-time population of data structures. The DOM API on the other hand will read the XML document and create and populate a tree structure in memory based on the hierarchy contained in the XML file. The elements of the resulting hierarchical object model can then be accessed and manipulated. The drawback of this flexibility is an increased memory requirement.
Although the general recommendation for machine-readable and generated data is to use SAX [90] in the case of PEDFLOW the object model approach was chosen. One reason is the complexity of the data structure. Writing event handlers for the variety of data by itself would be a daunting enough task, but also keeping track of the context by means of state variables is not feasible, especially considering the possibility of nested (phantom) entities.

The other reason for choosing DOM over SAX is the property inheritance in entity objects. When initialising an entity, the initialisation method of the parent class is called first in order to deal with the inherited properties. This works elegantly with the DOM API where the object model tree matches the entity hierarchy and just the relevant branch can be passed on to the parent class’ method. SAX event handlers on the other hand would need to keep track of the hierarchy level requiring cumbersome state maintenance.

Saving PEDFLOW data as XML is even simpler, bypassing the DOM. Every entity contains a toXML method that returns a string representation of all its parameters marked up as XML. Since the PEDFLOW XML structure mirrors the entity class hierarchy, a call to the superclass’ method will include all parameters inherited from it. Thus the save operation (i.e. the sequentialisation) is just a case of first writing the grid definition and then to iterate over all contained entities executing their toXML methods.

Figure 4.7 summarises the data flow within PEDFLOW, emphasising the added or modified components. While in the original version all entities were instantiated from the text file, there is now the added step of the DOM tree, created by the XML parser. This means that the complex parsing and validation process is performed by a library call. It also only happens during the initial load of the experiment. Consecutive runs will use the parsed data from the DOM tree. The Figure also indicates that only the constructor and save methods of the entities had to be modified to implement XML data storage so there is no impact on the function or performance of the model.
With the mechanism for XML reading and writing in place, the next question is whether to use the format for other data associated with PEDFLOW, namely the configuration file or the output data. For the configuration file, which merely holds a number of (parameter, value) pairs, the required implementation effort seemed to be inappropriate as there are no real benefits. As far as the model output is concerned, a similar situation exists. Most spreadsheet and statistical packages today have no import facilities for XML and thus a comma-separated variables file (*.csv) is more appropriate. Even if this situation was to change the structure of the data is simple and doesn’t justify the overhead of XML.

### 4.2 Agent characteristics of PEDFLOW entities

With respect to the categories from section 2.2, the PEDFLOW model is a microscopic agent-based model. The purpose-built environment that the agents live in is represented by a lattice with a default grid size of 0.5m. The agents work to complete a list of pre-set sub goals which can be adjusted by context-mediated behaviour. The source of knowledge used in the decision-making process is a static rule-set with parameters derived from observational research. Context-mediated behaviour provides the ability to react to multiple concurrent influences. Agents have limited means to communicate with each other in order to resolve contexts that require direct interaction. The location and movements of agents is displayed online. Colours and shapes can be used to visualise agent properties. PEDFLOW is implemented in Java and runs on any hardware that supports a Java virtual machine.
During the development of the PEDFLOW prototype, implementation decisions were made that resulted in the specific, sometimes unique, characteristics of the PEDFLOW agent entities (modelled pedestrians). The following list extends the agent model classification introduced in Chapter 2 by adding low-level implementation-specific categories. The option that applies to the PEDFLOW implementation is indicated.

**Agent activity (concurrent, sequential)**

This is concerned not so much with an implementation in a multi-processing/multi-processor environment, but with the question of agents that are active at the same model time who see exactly the same environment and base their decision on it. A mechanism needs to be in place to prevent collision between agents. Possible cases for concurrent observation/activity are: all agents, a fixed number of agents, a varying number of agents or only one agent at a time.

PEDFLOW: sequential implementation of quasi-concurrent behaviour

**Agent synchronisation (asynchronous, explicitly synchronised, implicitly synchronised)**

Synchronisation (access control of shared resources; mainly the location of other agents) is only an issue if a number of agents are active at the same model time, regardless of whether they see the same environment or not. It concerns the order of the update, as well as how to establish when the activation cycle for all of these agents has finished. Asynchronous means that agent activation cycles have no deterministic dependency on each other. In other words, is it possible for agent activation to overlap if they have started at a different model time. This is only feasible if there is no possible interference between agents, i.e. multiple agents can occupy the same grid element (not supported in PEDFLOW; see number of agents per grid element below) or rules prevent agents from occupying each other’s area. As soon as shared resources are available, agents will be synchronised either explicitly by a semaphore (or similar construct) or implicitly as a result of the program structure. In sequential activation, synchronisation is always implicit. An example for a synchronised model would be any turn-based game.

PEDFLOW: implicit as the result of the sequential implementation.
**Advancement (time, event/space)**

Advancement is concerned with what drives the model. In a purely time-advanced environment, all agents will be activated at the same model time (but quite possibly sequentially). In an event-advanced model, agent activity is triggered by an event, a change of state in the system. It is usually used for time-reduced simulations. A specific case of an event could be a change in location of one or more agents in the model. This is also referred to as a space-advanced model.

PEDFLOW: event/space-advanced. Agents in the current timeslot or agents interacting with each other are active.

**Time representation granularity (limited, indefinite)**

In all concerned models, but especially in those which are time-advanced, time is quantified to a certain extent. Most commonly there is a minimum time unit (base unit) and all other times are multiples of it with the result that all times can be expressed by an integer multiplier to the base unit. It is important to choose the base unit as a compromise between desired accuracy and computational impact, especially in a time-advanced model. Also there is the possibility that time is measured with a precision limited only by the precision of the floating point representation. This is useful for models that are not time-advanced but synchronised by other means.

PEDFLOW: limited to multiples of a given time unit.

**Observable agent state (interrogate internal, copied to grid)**

In agent-based, spatial models, agents observe other agents based on their position. This bi-directional look-up (where not only do agents know their own position in the shared data structure (grid), but the shared data structure knows the position of every agent) can be achieved by different means. The grid can contain a reference to the agents or it can hold a replica of the agent’s internal state (or the public subset thereof). The former has the disadvantage that it requires carefully designed access locking to prevent an inconsistent representation (especially in a parallel environment), while the latter has an additional replication overhead.
There is also the possibility of ‘simulating’ the reverse lookup by querying all agents about their position, but it is highly ineffective in most cases. It would only be useful if the number of agents was minimal and the number of grid positions was high, so that the cost for the storage space of the grid would outweigh the cost of the query.

**PEDFLOW:** agents know their position and query the grid for other agents’ positions if appropriate.

**Agent size, agent shape**

Depending on the application area, agents might be displayed in a non-square shape. While often shape and size are unimportant or can be represented as an attribute (internal state) for agent observation, they can sometimes play an important role with regards to collision detection if agents occupy more than one grid element (see below). In certain instances it might be possible to combine agents to form compound agents (i.e. mother with child in trolley) to represent different shapes and sizes. But if the number of agents becomes too large, it will not be feasible to do so and the model would require a way to (recursively) represent groups of agents by single agents (as possible in CYBELE [72]) which will collectively contribute to the action. The visual representation might not reflect the shape of the object the agent represents.

**PEDFLOW:** uniform, square representation of pedestrians for collision detection, but visualisation options for different shapes.

**Possible number of agents per grid element**

If one assumes that the agent size is a representation of real objects, it should be impossible for agents to overlap. Therefore, only one agent per grid is possible for agent sizes equal or greater than grid elements. For agents of sizes smaller than grid elements there is no way to guarantee non-overlap if they were to occupy the same grid element (unless there was a form of hierarchical grid structure). Hence the number of agents per grid element is restricted to a maximum of one. Also, without the restriction the framework would be overly complicated, as algorithms would be required to define (and validate) the order of agent-activation per grid element.

**PEDFLOW:** maximum of one agent per grid element.
Rule representation
Agent behaviour is determined by a rule system (or another mechanism that maps input values to output values such as neural nets). There are a number of possibilities for rule-representation, ranging from AI scripts that are interpreted through an Expert Systems class, through manually constructed decision trees to the simpler lookup tables that hold all possible input combinations together with their result values.

PEDFLOW: multiple rules are applied depending on the context the agent is in and the action determined by a weighted choice.

Observation of the virtual environment
Agents observe their environment by interfacing with the grid. Although they could enquire about the state (vacant/occupied by an agent) of every grid position by absolutely addressing it, the observation area is usually limited by the model. Also, some of the observations will be of type ‘find closest agent in area’, where area is usually defined relatively to the enquiring agent’s position. Such queries could be implemented on the agent side using multiple direct requests (return agent at position x, y), but work more naturally as search methods of the grid object.

PEDFLOW: area relative to agent in walking direction.

4.3 Obtaining input values for the PEDFLOW model
Observational methods are necessary in exploring the movement patterns of individual pedestrians. While macroscopic measures like speed and flow are readily available (e.g. [91; 92]), no such data is available regarding the microscopic behaviour. In particular this means investigating how people negotiate obstacles in their path and how they position themselves within their environment. It is important, however, not to rely solely on these techniques, as observational methods are unable, on their own, to explain the decision-making processes underpinning walking behaviour. Although the decision rules in the PEDFLOW model do not need to be identical to those underpinning human behaviour, some understanding of why pedestrians react to their environment in particular ways can only enhance the development of the behavioural rules in our model.
The behavioural research being undertaken as part of the PEDFLOW project combined interview and observational techniques to explore the relationship between pedestrians’ perceptions of the environment and their behaviour. In-depth interviews were carried out as part of the PEDFLOW project with volunteers to explore their decision-making processes in some detail, and large-scale questionnaire-based surveys employed to uncover generalities in pedestrian decision-making within various sections of the pedestrian population as a whole [69]. Although not directly part of the work of this thesis the results influenced the modelling of the decision-making process of the agents in the PEDFLOW model. This section concentrates on observing pedestrian movement within a range of different environments and in particular the methods that were devised to track the movement patterns of pedestrians in a fast and accurate way.

The observational studies are based on the premise that the optimal method of exploring how pedestrians negotiate urban space is through the use of unobtrusive, video-based surveys. Observation must be unobtrusive in order to avoid the well-known “observer effect” of individuals changing their behaviour when they know they are being watched [93]. To capture naturalistic movement patterns, one must ensure that pedestrians are unaware they are being observed. In order to address the obvious ethical issues associated with such methods, all studies were carried out in accordance with a strict Code of Conduct.

Observational studies can take a number of forms. The great power of the video survey over other methods lies in its ability to provide a permanent record of the environment under observation [94]. Extremely accurate measurements of both time and space can be derived from this record – providing, of course, that it is calibrated effectively with respect to the real-world measures of each. Two main drawbacks with this technique explain why no large-scale, systematic exploration of pedestrian movement at a microscopic level has been attempted to date. First, the problem of calibrating a foreshortened video image (which arises when the camera is not positioned at 90° laterally and horizontally to the ground) has restricted video surveys to locations where bulky scaffolds can be erected to hold the camera directly above the observation area [95]. Such
techniques restrict the range of environments that can be surveyed. Second, plotting the microscopic trajectory of pedestrians as they navigate urban space has so far proven laborious and time-consuming.

Recent developments in digital image processing provide the potential to overcome both these difficulties. A process was developed through which footage of pedestrian environments is collected, calibrated and analyzed and as a result a picture of how pedestrians behave at a microscopic level developed.

4.3.1 Filming, sampling and calibration

Once the survey location is selected, video footage of pedestrians is collected using either a standard hand-held digital camcorder mounted on a tripod or CCTV cameras operated by collaborators within local City Councils. Private survey companies are contracted to film areas that cannot be accessed using either of these methods. Between one and twelve hours footage are collected in any single session. Video recordings are converted into digital format using a standard video capture card housed in a PC.

![Figure 4.8 Calibrating the tracking software by matching real-world to image co-ordinates.](image)

For each of the empirical questions identified during the development of the PEDFLOW model, all instances of the appropriate scenario in the digital movie files are selected according to pre-defined, objective sampling criteria. These are then saved as short clips, typically a few seconds long, for subsequent analysis. For example: in comparing the deviation distance associated with an item of street furniture (such as a lamp post) with that for a cluster of stationary
pedestrians, all instances in which a pedestrian walks within a certain trajectory of the object (lamp post / cluster of pedestrians), and in which no other confounding variables enters the frame, would be clipped for inclusion in the sample. Each clip is then imported into a commercially-available motion analysis package. This software contains a method of calibrating images according to known values of x and y distances in the area under investigation, thus overcoming the problem of perspective foreshortening. The (chalk) markings in Figure 4.8 were created at filming time and their absolute position recorded. By selecting their pixel co-ordinates in the image, the software builds a calibration model that allows the approximation of real world co-ordinates for all pixel positions.

4.3.2 Tracking individual pedestrians

Image analysis software is used to plot the frame-by-frame position of selected objects (in this case, a pedestrian approaching an obstruction) for the duration of the clip. In order to provide an effective environmental context for pedestrian movement, the tracking function in the program is used to generate a "static make up" (SMU) file, which marks out the location of salient, static features of the environment (such as street furniture or pavement boundaries). The software outputs the real-world x and y positions of each tracked object separately as a function of time (t) in frames.

Figure 4.9 Semi-automatic tracking of pedestrians using image analysis software.
Unfortunately, this format does not allow the subsequent measurement of distances (and thus speeds); what is required is an x versus y plot, which essentially represents the pedestrian’s trajectory in real-world space. In order to rectify this, a custom software tool was developed that takes the output of the image analysis program, collects the SMU file associated with the clip, and generates a visualization of the x versus y trajectory of the pedestrian(s) under examination within its environmental context. In our original example, the deviation distance of pedestrians with respect to any object in its environment (either stationary, or moving) can be plotted with an accuracy as great as the calibration and the tracking processes allow – typically in the range of a few centimetres.

Figure 4.10 Custom software to visualise the trajectory of pedestrians and extract measures.

The screen capture in Figure 4.10 shows the user interface of the Analyse Tool including a short description of the available controls.
The workflow for extracting measures is as follows:

a) A set of output files from the image analysis program is imported.

b) The display can be adjusted such that the area of interest fills the centre of the screen. The measure of interest is named for identification in the result file.

c) If multiple pedestrians (or entities such as street furniture) are involved, they can be selected by their assigned colour. The selection is a snapshot of their current position and the associated parameters of time, x-position and y-position, is displayed in the text boxes below. The third row of text boxes displays the time difference, the distance between the two positions and quotient between distance and time difference. The latter can be interpreted as average time under certain conditions.

d) The whole set of values can be saved cumulatively in a text file, so whole series of measures can be performed efficiently.

e) In order to obtain meaningful measures, the controls at the bottom can be used to select a time in the scenario. They include a slider for course selection, single step buttons, a range limiter for narrowing down the time period of interest and a feature that will indicate inflection points in the curve by calculating an approximation of the 1\textsuperscript{st} and 2\textsuperscript{nd} derivative of the x/y curve. This is useful in reducing the human factor when determining angular separation, for instance while measuring the deviation distance.

Despite substantial computer support, the process of extracting measures is extremely time-consuming.

4.3.3 Informing the PEDFLOW model

By focussing on a single example of an event (in this case, a pedestrian approaching an obstruction), the technique outlined here allows a truly global analysis of movement behaviour in a given situation. Not only can individuals' movement behaviour be described – in terms of deviation distances, gap sizes, walking speeds, etc. – but factors that may affect the decision-making process (such as age, sex, or group size)[96] or the type of environment can be explored. In our study, any such factors that may be observed from the video footage are recorded in a spreadsheet alongside the descriptions of the
behaviour for each individual clip. This allows for the statistical analysis of whole data sets to uncover general differences in behaviour between certain sections of the population as a whole, or according to certain environmental features. The analysis was executed by a team of researchers and the result of the study can be found in [70].

For the purpose of informing the PEDFLOW model the analysis yielded three different types of outputs:

- raw data (speed, deviation distance, passing distance, passing position, entry position, exit position, group size) for the investigated scenarios
- normalised distributions of the raw data derived by clustering
- correlations of the above parameters to sub-populations with common attributes (e.g. gender, age group or size group) to be able to potentially compose new artificial populations with different percentages of constituents.

The research on this video was gathered in over 15 locations, but time constraints only allowed the exhaustive analysis for two sites:

- Petergate in York (narrow pavement width, sign post)
- Princes Street in Edinburgh (wide pavement width)

All agent parameters and the majority of the experiments in Chapter 5 were based on these data sets. However, due to the dynamic configuration the model is not limited to these configurations. When new data for different locations becomes available, it can be used by PEDFLOW after transformation into the XML format described in section 4.1.6.

4.4 User interface

4.4.1 The PEDFLOW Application window

The program presents itself as a typical windows application (Figure 4.11). There is a menu bar, complete with mnemonics, a tool bar for easy access to often used functions, the main display area (with optional scroll bars) and a status bar at the bottom. The larger part of the status bar displays error messages and information about the current or last action, and the small field in the far right contains the current model time.
Throughout the program one will be presented with tool tips for the main control elements to make the use of the program as easy and intuitive as possible.

![Figure 4.11 The application's main window.](image)

### 4.4.2 Input data files

PedflowApp needs two data files to run. One is `prefs.txt` and contains display settings and file locations (see also section 4.4.5). Its maintenance doesn’t require any user intervention; it is automatically created and saved. This requires a writable medium as the application will not run from CD-ROM.

The second, more important file contains all the information for the scenario (space layout, timing, etc). It is in XML format (see also section 4.1.6) and for convenience it has the extension `.xml`. PedflowApp will give the user access to edit the contents via its GridEditor, but in certain cases it might be desirable to edit it with another application or view it with a browser capable of displaying XML. When editing the file externally, one must adhere to the structure described in the file `pedflow.dtd`. The file access is via the file menu in PedflowApp, where one can find the usual New/Open/Save/SaveAs as in most Windows Applications (Figure 4.12).

![Figure 4.12 File menu (left) and directory structure (right).](image)
A number of additional data files are used by the program to store intermediate data and also results. Although PedflowApp will open/save files from any location, it is recommended but not required to keep to the following directory structure (relative to the current directory):

- DMP – for raw event data dump (used for replay)
- MAC – for macro files (automated batch execution)
- STS – for stats files (collected statistical values)
- STV – for stats values files (all gathered raw values)
- TEM – for entity templates (optional)
- XML – for xml data for scenarios (must also contain the provided pedflow.dtd)

### 4.4.3 Setting up a simulation scenario

When PedflowApp is started, it will try to re-open the last used simulation scenario, or (if this fails) start with an empty scenario. To start with a new scenario select **File/New**. See section 4.4.4 on how to edit an existing file.

Selecting **Edit/Grid** opens the grid editor. It is divided in three areas: the top-left deals with entity templates (stored, re-usable data files), the top-right with the grid dimensions and the bottom part with selected entities from the grid. Tool tips are provided for most elements.

![Grid Editor Interface](image)

Figure 4.13 Grid editor interface.

First the physical dimensions and their measurements need to be entered. The *space* field holds the size of a grid element in mm (typically 500mm), while *time* contains the smallest time unit in ms (typically 50ms). It is recommended that they are left at their default value, as changing the value has impact on the
quality of the modelling (\textit{space}) and the performance (\textit{time}). The size of the modelled area is then set as \textit{maxX} and \textit{maxY} in grid elements. Pressing \textit{Set size} will apply the new values, which should be reflected in the main PedflowApp window.

Now the area needs to be populated. This is done in two steps: create a template and add the template to grid. A template is an entity with all parameters set up as required - apart from location and size, which are filled in when the template is replicated (cloned) and added to the grid. This way complex templates (e.g. with distributed parameters) can be re-used for similar entities. To keep individual templates between sessions, the Save/Load buttons provide a way to store and retrieve entities with all their parameters.

![Blockage Editor](image)

Figure 4.14 Entity editor while editing a simple entity (blockage).

Pressing the \textit{Edit} button will open the entity editor. The window bar shows the entity type followed by the \textit{entity ID} (assigned by the program, -1 for templates) and a list of the actual parameter values. Although the available parameters differ for the various entity types (see 0 for explanation of their meaning), the following is valid for all of them. All parameters can be entered as single integer values, but some also as a distribution of integer values. Possibilities are normal distribution or even distribution (see appendix for complete list and syntax). The actual value is generated randomly from the distribution when the entity is added to the grid. The unit of measurement is given in brackets after the parameter name. If omitted, it is either a plain number or the parameter is measured in model units (as set in the grid editor). A range check is performed for single value parameters. If it fails, the offending value is marked red. Use the tool tip or the reference in the appendix to find out the valid range. The \textit{OK} button applies the value and closes the entity editor, while the \textit{Apply} button leaves it open. \textit{Cancel} will keep the existing distributions (not values) intact.
Once all parameters for the template are set, it can be added to the grid. Select a grid element in the main window (an orange cursor will highlight the selection) and press Add in the grid editor to generate an entity from the template and place it in position. By clicking and dragging the cursor, the area can be made bigger - co-ordinates and size are displayed in the status field. For singular entities, an individual object is created for every grid element inside the cursor, while for range entities all grid-elements are linked to a single entity. Entities (especially non-blocking ones) can usually overlap without a problem. The exception is that a stationary, blocking entity with a low ID number will stop other blocking entities with higher IDs to enter the model (even if they are mobile). This can lead to problems and should be avoided in the design (a typical mistake is to overlap parts of a building with a distribution of start co-ordinates).

By repeating the process with various entities, the whole grid can be populated with pedestrians, buildings and street furniture. If mistakes are made or corrections are needed for other reasons, see section 4.4.4 for ways to edit an existing scenario. Once everything is set up satisfactorily, the grid editor can be closed with the Exit button (entities with startDelay>0 will disappear from the display) and the scenario saved with File/Save.

4.4.4 Editing an existing simulation scenario

Editing an existing scenario is similar to setting up a new scenario. Once a scenario is opened and in stop mode, it can be edited. However, it is recommended that the simulation is reset before starting the grid editor in order to avoid accidentally editing an intermediate scenario. Another potential problem is in shrinking an existing scenario. Entities outside the new boundaries will be lost, possibly with follow-on effects. Therefore one needs to check and transfer existing entities (including x or y distribution!) or limit resizing to growth.

<table>
<thead>
<tr>
<th>Type</th>
<th>ID</th>
<th>Start X</th>
<th>Start Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>person</td>
<td>0</td>
<td>13</td>
<td>13</td>
</tr>
<tr>
<td>blockage</td>
<td>1</td>
<td>12</td>
<td>14</td>
</tr>
<tr>
<td>blockage</td>
<td>3</td>
<td>15</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 4.15 Entity selector of the grid editor.
The editor for a blocking, singular entity can be accessed by simply right-clicking on the grid element. For an entity of another type it is necessary to mark the area containing it with the cursor. All entities within the orange rectangle are displayed on the list in the lower part of the grid editor. The required entity can then be selected with the mouse and edited by pressing *Edit*. Entities can also be deleted by pressing *Delete*. The *Save* button is provided to save the selected entity as a template, which is convenient when wanting to reuse an entity from another scenario.

It is recommended that the resulting file is saved from the file menu. *SaveAs* is required if you want to keep the original file.

**4.4.5 Preferences**

Selecting *Edit/Preferences* from the menu bar will open the preference editor. The *Display* tab enables the user to modify the appearance of the model, the *Visualisation* tab gives options for the display of additional info in, the *Tracking* tab allows modifying what is displayed for entities that have their verbose flag set and the *Stats* tab deals with the gathering and extraction of statistical data from the model.

![Preference window](image)

Figure 4.16 Preference window.

The *Display* options are straight forward: the *Zoom* slider controls the display size of individual grid elements, the *display grid* options enables the display of the grid and the combo-box selects the geometric shape for the agent display. Options are *Square* (fast), *Triangle* (for direction display), *Circle* (aesthetic) or *with ID* (very slow; mainly used for debugging).

*Visualisation* has only one option at the moment: the display of hotspots. If enabled it will indicate the number of times a person has stepped on a particular grid element. The more people, the darker the field.
Figure 4.17 shows a simulation using the data from the High Petergate scenario. The black square on the right hand side is an obstruction entity modelling the lamppost. The dark areas on the side and the lighter areas below and above give an impression on how the agents have avoided the obstruction. This mechanism can be used to identify bottlenecks in the scenario layout.

![Image](image.png)

Figure 4.17 Visualisation of 'hotspots'.

Tracking enables the user to select what kind of information is displayed for agents who have their *verbose* flag set (either manually or triggered by condition). Options are *rubber band* (a line connecting the entity to its goal), *path* (markings on all grid elements the agent has stepped on), *decision* (text comments in the log on the decision making process of the entity) and *observation* (display of the observation area). Figure 4.18 shows an agent with the *verbose* flag set and all tracking features activated.

![Image](image.png)

Figure 4.18 Tracking features in PEDFLOW: path, rubber band and observation area.

### 4.4.6 Running a simulation

All functionality for actually running a simulation can be accessed via the *Simulation* menu or (more conveniently) from the toolbar.
The functions are:

**Step** - Run simulation until at least one active entity is activated; activate all other entities in the current slot; redraw display. This is not a constant time step, although for a densely populated area it most likely will be (there is at least one entity per timeslot).

**Run** - Run simulation continuously in real-time. More precisely, repeat the following cycle until **Stop** is pressed: advance time-slot, activate all entities in time-slot, request display update if timeslot was not empty, pause. Note the timer display in the bottom right corner.

**Stop** - End run cycles.

**Reset** - Reload scenario from disk.

**RunTo** - Opens a new dialog box that allows to input a time until which the simulation will progress. The time is absolute and not relative to the current time. The **Run** button will execute the simulation and **Stop** will abort prematurely. If **no display** is ticked, the simulation will run as fast as the computer allows.

PedflowApp supports a simple macro recording facility to automate repeated runs of experiments. The resulting files are also compatible with pedflowCMD, the command line version of the program. The menu items should be self-explanatory.
The macros which support the execution of the following actions are:

<table>
<thead>
<tr>
<th>Macro command</th>
<th>Corresponding menu options</th>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open</td>
<td>File/Open</td>
<td>Filename</td>
</tr>
<tr>
<td>Save</td>
<td>File/Save</td>
<td>Filename</td>
</tr>
<tr>
<td>dumpMOS</td>
<td>File/DumpMOS</td>
<td></td>
</tr>
<tr>
<td>Reset</td>
<td>Simulation/Reset</td>
<td></td>
</tr>
<tr>
<td>runTo</td>
<td>Simulation/RunTo</td>
<td></td>
</tr>
<tr>
<td>Step</td>
<td>Simulation/Step</td>
<td></td>
</tr>
<tr>
<td>statsFreqSave</td>
<td>Prefs/Stats/FreqSave</td>
<td>Filename</td>
</tr>
<tr>
<td>statsSave</td>
<td>Prefs/Stats/Save</td>
<td>Filename</td>
</tr>
<tr>
<td>statsValuesSave</td>
<td>Prefs/Stats/ValuesSave</td>
<td>Filename</td>
</tr>
<tr>
<td>statsReset</td>
<td>Prefs/Stats/Reset</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.3 Macro commands and corresponding menu options.

The command sequences are stored in macro files (*.mac) and contain one keyword per line. Required parameters are listed immediately behind.

4.4.7 User interface and usability

The primary focus of the PEDFLOW project is the modelling engine. As such there has been no formal evaluation of the user interface of the prototype implementation. However, the application and user interface has been demonstrated in the regular focus group meetings with representatives from the potential user community (City of Edinburgh Council) and producers of pedestrian models [97; 98] (Halcrow Fox), and feedback has been taken on board. The usability of the program has been informally tested, when during the course of an open day session a group of interested pupils, after a short introduction, were asked to set up a simple scenario and perform experiments with different pedestrian model parameters. All participants were able to do this and showed enthusiasm and creativity in experimenting with the software.
5 Evaluating the model’s capabilities

This chapter is concerned with the model’s validity, extensibility, usefulness and performance. Validation of the model is achieved by comparing the output of the model against data from the real world that has not been used during the calibration of the model. Visualisation is used to subjectively estimate how realistic the behaviour is. Typical macroscopic configurations (queues, clusters) are investigated that also can be found in reality. Some of these behaviours rely on the addition of modelled (virtual) entities that have no equivalent in the real world. The chapter reports about the success in extending the model to include them. The model not only provides visual output, but also numerical. Such measures are for example exit-distribution (temporal and local) or pedestrian density. A method is outlined on how such measures can be used to reason about the design of pedestrian areas. At last the performance limits of the model are investigated. There are two approaches: One is to generate synthetic worst case (with regards to the model implementation) scenarios to investigate the computational limits. These limits depend on the hardware used and need to be understood in order to establish the boundaries in which simulation experiments can be performed. The second approach is to take a real scenario and increase the number of pedestrians until saturation occurs (more enter than leave, eventually resulting in gridlock). This saturation point describes the limit of the model, as no useful results can be achieved under these circumstances.

5.1 Investigation of macroscopic, emergent behaviour

5.1.1 Platooning and lane movements in bi-directional flow

Macroscopic phenomenon

The distribution of moving pedestrians is not uniform; as they move they are grouped in clusters or lines [95; 99]. While this can be the result of external causes such as traffic lights (see also section 5.1.3) or people exiting buildings in groups, the passing (or not) of slower walkers also plays an important role. People will often not overtake slower walkers if there is no room. Even on wide pavements, where there is enough room, pedestrians might not pass out of choice and select to slow down and follow the group ahead [100]. Having passed a slower-moving pedestrian, people will often continue to move with the new trajectory rather than fall back into their original line. Such decisions lead to
platooning which can be easily observed in the real world. Figure 5.1 shows an example from video footage taken at Edinburgh’s Princes Street.

![Figure 5.1 Example of clustered movement of pedestrians (platooning).](image)

The fact that the clusters can be easily detected visually, makes this particular movement pattern a good indicator for the validity of certain aspects of pedestrian models and is often used in publications, for example [101] and [11].

**Hypothesis**

In the PEDFLOW model there are no explicit rules to generate such a formation. So if platooning can be observed as an emergent behaviour, it will be an indication of the validity of the model.

**Experiments**

In order to generate output comparable to real world equivalents, the layout is modelled across the areas where measurements have been taken (see section 4.3.3). Walking areas of different width were investigated to compare the effect of width:

- narrow pavement width (Petergate in York)
- wide pavement width (Princes Street in Edinburgh)

The most obvious choice for the source of agents’ parameter distributions would be to use the data sets directly derived from the observations in the respective areas. However, the measurements were done with low pedestrian density in order to isolate the pedestrian from unwanted influences. During experimentation the density was artificially increased while maintaining the (relative) distribution of the agent parameters. This was achieved by increasing
the number of people in the scenario while maintaining the distribution of their parameters including entry position.

The display size was chosen for good visibility and the area of visibility limited to the centre of the modelled area in order to avoid edge effects. When the agents enter the simulation their position is pre-determined. For emergent behaviour to manifest itself, agents need to actually have navigated the simulated area.

The simulation was run for a variety of pavement widths with the datasets and the appearance of platoons recorded with regards to their length and persistence.

**Results, interpretation and conclusion**

Due to the dynamic nature, it is impossible to obtain objective measures for what essentially is a volatile phenomenon. For example, fast agents will often break the formation temporarily and the group will later reform in different lanes.

Figure 5.2 Emergent platooning in PEDFLOW.

Nevertheless, platooning could be easily seen (Figure 5.2) and the following observations were made with regards to varying data set, pavement width and agent density.

The different datasets didn’t result in different agent behaviour. Considering the similarity in speed distribution, this is plausible and expected.

There is very little visible dependency on the pavement width, though there is a slight tendency for relatively more platoons to form on wider pavements. Occasionally agents cluster together. Clusters form when local pedestrian density is high, and this is more likely to happen at greater pavement width. On the dispersion of such clusters, platoons are formed.
Agent density has the greatest effect on the forming of platoons. With low density, no platooning is observed, as agents would continue in their original direction towards their goal, only moving sideways avoiding oncoming or slow pedestrians. This is consistent with observation of platooning in the real world.

With increasing density, the likelihood that multiple agents end up walking close together in the same direction increases. Avoiding oncoming agents, they will keep changing lane until a balance is achieved where both directions can move in their independent lanes.

Further increasing the density will lead to saturation (Figure 5.3). Agents will face new obstacles immediately after they have moved and will have to move again so that the balance is constantly being broken. Local short platoons can still be seen, but they disperse quickly.

![Figure 5.3 Local, short platooning in high agent density.](image)

Overall the appearance of platoons in the model is similar to what can be observed in the real world. The model shows its limits at high agent densities. This however is not a drawback, as the aim is to create layouts that actually reduce high pedestrian densities.

### 5.1.2 Moving in pairs

**Macroscopic phenomenon**

One discovery made during the analytical work in developing PEDFLOW was the fact that people very often move in pairs or groups, and that this as a big
influence on their behaviour [70]. Figure 5.4 is an image, cropped and enlarged, taken from video footage shot in York. It shows a couple moving towards a lamp post – there are several options for how they will behave in order to avoid a collision. Experience suggests that regardless of how they circumvent the obstacle, the couple will soon be reunited and continue their walk together side by side.

![Figure 5.4 Couple walking towards a lamp post.](image)

**Hypothesis**

PEDFLOW implements a special context to support the scenario as described in section 3.2.3. To validate the performance of the mechanism in circumstances where multiple contexts apply, pairs of agents were sent on routes where they had to avoid obstructions. It was expected that they will join up again afterwards.

**Experiments**

Two types of experiments were performed. In a controlled environment, agent pairs were released on one side of the model with a goal on the other side. The direct route contained obstructions in the form of blockages (static obstructions). The route of the agents was recorded. In a second experiment agent pairs were injected in the datasets used in section 5.1.1 but marked with a different colour. When running the experiment, the navigation of the pairs across the area could be observed while they tried to avoid oncoming agents (moving obstructions).
Results, interpretation and conclusion

An illustration of the result of the first experiment is given in Figure 5.5. Even without the time component available in the animation, it is obvious that the agents’ behaviour is aimed at staying with the partner while still pursuing their goal of moving on.

![Figure 5.5 Agent pairs rejoining after having circumvented two obstacles.](image)

When trying to explain the behaviour, one must keep in mind that other contexts have an influence on the agents’ behaviour and the choice of action is made randomly. For example, an obstruction will force them to deviate from their path. The agents will move independently and the relative distance has hence to be re-established for every activation. In all six cases the two agents have the same parameters and want to move from the left to a goal on the right. Obstructions in their way force them to split up. Depending on the weighted random selection different routes emerge. In all cases they will walk close together again eventually, although the time until they achieve it and the route they take will differ.

In the second experiment the agents also behaved as expected and tried to stay together. As a result they created an obstacle to oncoming agents which was harder to avoid than a single person. Such a phenomenon is also observable in the real world, where couples of pedestrians tend to be the slow and inflexible party in an encounter with single pedestrians who are consequently held up in their attempt at walking past.

5.1.3 Pulsation on Pedestrian Crossings

Macroscopic phenomenon

A feature in the urban environment which has been under scrutiny [102] for its effects on pedestrian flow is the signalised pedestrian crossing. The primary effect is the red light, which will cause affected people to stop at the edge of the kerb. The secondary effect is that people behind the first line will start to queue;
in other words they will not try to avoid them as they would ordinary obstructions but instead move in close behind, possibly filling gaps as the number of waiting people increases. Furthermore, people walking in a different direction who are not affected by the light are now faced with a crowd of people blocking their path.

When the light switches to green the crowed will start to cross. If there are crowds on both sides of the street, there will be a confrontation as people want to get past. Once the two crowds have passed each other, the high density crowd will disperse, due to the different walking speeds. For an uninvolved observer watching the lights change for a period of time, the visual result is a pulsating effect [103] of people accumulating, milling to get past and dispersing into a lower density.

**Hypothesis**

While the context “pedestrian crossing” with associated triggers and actions determines only the behaviour of individual agents, the macroscopic behaviour described above can be replicated in the PEDFLOW model.

**Experiments**

A layout was used that was loosely based on measures and observations from a road junction of Princes Street, Edinburgh. It comprises of a wide pavement with a building edge and directional set of pedestrian lights at the street corner as shown in Figure 5.6

![Figure 5.6 PEDFLOW agents at a modelled pedestrian crossing.](image)

An agent moving from left to right is affected by a red light and will not be able to make movements that would take it out of the associated area for that particular light (black border). Agent C will therefore continue to move towards
the road while still being subjected to other contexts (e.g. the avoidance of B). Once it has reached the road it has to stop. If other agents have already lined up on the kerb, the agent will also be subjected to the “crowd” context to allow it to get close to other pedestrians. Being confined to the context area also means that it will not be possible for the agent to swerve onto the road to avoid the queue. If the light changes to green all restrictions on the agents are lifted and they will move according to other applicable contexts.

In the experiment the pedestrian data sets from Princes Street were used together with a set of red/green timing measures gained from the traffic lights in Princes Street.

**Results, interpretation and conclusion**

Running the experiment with the original data immediately yielded the expected result of build-up and relief. By varying the timings of the lights and/or the density of the pedestrians, a number of realistic macroscopic behaviours could be simulated ranging from little impact on the pedestrian flow for low density and short time periods to massive build-ups which were unable to disperse for long red periods and high pedestrian density.

![Figure 5.7 Pedestrian crossing in PEDFLOW: a) build-up, b) crossing, c) dispersion](image-url)
The dynamic nature of the phenomenon makes it difficult to capture the different phases, but an attempt at illustration is made in Figure 5.7. In image a) the light has been red for some time and groups of modelled pedestrians are building up on both sides of the road. Once the light as changed to green in image b), agents in the simulation start crossing and have to avoid the oncoming agents. Note how the used area is wider than the one occupied initially. In image c) the agents are now able to move at their desired speed again and when they encounter oncoming pedestrians, lane formation can be observed.

From the experiments undertaken it can be concluded that PEDFLOW is a useful tool when trying to determine a safe timing for pedestrian crossings. It can be determined by finding the amount of people that can cross without noticeable delay and without having to resort to diverting onto the road and combining that knowledge with the expected pedestrian density. If there are multiple crossings along a road, their dependency can also be taken into account and adjusted to achieve the best throughput with a minimum of waiting time. The fact that these timings also affect vehicular traffic where they might be inappropriate is a different issue which is not within the scope of the model.

5.1.4 Dynamic assembly around attractions

Macroscopic phenomenon

A commonly found feature in city centres and other tourist areas are street artists and the associated audience of pedestrians who have interrupted their journey to gather around the artist and watch for a while before they continue their journey. An example that has been cropped and enlarged from video footage in Edinburgh’s Princes Street can be seen in Figure 5.8.

“Attractors” are features of the environment that attract the attention of a pedestrian and distract it from its current goal (e.g. street musicians or shop displays). Attracted people will move towards the attractor and pause for a certain time before continuing to move towards the original goal, while others will ignore it. Unlike traffic lights, attractors do not affect all passing pedestrians in the same way.
Figure 5.8 Artist with audience (C), attracted people (B) and non-interested passer-by (A).

Hypothesis
To detect the context of attraction, a similar approach to pedestrian crossings is used. If the distance to an attractor is smaller than the attraction range of the attractor and the attractiveness of it higher than the “attraction potential” (likelihood of being attracted) of the agent, the context of attraction is entered. The agent will stop moving if the distance to the attractor is equal to or closer than the attractor’s attraction distance. This is similar to the “walking with a partner” context. Once this limit is reached, the agent will leave the context of attraction and pursue the original goal again. By associating a delay with this removal, the time spent by the agent in close vicinity to the attractor is represented.

Experiments
In order to investigate the behaviour in the attractor context, an attractor entity was placed near the edge of a simulated pavement with medium bi-directional flow, similar to the site where the data has been collected. The density of the pedestrian population is varied as are the attractor’s parameters.
Results, interpretation and conclusion

For sparse to medium density the model works as expected. A half-circle of pedestrian agents is formed as shown in Figure 5.9. A single agent in the top left has not been attracted and continues its journey undisturbed. What can’t be seen in the static picture is how some agents are settled in their position while others appear to be trying to get closer to the attractor until they find a position. This realistic effect is the result of the random factor in the weighted random choice when evaluating the effect of the different contexts (attracted, but way is blocked). Eventually agents can be seen leaving, making space for new arrivals.

![Figure 5.9 Simulation of an attractor in PEDFLOW.](image)

When increasing the density of the flow, the same saturation effect as already observed in section 5.1.1 occurs. The model will degrade into a deadlock situation where more agents enter the simulation than leave. While undesirable, this situation can also occur in real life when the crowd around a street artist completely blocks the pavement.

Tests with varied parameters for the attractor entity showed the expected results, such as the radius around the attractor changing proportionally and agents attracted from further away. In the experiment, synthetic values were used for the parameters, although they are in fact measurable as shown in [104] and would lead to a more correct model.
5.2 Extension of the model by external parties

5.2.1 Pedestrian behaviour at bus stops

As part of the evaluation of the PEDFLOW model, a student project [105] was undertaken that investigated how easy it was for the PEDFLOW model to integrate additional contexts, and how well it would perform with a modification that was not in the original design.

Macroscopic phenomenon

The pedestrian behaviour at (British) bus stops (see Figure 5.10) is usually well organised (it is assumed the bus stop only serves one route). People arriving in sequence will form a queue parallel to the road, often under the cover of a bus shelter. Rarely will people barge in at the front or otherwise reshuffle their position.

![Figure 5.10 Orderly queue at a British bus stop.](image)

Hypothesis

This predictable behaviour makes it easy to implement the additional context-mechanism, and also to verify the validity of the model output. For a correctly working model, agents would navigate past other agents to the end of the queue and stay stationary. Triggered by the "bus arrival" event, the agent at the head of the queue would disappear, the other agents would take its place and disappear one by one.
Experiments

A bus stop entity and an associated context were created. In the model the bus stop context is triggered if an agent’s sub goal is the bus stop entity. Agents move, other contexts permitting, towards the end of the bus stop and proceed as far forward as other agents allow.

Figure 5.11 PEDFLOW application with a bus stop simulation.

The student experimented with the agent parameters until a realistic behaviour was achieved. This approach was necessary as no measured, observational data was available for this scenario.

Results, interpretation and conclusion

In an iterative process, the student was able create a simulation that mimicked real-world behaviour. This confirms that the model is easily extendable to new scenarios. The lack of authentic input data makes it somewhat difficult to achieve realistic behaviour.

5.2.2 Queue formation at automated teller machines

An MSc project [106] continued the research from [105] to look at other forms at queues and how they can be recreated using the PEDFLOW model. The results of the research were also published in [107].

Macroscopic Phenomenon

Queues found at automated teller machines (ATMs) differ significantly from those at bus stops. There is no feature like a bus shelter to enforce the direction of queuing, so people are often found milling around the ATM while maintaining a certain distance to protect the privacy of the person operating the ATM.
(Figure 5.12a). Occasionally a feature of the environment will influence how a queue is formed (Figure 5.12b), but if there is enough space straight queues can be observed as well (Figure 5.12c). In all cases the safety distance is maintained.

Figure 5.12 Queuing behaviour at an ATM: a) milling b) environmental and c) straight [106].

Hypothesis

Using the context-mediated behaviour mechanism in the PEDFLOW model, it is possible to create the different type of queues as emergent behaviour in a scenario where pedestrians walk along a long pavement with ATMs.

Experiments

An incremental approach was used in implementing the different phases of the queuing process:

- attraction (triggered randomly if passing an ATM entity)
- approach (moving towards the ATM avoiding collisions)
- waiting (waiting in queue and advancing if possible)
- serving (pausing in front of the ATM)
- leaving (continue moving towards the original goal)
Each is implemented as a different context and the transition is triggered in interaction between other agents or by the elapsing of time (e.g. serving). It was possible to extend (subclass) existing classes, (e.g. the generic attractor for the attraction phase) and extend them with specific behaviour relevant for the ATM context. The problems in creating a dead-lock free implementation are explained and solutions described.

Results, interpretation and conclusion

The impact of different implementations and varying parameters on the shape of the queue were investigated, and this led to the following conclusion (see [106], page 46): “In summary queue formation was witnessed and various queue types were observed. Superficially the queues were an accurate representation of the real thing. Fieldwork confirmed not only the shape but the queue dynamics was also present.”

The successful modelling of compound/multi-part contexts further confirms the suitability of the PEDFLOW model for the simulation of pedestrian scenarios with complex interactions.

5.3 Microscopic measures of service

Introduction and explanation

One of the aims of the PEDFLOW project is the definition of measures that describe the ‘quality’ or ‘walkability’ of a certain pedestrian area and the extraction of values for these measures during model runs. This will enable the town planner to compare options for new designs based on objective criteria rather than subjective estimates. As a result, designs could be optimised with regard to a certain quality measure, although of course the usual layout restrictions (legal etc.) still apply.

With the existing model implementation there is the unique ability to measure a large number of variables. Some of these values can be easily measured in real life (e.g. number of pedestrians, time) but others are more difficult to obtain by empirical research (e.g. changes in speed) or don’t even have an equivalent in the real world (e.g. number of pauses, number of deviations from ideal path).
Examples of such measures are:

- time spent in an area
- enter and exit position of agents
- distance covered by agents
- occupancy of area
- number of pauses because of blockage
- number of deviations from ideal path
- number of swaps (agent position change)
- number of actions based on particular context (e.g. couple)
- number of pedestrians entering/exiting per time unit

The problem with such raw data is that it is not an intuitive expression of quality as they do not translate directly into established levels of service (LOS [65]) and it does not allow comparisons between scenarios. It is, however, possible to combine measures to achieve a better effect. For example:

- covered distance relative to shortest distance
- delay in relation to ideal time
- occupancy per area as a density measure

If certain microscopic variables are to give an indication of the ‘walkability’ of the area then their values need to be extracted, transformed (normalised) and presented in a usable form (e.g. a graph). By normalising the measure (with regards to the area, the number of pedestrians and/or the optimum value) a set of outputs can be obtained which can be used to compare different scenarios. It is important to choose the variables that are best suited for the purpose.

Below are some examples of measures, their transformation and an indication of what they express with regards to the usability of pedestrian space. For a given scenario the following variables can be measured:
Table 5.1 Possible interpretation of microscopic measures.

<table>
<thead>
<tr>
<th>To measure:</th>
<th>Normalise against:</th>
<th>Indication of:</th>
</tr>
</thead>
<tbody>
<tr>
<td>walking speed</td>
<td>desired walking speed →</td>
<td>unwanted delays</td>
</tr>
<tr>
<td>covered distance</td>
<td>shortest distance →</td>
<td>unwanted detours</td>
</tr>
<tr>
<td>travel time</td>
<td>ideal travel time → (walking at desired speed)</td>
<td>combination of delays and detours</td>
</tr>
<tr>
<td>number of pauses/swaps</td>
<td>distance or time →</td>
<td>unwanted manoeuvres</td>
</tr>
<tr>
<td>(model-internal)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>distance covered at desired speed</td>
<td>covered distance →</td>
<td>unwanted decelerations</td>
</tr>
</tbody>
</table>

The measures could be represented as an average (e.g. an average of n% delay over optimum time), as a distribution (e.g. delays grouped in clusters and the associated frequency of occurrence) or as a percentage over or under a critical value (special case of the previous occurrence with two clusters). The advantage of the latter is that it only consists of a single number; however, the critical value needs to be carefully chosen by the experienced town planner.

By running the model with varying numbers of pedestrians, the above measures can be plotted against the macroscopic measures density and flow rate as a way to show trends and dependencies (e.g. increase in delay with increasing density).

The following is a demonstration of the potential of MOS given at a PEDFLOW steering group meeting to a group of town planners from York and Edinburgh council and representatives from the consultancy Halcrow Fox.

**Example walk-through**

The location chosen for this example was the pedestrianised area High Petergate in York for which sufficient data and information was gathered and analysed as part of the process described in section 4.3. The (fictitious) task was to investigate the impact of increasing the number of lamp posts in the area on the quality of service for pedestrians and more specifically the relative delay people will experience. Consequently the chosen independent variables are:

- number of obstructions
- pedestrian density

The model was set up with the location and agent data. In addition the microscopic measures which would be collected during the model run were specified. Figure 5.13 shows how an MOS entity is created in the grid editor.
The measure of interest is the time it takes the agents to cross the area, or more precisely, the delay they experience, as it has a great influence on the quality or ‘walkability’ of the area [108].

![Figure 5.13 Setting up the model for the collection of microscopic measures.](image)

The model was then run repeatedly with a varying density of pedestrians and number of lamp posts and the data collected is transferred to a spreadsheet. The raw data was transformed as follows to derive the measure or relative delay as follows:

- density = pedestrians/area
- minimum time = shortest distance/speed
- absolute delay = time - minimum time
- relative delay = absolute delay/minimum time

Finally the continuous values were grouped in bins of suitable size, i.e. units of 5 people per 100m^2 for the density and 5% for relative delay and graphs plotted to visualise the dependency. Figure 5.14 shows the result. It can be seen that for a given density of people the experienced average delay increases with the number of obstacles and that for a given number of obstacles the delay increases with the increasing density. In both cases the likelihood increases that
an agent will encounter another entity and will have to move past (detour) which
will result in a longer route which will (at constant speed) result in an increased
relative delay (as a measure for comparison against the ideal direct route). At
zero density (i.e. no additional agents in the field) and no obstructions, there is
relative delay as the route taken is the direct (i.e. optimal route). The
dependency is approximately linear with a slight decline at higher densities.
There is a slight dip at the curve for a single obstruction at a density of 35
p/m^2, which is considered to be insignificant and an artefact of the modelling.

Another presentation of the model output is the plot presented in Figure 5.15.
Here the distribution of different delay values is visualised for different agent
densities (unlike Figure 5.14 which only represented average values). This is
the combined result for all experiments/number of obstacles. Put differently, it
shows what percentage of the population experienced which relative delay
under different density conditions. While a large number of pedestrians
experience a small delay, only a few are delayed by more then 25% of there
journey time. Extreme delays are very rare, even with larger densities. There is
an unexpected inconsistency in the curve for high pedestrian density, which is
still within the trend of declining delays and considered an insignificant artefact
of the modelling process.

Conclusion
The audience agreed that there was good potential in the ability of the program
to extract microscopic measures from model runs and to transform them into
quantitative measures of service. However, no experience exists in how to
interpret the results, nor what measures would give good and meaningful
results. Further research is required as part of a new research project, working
in close co-operation with the target audience.
Figure 5.14 Average relative delay over density for different numbers of obstructions.

Figure 5.15 Distribution of relative delay for different agent densities.
5.4 Speed-Density diagram

The dependency of pedestrian speed from the density of the walking area, is one of the most fundamental criteria used to evaluate the quality of walkways and was already used by Fruin in 1971 [65]. It is sometimes described as the “fundamental diagram” [109] and has a scientific basis with results published in many places [110]. As such it is an objective way for evaluating a pedestrian model and is used in such capacity for PEDFLOW.

Hypothesis

The speed/density distribution obtained from the model should match the distributions in the literature, decreasing steadily with increasing density. Limitations of the PEDFLOW model do not allow the modelling of high densities (over 3 people/m^2), so results above that limit will be atypical.

Experiment setup

Speed/density data is collected along traffic links. For the experiment setup the High Petergate (York) scenario was used, a pedestrianised area for which data had been gathered during the data collection described in section 4.3. The width of the modelled area is 8 meters and the length 15 meters (based on the investigated area, the actual road is longer). Agents enter the area from both directions (counter flow). The distribution of desired walking speed (free speed) from the empirical research in [70] was used (average speed 1.4m/s). Data was collected using the mechanism described in section 5.3 to store the time between entering and leaving the simulation area for every agent. The speed was derived by dividing the length of the walkway by the time required to cross it. Density was recorded at the point when agents entered the simulation. Different densities were created by progressively introducing more agents into the simulation.

Result, interpretation, conclusion

The raw time and density values from the experiment were collected into a spreadsheet, transformed to commonly used units and plotted in the diagram shown in Figure 5.16. As expected, the effective speed decreases with increasing pedestrian density. Noticeable is the sharp initial drop in speed from unobstructed movement to very slightly obstructed (0.5 person/m^2). An
explanation is that the agent parameters were derived from data collected under very low density conditions [111]. People would start the avoidance process earlier than necessary at their own convenience. However, in a more highly populated area that behaviour is potentially counterproductive, as it will bring the (modelled) person in the path of another obstruction, causing additional delays.

Figure 5.16 Speed/density diagram for High Petergate simulation.

Comparing the results with diagrams from other authors using real data, there are some commonalities and some differences, but in general the diagrams are similar.

Figure 5.17 Speed/density diagrams in the literature [109].
All curves have in common that they start with a free speed of about 1.4 m/s (the exception is Virkler [110] with a low starting speed of 1 m/s). At a density of 3.5 person/m², the average speed has dropped to about 0.3 m/s and speeds beyond that density differ widely. The PEDFLOW curve has an initial steep drop that is not mirrored by any of the other diagrams where the dependency approaches linearity (with the exception of Fruin) and it only flattens out at a density of 1 person/m². A possible explanation for this phenomenon is the fact that real pedestrians will adapt their avoidance strategy and are more flexible in low-density situations while the model uses the same avoidance mechanism regardless of closeness to other agents, unless additional contexts apply.

However, taking into account the diversity of the existing speed/density curves, which is the result of the different circumstances under which the data was obtained, the outcome from the experiment still indicates that the emergent speed/density dependencies in PEDFLOW are sufficiently realistic.

5.5 Performance and scalability

5.5.1 Relevance

Despite the continuous increase in computing performance [112], computing tasks still require a finite time to execute. If this time becomes too long, it will tax the patience of the user or it will even mean that the result becomes useless.

Performance measuring or benchmarking of software products is a controversial topic. One reason for that is the fact that it can be abused for marketing purposes by intentionally only evaluating the strong points of a product. But the more important reason is that performance is not very well defined, even though many attempts have been made (Lilja [113]) to develop a universal and generally acceptable definition that would allow objective comparisons. Typically one or more critical measures (such as overall execution time, operations per time unit) are defined and measured under varied conditions.

Performance also depends on factors that are external to the program (e.g. access to computing resources), especially in a multi-tasking environment where such resources are shared with other processes. The number of interrupts and other concurrent threads directly impacts the completion time for
tasks in the main program. Since they will be present during the normal use of
the program as well, they must not be ignored.

In PEDFLOW, the critical measure is implicitly defined in the requirements; it
must be modelled and visualised in real time. If this is not fulfilled, the model is
less useful or even useless. The requirement of “real-time” or “faster than real-
time” [114] means that the execution time for a modelled scenario must not
exceed the time the same situation would have taken in the real world. This
must be achieved on a desktop computer. For the purpose of this thesis, the
following specification was used: an Athlon PC with a 1.33GHz Thunderbird
processor, ATI 128Pro video card, 256Mb RAM and 40Mb hard disk.

An often quoted paragraph ([115]) defines scalability as “the ability of a
computer application or product (hardware or software) to continue to function
well as it (or its context) is changed in size or volume in order to meet a user
need. Typically, the rescaling is to a larger size or volume”. In other words, an
application is scalable if any parameter that influences performance can be
changed and the program will continue working according to specification. This
parameter change is not unlimited. The limits of scalability give a good
indication about the usefulness of the program.

Scalability cannot be measured as such. Instead, the dependency of a
performance measure on a parameter is investigated and an approximated
function used to predict the limits. For simple algorithms it might be possible to
deduce the dependency from the algorithm (e.g. linear or exponential), but it is
usually determined empirically.

Obvious candidates for variable parameters in PEDFLOW that have an impact
on performance are the absolute number of modelled active entities, the
number of entities per area (density) and the absolute size of the modelled
area. However, there are also less obvious variables such as movement speed
and number of complex interactions. Some of those parameters are dependent
on each other. Therefore it is important to identify critical dependencies.
5.5.2 Run-time environment (external factors)

Although the hardware used has an obvious impact on performance, it will not be investigated here. The requirements state “a typical” desktop machine on which the experiments will be conducted. The reference machine used for all experiments was an Athlon PC with a 1.33GHz Thunderbird processor, ATI 128Pro video card, 256Mb RAM and 40Mb hard disk. The amount of memory is more than sufficient, a fact that has been confirmed regularly by querying the task manager. The machine is running Window 98SE as operating system. The number of concurrently running processes has been kept at a minimum for the majority of the experiments, with no other applications running. Control experiments have given an idea of the impact that the execution of other concurrent applications have on the performance of PEDFLOW.

Since Java is an interpreted language, the implementation of the Java Runtime Environment (JRE) or the “Java virtual machine” plays an important role in the overall performance of a Java program. Several vendors have implemented their version of the product, optimised for specific tasks. For the experiments, the reference version of SUN Microsystems JRE version 1.3.1 has been used (the current version at the time).

<table>
<thead>
<tr>
<th>Java version &quot;1.3.1&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java(TM) 2 Runtime Environment, Standard Edition (build 1.3.1-b24)</td>
</tr>
<tr>
<td>Java HotSpot(TM) Client VM (build 1.3.1-b24, mixed mode)</td>
</tr>
</tbody>
</table>

Table 5.2 Java version used during the performance evaluation.

The Java HotSpot VirtualMachine (VM) is an improved version of the just-in-time (JIT) compilers used by previous versions of the Java SDK. According to [116], it “has been specially tuned to reduce application start-up time and memory footprint, making it particularly well-suited for client environments”. A just-in-time VM, instead of interpreting the byte code, compiles it into native code which is then executed much faster. This compilation is performed on a per-method basis. On a second call to a method, it can be executed without having to re-compile it. Therefore it is beneficial for software where methods are called repeatedly as is the case with the PEDFLOW model. The JIT is integral part of the JRE and cannot be disabled. Other improvements of the HotSpot VM are a new memory model, improved garbage collection and fast thread synchronisation.
For the experiments to run in a controlled and repeatable manner, batch processing was used. The sequence of commands was recorded as a macro to guarantee exact repetition. It typically consisted of:

- loading the scenario
- running the simulation until a stable situation is achieved
- resetting the statistics
- running the simulation for a suitable time period
- saving the results to disk
- quitting the program

The equivalent script is shown in Table 3.

<table>
<thead>
<tr>
<th>Command</th>
<th>Argument</th>
</tr>
</thead>
<tbody>
<tr>
<td>open</td>
<td>d:\my Java\PEDFLOWproject\work\xml\experiment.xml</td>
</tr>
<tr>
<td>runTo</td>
<td>600, 0</td>
</tr>
<tr>
<td>statsReset</td>
<td></td>
</tr>
<tr>
<td>runTo</td>
<td>3600</td>
</tr>
<tr>
<td>statsSave</td>
<td>d:\my Java\PEDFLOWproject\work\sts\experiment.sts</td>
</tr>
<tr>
<td>statsValuesSave</td>
<td>d:\my Java\PEDFLOWproject\work\stv\experiment.stv</td>
</tr>
<tr>
<td>statsFreqSave</td>
<td>3</td>
</tr>
<tr>
<td>exit</td>
<td></td>
</tr>
</tbody>
</table>

Table 5.3 Example macro for a single simulation run.

Keeping all external factors constant guaranteed that the results of the experiments were comparable. Experiments were repeated and the resulting measures averaged to account for external factors that could not be (easily) controlled, such as load produced by low-level functions of the operating system.

5.5.3 Model-related factors (internal)

In opposition to the external factors, a number of settings in the model will have an impact on performance. These settings are independent of the modelled scenario and thus from the variable parameters. They are instead concerned with visualising the results of the simulation and/or the collection of data.
For the majority of the performance measurements, all of the optional settings will be switched off or set to the setting with the least performance impact. More precisely:

- no grid lines will be drawn
- entities are visualised as squares
- no additional information (path, ‘rubber band to destination’, observed area) is displayed
- no statistical data about the running of the experiment (apart from the required timing) is collected

These conditions define the minimum requirement (essential features) for a useful modelling process.

### 5.5.4 Methods of measuring performance in PEDFLOW

Performance is measured as the execution time of the model. However since the model works as a series of steps which happen at regular intervals (see section 4.1.4), this time cannot be directly measured as running time because of the buffer time between each step which contributes to the overall running time. This is further complicated by the fact that another thread runs concurrently which is responsible for the graphical representation of the results. There are two ways to achieve a performance measure: total step time and average step times with or without display.

A measurement of total step time can be achieved by modifying the program so that graphical output is suppressed and the gap time is eliminated. This has the advantage that only one measurement (of a relatively large time period) needs to be taken. The average step time can be derived by (programmatically) counting the number of steps and dividing the total time by the count. The disadvantage is that the overhead of the display is not captured. It is mainly dependent on the number of steps, but also on the number of entities being displayed. If the latter is constant, or the average known, an estimate of the time can be made and the time compensated.

Measuring the step times individually introduces a larger error as the measured times are now in the range of the resolution of the timer. If the display thread is run concurrently, its impact is partly (but not completely) contained in the
measure. A better solution would be to run the model and the display thread in sequence. This can be achieved by adding a sufficiently large offset to the buffer time and measuring the display time independently. The total execution time would be the sum of all model and display times.

A modification of the last approach can be used to determine what percentage of modelling steps can be performed within the step time. The measure here is not the modelling time, but the percentage of steps over-running their allocated timeslot.

To summarise: for a given scenario and given value of the variable parameter, the model can perform insufficiently (the overall execution time is greater than the modelled time), sufficiently (while individual step times my take longer than a cycle, the overall execution time is smaller than the modelled time) or it can perform well (step and total execution time are within the limits).

### 5.5.5 Performance evaluation with synthetic data

For the performance measure the approach was chosen to only measure the actual model execution time. While the display can be a significant part of the overall performance, it linearly depends on the number of modelled entities and the size of display (i.e. the zoom level and which part is visible). Another influence is the complexity of display (shape of the agents, additional info that is displayed). There are mechanisms in place to skip updates in time critical situations. Update requests are queued in the display thread and only the last one executed.

However, model time is the total activation time of all agents. The total depends on the number of activated agents, the speed they are moving and the number and complexity of applicable contexts. At a minimum this is the base context, (i.e. walking while avoiding obstructions). Its impact on execution time depends on the size of the observed area and whether obstructions are detected which need avoidance. For other contexts the execution time is impacted by the computational effort for context detection and (if the contexts apply) on the complexity of dealing with the contexts.
Experiment setup

Owing to the intended purpose of using the model in an office environment, a screen size 1024*768 was used for the experiments. A default display size of 10 pixels times 10 pixels per grid element was chosen. Adding a few extra pixels for window frame, menu and status line, 98*65 = 6370 grid elements can be displayed. This is the equivalent of 49m * 32,5m. For comparison, the pavement width of Princess Street is about 10m). A completely unobstructed movement requires about 3m free in front of the agent to be uninhabited. The setup comprises a population of 14 times 65 equal 910 agents moving all the time. That is ~14% (1/7) of a fully populated area. An impression of the model layout can be seen in Figure 2.1.

Simplifying the average speed of 1,4 m/s to 1 m/s results in 10 time units between steps which means the 910 agents make a step every 0.5s (see also Table 3.1).

Figure 5.18 Basic experiment setup to determine performance in a worst case scenario.

Results, interpretation and conclusion

To keep measurement errors down, a time period of 100s was chosen for the run. The execution time on the PC described in section 5.5.2 and without display 27s (±1s). A time period of 100s means 2000 time units which equals 200 steps (or activations). Dividing 27s by 200 steps results in 135ms per step. The problem is: there are only 50ms available per slot. This, however, does not mean that the model can not run in real time, as for every timeslot used, there are 9 unused ones in the experiment which would be used to catch up with the overrun (see also Figure 4.3).
To put the timing in perspective: 135ms for 910 agents means that a single agent's activation uses a time slice of 0.148ms.

A different approach to measurement would be to measure the individual activation times. The problem here is the relatively coarse timer resolution of 10ms that is available in Java. Also storing the result introduces additional overhead into the model engine. The built-in mechanism for extracting measures (see Figure 5.19) which was already introduced in section 0 elevates the latter problem somewhat.

Figure 5.19 Collecting internal measure from model runs.

Repeating the experiment with the option enabled, yielded an average slot time of 160ms. Again the measurement is divided by the number of agents in the simulation in order to achieve normalised, comparable measure:

\[
\frac{160 \text{ ms}}{910 \text{ agents}} = 0.176 \text{ ms.}
\]

The fact that that 0.148 ms and 0.176 ms are within the same order of magnitude confirms the validity of the two approaches.

The next step was to vary the parameters of the experiment:

- Different speed: 0.5m/s
  Total time: 13500ms
  \[
  \frac{13,5\text{s}}{100 \text{ steps}} = 135\text{ms per step} 
  \frac{135\text{ms}}{910 \text{ agents}} = 0.148\text{ms}
  \]

- Different observation area: 1.5m
  Total time: 25000ms
  \[
  \frac{25\text{s}}{200 \text{ steps}} = 125\text{ms per step} 
  \frac{125\text{ms}}{910 \text{ agents}} = 0.137\text{ms}
  \]
The experiments show that the limiting factor here is the number of agents involved in the simulation, and not their parameters or parameters of the modelled environment. This was expected as the number of interactions was controlled and kept to a minimum. It is possible to investigate the impact of added contexts on the performance, but due to the number of possible combinations an exhaustive investigation was not performed. Spot checks however confirmed that there is no great performance penalty, mainly because complex contexts only apply to a limited number of agents over limited time periods. From the performance of the worst case scenario it can be concluded, that the PEDFLOW model is capable of running real-time simulations.

5.5.6 Performance evaluation with realistic data

This experiment aims to confirm the conclusion from the previous section by using parameters from real world. It is investigated how increasing agent density will influence the performance of the model. The chosen scenario is High Petergate in York. It is a pedestrianised Zone for which measures and agent parameters were derived from video analysis (see section 4.3). The chosen dataset is of counter flow.

In the experiment the pedestrian density is increased by multiplying the influx of agents while at the same time maintaining the ‘real’ distributions for the other agent parameters. Initially it was considered to have additionally stationary obstructions in the scenario, but it was established that there was no significant impact on the timings, as the number of interactions between agents and obstructions would be lower than agent-agent interactions.

Experiment setup

Due to the varying number of agents involved it is not suitable to compare total time between runs. Due to technical restrictions (timer resolution) it is not possible to measure individual activations with the required precision. However, the number of activations can be counted, in total and per activation cycle and then normalised by dividing it by the number of agents.

Results, interpretation and conclusion

The program was run for 2h model time with the ‘influx’ (number of pedestrians entering the simulation) changed between the original (very low) value and 100
times the original value. The absolute execution time was measured and the
number of agents involved in the simulation recorded. Both measures were
used to derive a normalised value representing the average execution time per
activation as shown in Table 5.4.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>10</th>
<th>20</th>
<th>60</th>
<th>100</th>
</tr>
</thead>
<tbody>
<tr>
<td>total time (ms)</td>
<td>2614</td>
<td>4636</td>
<td>22642</td>
<td>54669</td>
<td>80596</td>
<td>86004</td>
</tr>
<tr>
<td>activations</td>
<td>47517</td>
<td>96854</td>
<td>503148</td>
<td>1096479</td>
<td>1628156</td>
<td>1644096</td>
</tr>
<tr>
<td>time / activation (ms)</td>
<td>0.0550</td>
<td>0.0479</td>
<td>0.0450</td>
<td>0.0499</td>
<td>0.0495</td>
<td>0.0523</td>
</tr>
</tbody>
</table>

Table 5.4 Total execution time, number of activations and derived average execution time.

Comparing the average activation time to the measures from the previous
section confirms that realistic parameters result in shorter activation times than
the synthetic values from the worst case scenario; in this case about a third. It is
interesting to note that they do not vary significantly with increasing agent
density.

Figure 5.20 show the resulting graph. The local minimum at an influx value of
about 10 is the result of the superposition of two effects: At very low densities,
agents have to observe a comparatively large area in front of them to make a
decision about their movement, as it is unlikely that they will encounter an
obstruction. But with increasing density the likelihood increases that another
agent will be blocking the path and a decision about the (avoiding-) action can
be made sooner. With even higher density more and more agents will be in a
position where no avoiding actions are possible or additional contexts like side-
stepping and swapping become active.

Figure 5.20 Average activation time in relation to agent influx.
The results of the presented experiments show that performance of the model is adequate to achieve real-time simulation speed. Scaled-up ‘real’ scenarios give a better idea about performance than synthetic scenarios which in turn are useful to determine the “hard” performance limit of the model. The more complex the scenario the harder it is to reason about the result.
6 Discussion and future work

6.1 Conclusion

The PEDFLOW model was developed to overcome the limits of existing traffic models when applied to people, whose freedom of movement as well as acceleration/deceleration profiles differ greatly from those of vehicles. Using an autonomous agent approach, every pedestrian in the model operates independently (but not in isolation) to achieve a goal (typically to reach a location). Within the virtual world an agent reacts to passive entities (e.g. obstructions) and interacts with active entities (e.g. other pedestrians). As a result of these microscopic, individual actions macroscopic behaviours emerge which are comparable to the observed behaviours of real pedestrian flows. The model can be described as autonomous agent-based, emergent, microscopic model that feature agents with spatial awareness, mobility and graphic representation.

The model satisfies the requirements for use as a tool in the urban planning process in the following categories:

Realism and suitability

It has been shown that the model is able to create realistic macroscopic pedestrian flow characteristics for low to medium density scenarios. Typical examples of macroscopic phenomena such as platooning can be recognised in the model output.

Extensibility

The mechanism of context-mediated behaviour that was developed for PEDFLOW allows the incorporation of other contexts without breaking the functionality of the model.

Usability

It has been informally verified that the model can be used by third parties to setup and run simple scenarios. However, more work needs to be done to convert the prototype into a production application (see also section 6.3).
Portability
The model is implemented in Java and therefore runs on any computer hardware with a Java virtual machine. Adaptation to different interface requirements is possible due to the modular structure.

Performance
The program has only moderate memory requirements and performance on the test computer was sufficient for faster than real-time modelling. Worst-case scenarios that limit the scalability of the model have been explored. However, advances in computer technology mean that faster computers and better Java run-time environments are readily available even in the office environment, so the achieved simulation speed can be considered sufficient.

Overall it can be concluded that the approach to pedestrian modelling presented in this thesis offers the potential for the creation of a modelling tool that can be useful to help town planners in their task to create an attractive walking environment, where people can walk with a minimum of delays.

6.2 Contributions
This thesis makes the following original contributions to knowledge:

Classification of pedestrian models
As part of the literature review of existing pedestrian models, a classification was created that helps to compare not only autonomous agent models, but also other forms of pedestrian models with regards to the following categories: their primary task, source of knowledge used in the decision-making process, extent of spatial awareness, inter-agent communication, creation and termination, implementation/ programming language, agent representation/visualisation For its intended purpose it is superior to other classifications for agent systems (which suffer from the broad definition of the term “agent”), because it ignores agent aspects that are not applicable to pedestrian models. Instead it emphasises features that help to distinguish between these models.

Combining autonomous agent technology and microscopic modelling
While agent technology has been used in pedestrian models before (e.g. in the STREETS model [37]), at the start of the project it had never been done on a microscopic level. Here simple CA models, such as Blue and Adler [30], which
are suitable for flow statistics but fail to model more complex situations, dominate. PEDFLOW uses agents technology on a microscopic level, where every agent presents a single pedestrian and movement is modelled on a step-by-step basis. The combined effect of the individual microscopic actions is an emergent self-organisation of the agents which is observable as macroscopic phenomena.

**Use of data derived from observational studies as model parameters**
The lack of knowledge about how pedestrians navigate their environment meant that a novel methodology needed to be developed in order to extract this information from video footage. With a combination of bought-in image analysis software and a custom written program, a semi-automatic workflow was developed that allowed the extraction of measures such as distance and speed with reasonable efficiency and produced parameter distributions as direct inputs to the model.

**Modelling multiple concurrent influences and context-mediated behaviour**
While basic behavioural rules in combination with realistic parameters can account for the majority of the agents' behaviours when navigating the virtual environment and avoiding other agents and static entities, they don't allow the modelling of more complex situations as the additional triggers required are not there, nor is a mechanism to decide arbitrate between different influences. A mechanism called context-mediated behaviour has been devised as a means of controlling agents affected by multiple concurrent influences (e.g. group dynamics when walking with a partner). The mechanism works by evaluating the required actions for all applicable contexts and then making a weighted choice from the results to determine the behaviour of the agent.

**Microscopic measures of service**
Apart from the visual output, the model also provides quantitative data about the simulations. Areas of interest (e.g. around bottlenecks) can be marked in the modelled scenarios where specified data is gathered. The available choices include simple counts, conditional counts and time measures as well as aggregate functions and simple statistical functions applied to these measures. They are called microscopic Measures of Service (MOS). While the technical basis is there, the interpretation of the values and the assessment of their
relative levels of usefulness in helping with the evaluation of urban layouts is still an unresolved issue.

6.3 Exploitation and future work

The existing prototype implementation should be seen as a proof of concept. To turn the program into a product, more work needs to done in the following four areas:

**Input data**

The biggest strength of the model is also its greatest weakness: the use of parameters derived directly from real world measures to ensure correct modelling requires these parameters to be available. To make the PEDFLOW model available for practical use, more behavioural research needs to be performed. In recent years new research (e.g. Daamen and Hoogendoorn [117] or Kerridge et al. [118]) has opened up new ways to gather the data which will hopefully make the collection process less cumbersome. But information needs not only to be gathered and analysed, but also prepared for universal use. This could, for example, include grouping parameter distributions in suitable profiles. The town planner should be able to create a virtual agent population for a simulation that corresponds to the expected pedestrian population to avoid situations where, for example, the layout of a sports ground is evaluated with agents using a pensioner profile. Although [70] offers some suggestions, it is not completely clear how these profiles could be created. It would be desirable to have expertise in the area of statistics and psychology available.

**More contexts**

The currently implemented contexts (beside the basic context of moving while avoiding obstructions) are: position swap at bottleneck, walking with a partner, pedestrian crossing, attractor, bus queue and ATM queue. While they cover a wide variety of situations, more will be required depending on the place to be modelled.
Possible candidates are:

- “walking in a group”, which could be implemented as an extension of the “walking with partner” context.
- “window shopping”, which could be implemented as an extension of the “attractor” context.
- “repulsor” (the opposite of an attractor, e.g. an aggressive salesman), though the pedestrian behaviour in this scenario needs further study.
- “unregulated road crossing”, which could be implemented as an extension of the “pedestrian crossing” context, but which would require the existence of vehicular traffic to be modelled.

Every new context would need to be integrated into the model framework and evaluated by comparing the output of the model against the real world.

**Usability**

In addition to the improvement of the model there is work to be done to make it accessible to the target user, the urban planner. The current user interface is functional, but would profit from an adjustment according to current interface design guidelines. Additional drawing tools for aiding the setting up of new scenarios would be helpful for the town planner using the model as a tool, as would be an import filter to read existing plans or blueprints. As already mentioned in section 6.2, the effective use of MOS requires further efforts. A thorough requirement study and associated field tests is suggested in order to establish guidelines for their use. Ideally the result would be a set of quality of service levels similar to the ones developed by Fruin in [65], but adapted for the microscopic measures of the model. Such levels could be used without the detailed knowledge of the underlying technical implementation, thereby increasing the utility of the software in the urban design process.
7 References


[104] Carreno, M., Quality of service provided for vulnerable pedestrian groups in city centre walking environments, PhD Thesis, School of Computing, Napier University, 2005.


[106] Gallagher, K., Extending PEDFLOW to include queue formation behaviour, MSc Thesis, School of Computing, Napier University, 2001.


Appendix: Implementation details

Although the design of the PEDFLOW model is parallel, its implementation is serial. Entities are instantiated from data files, or generated dynamically by agents. Their position in space is recorded as their index in a collection class Grid. Precision is determined by gridsize, the smallest unit of measurement. It also determines the size of the entities. Their position in time (next activation) is recorded in a collection class TimeSlot. Time is measured in multiples of a base unit. Movement is a change of position in Grid and TimeSlot. In order to represent a sensible selection of different speeds, time and space units must relate.

PEDFLOW is an object oriented software product and as such it can be best understood by looking at the various classes, their use and relationships. The most important ones are the entity classes used to represent real life objects ranging from street furniture to pedestrians. To keep track of them within the software model, a set of modelling classes is used. The user interface classes allow controlling the model. They use and are mainly based on swing classes. Last but not least there are a lot of utility classes used to encapsulate and pass information within the model or just generally make the workings more transparent.
1 Modelling Classes

The four modelling classes are: Grid (position in space), Timeslot (time management), Entities (list of all entities) and Simulation (run-time control). The first three are basically collection classes that allow efficient access to entities depending on requirement. This is different to other models (generic agent frameworks) where there is typically only one unstructured collection class that gets filtered according to purpose. The approach used in PEDFLOW can be considered as a data base interpretation: indexes are used for fast object access and a more intuitive design approach. It can be tweaked to compromise space for speed and vice versa.

Grid
Extends JPanel but overwrites the Swing implementation of add, remove, paint etc. Uses a 2D array for blocking entities and an additional 2D array for non-blocking (feature) entities. Contains methods for adding and removing entities and for observation, movement.

TimeSlot
A ring buffer to keep track of time. Times that are greater than the ring buffer size can be achieved using the loop counter

Entities
Extends ArrayList to include a new constructor and some special methods for adding and removing and saving entities.

Simulation
Extends JPanel and implements runnable. As a JPanel it can be inserted in any Swing container. It provides the low-level interface to the model that can be called by the user interface. Also contains the thread spawning features for running the model out side the main event loop.
2 Entity Classes

Not only pedestrians but also:

- Static, passive entities (e.g. street furniture)
- Static, active entities (attractors / repulsers)
- Mobile entities (e.g. vehicles)
- Compound entities (groups of …)
- ‘Quality’ of area (pavement, road, noise)

They all have a set of common features, e.g.:

- constructor from text representation
- toString (creates text representation)
- clone method
- parameters: x,y co-ordinates
- paint mechanism (colour/shape)
- pop-up editor

and specialised Features according to their purpose, e.g.:

- road: non-blocking (more than one entity per grid element)
- active: activation delay
- mobile: goal co-ordinates
- person: parameters and sophisticated activation method
<table>
<thead>
<tr>
<th>Entity Type</th>
<th>Purpose</th>
<th>New Fields</th>
</tr>
</thead>
</table>
| Entity            | root class defines common features for all other entities not actually used in the model | x - x position  
y - y position  
verbose - flag for verbose display (selected in prefs)  
blocking=true - flag to indicate that it is a blocking entity (only one blocking entity can exist at a grid position)  
active=false - flag to indicate that an entity gets activated periodically  
nosave=false - flag to indicate that it is a transient entity (not to be saved in XML) |
| Obstruction       | obstruct a single grid position used to model bins, lamp posts etc |                                               |
| Blockage          | obstruct a number of grid positions together as one block used to model buildings etc | xSize - size in x direction  
ySize - size in y direction |
| Active            | obstruct a single grid position perform a task periodically (every timeUnit) not used in modelling (abstract class) | startDelay - delay after which it appears in model  
modifies: active=true |
| Edge              | obstruct a single grid position unless it is directly in front of a person  
if person is about to step on it, person is taken out of model used to model entrances to buildings open streets |                                               |
| Road              | just to colour an area differently | blocking=false - flag to indicate that it is a non-blocking entity |
| Mobile            | simple movable object not used in model but basis for further specialisation | goalx  
goaly  
repeatCount  
repeatDelay |
| Person            | modelling pedestrian main entity | DWS – desired walking speed  
SC -  
DDB – deviation distance  
GSB - |
| AdvancedMobile    | better speed modelling than mobile | stepSize  
pause |
| Attractor         | - modelling street musicians etc | attractiveness |
| Tight             | - modelling pedestrian crossings | xsise  
ysize  
redtime  
greentime  
direction |
| MOS               | - area where data is collected  
- used to define/extract microscopic levels of service |                                               |
3 User Interface Classes

The PEDFLOW model (see simulation class) can run in various environments. The following is true for the generally used PedflowApp application:

**PedflowApp**
The main application class.

**MainFrame**
Extends JFrame and makes it into a fully fledged Application window with Menu, Toolbar and statusline. Basic Swing classes, mainly designed using the interface builder.

**Grid Editor**
Extends JDialog and provides a user interface to editing the grid. Mainly designed using the builder (thus messy). Close co-operation with grid and entities. Implements Mouse Listener for edit-related mouse clicks on grid. Makes use of editor to edit parameters of entities.

**Editor**
Extends JDialog and is a flexible way to edit values of an entity. Will get spawned by the entity in question and re-used if required. Note how the value list gets built up from the lowest entity class.

**Prefs**
Extends JDialog and provides a user interface to setting various run-time parameters. Mainly designed using the builder (thus messy). It also serves as a central repository for various setting that will stored in prefs.txt.

**AboutBox**
Useless JDialog to display some version info.

**RunTo**
JDialog to allow user input for time-limited modelling
4 Utility Classes

Dir
Class to express direction and relative direction change. It is basically an integer with various handy methods to change it or extract physical co-ordinates from it. Also includes draw mechanism, which is employed by Person.

Observation
This is to abstract the area in front of a pedestrian (regardless if walking direction). It contains a 2 dimensional array (3xDD) of entities in front of the person in question. When qualified, distances are made relative according to internal parameters - this used to be more sophisticated, but is rather pointless with the current rules.

Action
Abstract data type (ADT) to store potential actions to take and their probability.

Distribution
This has been developed in order to deal with weighted, random choices. Each integer value within a certain range has a number associated that represents its probability. The actual numbers and hence the sum is unimportant, but the sum represents a likelihood of 100%. There are various constructors for this class.

EntityNode
An ADT to simplify the DOM interface for use in PEDFLOW. Can hold a DOM org.w3c.dom.Node and is typically created from an XML document. It can hold a branch of it, usually:

<grid> (if top level) or <entity> in its domNode variable. Use get() to get values for entity parameters or use getFirstEntity/getNextEntity to get nested entities. GetDistribution is used to get a Distribution from the current position.

EntityTableModel
Part of the implementation of the entity table display in the Grid editor. Currently a little over the top as a large amount of functionality has been removed. Also note the sorter class.
Parameters for PEDFLOW entities:

<table>
<thead>
<tr>
<th>parameter (unit) [XML label]</th>
<th>type, range</th>
<th>purpose</th>
<th>entity / comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attraction [attraction]</td>
<td>scalar, 1-100</td>
<td>value describing the attractiveness</td>
<td>entity</td>
</tr>
<tr>
<td>Deviation Distance [dd]</td>
<td>dist, 0-20</td>
<td>deviation distance</td>
<td>person</td>
</tr>
<tr>
<td>Direction [direction]</td>
<td>scalar, 0=N 1=NW 2=W 3=SW ..., 0 - 7</td>
<td>direction for which the light is active</td>
<td>traffic light</td>
</tr>
<tr>
<td>Speed (dm/s) [dws]</td>
<td>dist, 5 - 25</td>
<td>desired walking speed of the agent</td>
<td>person</td>
</tr>
<tr>
<td>Green Time (s) [greentime]</td>
<td>scalar, 1 - 300</td>
<td>time for which the light is inactive</td>
<td>traffic light</td>
</tr>
<tr>
<td>Pause [pause]</td>
<td>scalar, unrestricted</td>
<td>delay between steps; together with step size this determines the speed</td>
<td>advanced mobile</td>
</tr>
<tr>
<td>Range [range]</td>
<td>scalar, 1 - 40</td>
<td>radius of the circle of influence</td>
<td>attractor</td>
</tr>
<tr>
<td>Red Time (s) [redtime]</td>
<td>scalar, 1 - 300</td>
<td>time for which the light is active</td>
<td>traffic light</td>
</tr>
<tr>
<td>Repeat Count [repeatcount]</td>
<td>scalar, 0 - unlimited</td>
<td>number of new entities to create</td>
<td>mobile entities</td>
</tr>
<tr>
<td>Repeat Delay (seconds) [repeatdelay]</td>
<td>dist, 0 - unlimited</td>
<td>delay after which a new entity of the same type and parameter distribution is created</td>
<td>mobile entities</td>
</tr>
<tr>
<td>Ring [ring]</td>
<td>scalar, 1 - 40</td>
<td>radius of the ring around which people will assemble</td>
<td>attractor</td>
</tr>
<tr>
<td>Side Choice [sc]</td>
<td>dist, 0=left 1=straight 2=right 3=undecided</td>
<td>preferred side choice</td>
<td>person</td>
</tr>
<tr>
<td>Start Delay [startdelay]</td>
<td>scalar, 0-1000000</td>
<td>delay before the entity becomes active and enters the grid</td>
<td></td>
</tr>
<tr>
<td>Step Size [stepsize]</td>
<td>scalar, 1 - 5</td>
<td>number of grid elements covered in one step</td>
<td>advanced mobile</td>
</tr>
<tr>
<td>X goal [goalx], Y goal [goaly]</td>
<td>dist, unlimited dist, unlimited</td>
<td>goal co-ordinates</td>
<td>mobile entities</td>
</tr>
<tr>
<td>X position [x], Y position [y]</td>
<td>dist, 0 - maxX dist, 0 - maxY</td>
<td>position of entity in the grid (start position for mobile entities)</td>
<td>all entities</td>
</tr>
<tr>
<td>X size [xsize], Y size [ysize]</td>
<td>scalar, 1 - (maxX-x), scalar, 1 - (maxY-y)</td>
<td>size of entity</td>
<td>blockage, road, MOS</td>
</tr>
</tbody>
</table>
Notation for parameter distribution

<table>
<thead>
<tr>
<th>Type</th>
<th>Syntax</th>
<th>comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single value</td>
<td>&lt;value&gt;</td>
<td>one value with 100% probability</td>
</tr>
<tr>
<td>List</td>
<td>L &lt;min value&gt; &lt;probability&gt; &lt;probability&gt; ...</td>
<td>lists the individual probabilities for a series of integer values starting with &lt;min value&gt;; the number of available probabilities determines the length of the series</td>
</tr>
<tr>
<td></td>
<td>(example: L 5 1 2 1 means 25% probability for 5, 50% for 6 and 25% for 7)</td>
<td></td>
</tr>
<tr>
<td>Equal</td>
<td>E &lt;min value&gt; &lt;max value&gt;</td>
<td>describes a series of integer values with equal probability in the range of &lt;min value&gt; and &lt;max value&gt;</td>
</tr>
<tr>
<td></td>
<td>(example: E 5 6 7 8 means 25% probability for 5, 25% for 6, 25% for 7 and 25% for 8)</td>
<td></td>
</tr>
<tr>
<td>Normal</td>
<td>N &lt;min value&gt; &lt;max value&gt; &lt;mju&gt; &lt;delta&gt;</td>
<td>describes a series of integer values where probability of the values is normally distributed according to &lt;mju&gt; and &lt;delta&gt;; only values in the range of &lt;min value&gt; and &lt;max value&gt; are considered</td>
</tr>
<tr>
<td></td>
<td>(example: N 1 3 2 1 means 13% probability for 1, 36% for 2 and 50% for 3)</td>
<td></td>
</tr>
</tbody>
</table>

5 Context-mediated Behaviour

The following contexts are currently implemented:

previous move was a forced swap
stay in place and delay a bit longer

goal reached
remove subgoal

if no more subgoals remove entity from grid

partner
check if still close enough (within limit)

if tracking required, try and walk towards partner

possible traffic light
if affected by a light (in area, same direction and red) do not move outside area

possible attraction
if person is attracted, insert new subgoal and adjust direction

moving towards goal
find best move according to following algorithm:

value (Straight,Left,Right) = object distance - parameter
if(S>0) go straight
else if((L>0) & (R<=0) go left
else if((R>0) & (L<=0) go right
else if((R>0) & (L>0) undecided
else if((R<=0) & (L<=0) crowded

could not determine a good move
if previous context resulted in ‘undecided’ predict the future
the same algorithm is applied

still could not determine a good move
use SIDE CHOICE values

is crowded
if all places in front are occupied, avoid left / pause / avoid right with equal likelihood

opposite moving person in front
swap places with person and flag it for speed adjustment

6 Difference between an attractor and a traffic light

The following table compares two similar contexts (attractor and traffic light) with regards to the implementation of the context change triggers.

<table>
<thead>
<tr>
<th></th>
<th>Attractor</th>
<th>traffic light</th>
</tr>
</thead>
<tbody>
<tr>
<td>condition for effect</td>
<td>not currently attracted&lt;br&gt;not previously attracted&lt;br&gt;within the range of an attractor&lt;br&gt;random choice whether attracted or not based on attractiveness</td>
<td>walking towards a traffic light (within area)&lt;br&gt;light is red</td>
</tr>
<tr>
<td>Effect</td>
<td>change subgoal&lt;br&gt;crowd context only (no side stepping)</td>
<td>crowd context only (no side stepping)&lt;br&gt;not to leave area</td>
</tr>
<tr>
<td>Stop</td>
<td>reached a certain distance to attractor (implemented by attractor query)&lt;br&gt;stopped by person in front</td>
<td>reached road (implemented by attractor query) - problem: overflowing?&lt;br&gt;stopped by person in front</td>
</tr>
<tr>
<td>premature abortion</td>
<td></td>
<td>light is green</td>
</tr>
<tr>
<td>normal restart</td>
<td>time elapsed</td>
<td>light is green (either polling or time elapsed)</td>
</tr>
</tbody>
</table>