Proceedings of the Operational Research Society Simulation Workshop 2018 (SW18)
A.Anagnostou, R.Meskarian, D.Robertson eds.

PROPOSAL OF A DESIGN PATTERN FOR EMBEDDING THE CONCEPT OF SOCIAL FORCES IN HUMAN CENTRIC SIMULATION MODELS

Peer-Olaf Siebers
YuFeng Deng
Jonathan Thaler
Holger Schnädelbach
Ender Özcan

School of Computer Science, University of Nottingham
Nottingham, NG8 1BB, United Kingdom
peer-olaf.siebers@nottingham.ac.uk

ABSTRACT

There exist many papers that explain the social force model and its application for modelling pedestrian dynamics. None of these papers, however, explains how to implement the social force model in order to use it for systems simulation studies. In this paper we propose a design pattern (reusable template) that supports the implementation of the social force model within an artificial lab to run experiments for human centric systems. It allows considering not only people but also static and moveable markups. We demonstrate how to implement the design pattern in two commonly used agent-based modelling packages, Repast Simphony and AnyLogic. For this we use an illustrative example from the Adaptive Architecture domain. Both packages require a slightly different implementation strategy, due to the API constraints they provide. Overall, we found that the design pattern provides very helpful guidance when working on the individual solutions for the different packages.

Keywords: Design Pattern, Pedestrian Dynamics, Social Force Model, Agent Based Simulation, Human Centric System

1 INTRODUCTION

In their paper "Social Force Model for Pedestrian Dynamics" Helbing and Molnar (1995) suggest that the motion of pedestrians can be described as if they would be subject to social forces. The paper then offers a mathematical model of pedestrian motion that describes the influence of social, psychological, and physical forces on pedestrian's choice of direction. When Helbing and Molnar talk about simulation examples, they provide parameter settings and justifications for these but they don't provide information about how to get from a set of equations to a fully-fledged and operational simulation model. Since the publication of the original paper, several related papers were published with extensions and applications of the Social Force Model (SFM) (for an overview see Duives et al, 2013). Still, the problem remains, that there is no clear advice on how to implement these.

When we worked on a simulation project involving the modelling of pedestrian movement, we found it quite difficult to do the implementation, as we came across many obstacles when we applied the SFM to real world scenarios within enclosed spaces. In this paper we want to share our experience and give some advice on how to implement the SFM for such scenarios. To ease the process of model development we have created a reusable template (design pattern) for modelling people's and object's motion behaviour and for modelling the environments they are moving in. Our design pattern supports the development of "artificial labs" for exploratory and explanatory studies of human centric systems in many different

domains. In this paper we focus on Architecture, but we have also used the proposed design pattern to develop models for studies in Social Science and Human Factors. For the modelling activities we employ an object oriented agent-based modelling approach. In the following we use some object oriented design principles and tools from Software Engineering to describe our proposed design pattern. The feasibility studies we conducted in the different domains helped us to identify several artefacts within the models and to work on solutions for removing these artefacts. While the resulting design pattern might not be perfect for all applications, at least it offers a starting point for a better understanding of the translation process from equations to code and provides a basis to work from.

While there are several simulation software packages that offer embedded pedestrian libraries, as for example AnyLogic (www.anylogic.com, accessed 1/12/2018), the issue is that these libraries use a black-box approach, meaning that they hide all details from the user and only provide a user interface. In contrast to this, we build our pedestrian representation bottom up, using a white-box approach. This provides transparency regarding the design and complete control over all model parameters. Furthermore, it supports specialisation of the agent; our design pattern can easily be extended to accommodate specific behaviours and capabilities the agent needs to possess, in order to best represent its real world counterpart. Depending on the purpose of the modelling exercise our approach provides the user with a big advantage by allowing this kind of control.

2 BACKGROUND

2.1 Agent-Based Modelling

Agent-Based Modelling (ABM) is the current state of the art in modelling and simulation for pedestrian dynamics (Castle and Crooks, 2006) and is based on treating each pedestrian as an individual "agent" that constantly assesses its situation and makes decisions based on a set of predetermined rules. Individual agents interact with each other and their environment to produce complex collective behaviour patterns at system level; agents are designed to mimic the behaviour of their real world counterparts. In contrast to objects, which are part of the environment, agents are capable of making autonomous decisions and showing pro-active behaviour. This naturally lends itself to modelling of pedestrians in a heterogeneous crowd. Movement decisions making of these pedestrians can be considered using the concept of social forces (Helbing and Molnar, 1995).

2.2 Social Force Model

The SFM is a socio-psychological model which could simulate the movement of pedestrians under simple situation (Helbing et al, 2000). It was originally introduced for simulating escape panic; however there are many studies which applied it to other social simulation scenarios and added their own improvement.

The SFM treats each agent as if it had an electrical charge, and so as two agents move towards each other they feel a repulsive effect. They also receive an attractive force from their destination point (usually an area). The resultant force acts on the agent, and gives it an acceleration (or deceleration), adjusting the speed of each agent. In addition to these psychological forces, when agents are physically touching two physiological forces are produced, based on granular interaction forces: a tangential force, and a frictional force. The same combination of psychological and physiological forces is produced when interacting with walls and barriers (boundaries). The related set of equations that allows computing the force on an individual agent at each time step can be found in Helbing et al (2000).

The Extended SFM (ESFM) introduced by Xi et al. (2010) defines a connection range and an attention angle, in order to add the notion of "vision" to the SFM. These two factors define the sector in which a person would react to the psychological forces caused by others that are in sight as people cannot respond to actions of others, which they cannot see. Furthermore Xi et al. (2010) consider group behaviour in form of a "socially attractive force" between members of a group. The group behaviour is implemented by considering an "intimate factor" that is multiplied with the psychological force. The main

idea is that a positive psychological force is applicable for people belonging to different groups while a negative psychological force is applicable for people belonging to the same group. Attention should be paid to the fact that a positive psychological force actually implies that agents are psychologically against each other. The extended set of equations that allows computing the force on an individual agent at each time step, under consideration of vision restrictions and group behaviour, can be found in Xi et al (2010).

The latest addition to the SFM family is the Headed Social Force (HSF) model by Farina et al (2017). It allows to consider a preferred direction of motion. In general humans prefer to move forward, while in the original SFM it is assumed that humans move isotropically (i.e. in all directions). The HSFM introduce a pedestrian's heading into the dynamics of the SFM. This improves the realism of the trajectories generated by the classical SFM.

3 THE DESIGN PATTERN

When we worked on the models for the feasibility studies mentioned above we realised that there were some reoccurring patterns concerning the structure of these models. We could start with a kind of template base model to represent a basic system and its element. Then the main thing to think about was to what class of elements the objects/agents we wanted to model belong to and if they were agents, how we should define the activities they would potentially engage in.

This got us to the idea to create a design pattern that follows object oriented principles and Software Engineering best practices, providing a structured approach for modelling scenarios of peoples' movement in non-enclosed and enclosed spaces which contain non-moving and/or moving object. Such a structured approach provides robustness to the resulting model and makes it easy to maintain and easy to extend.

3.1 Overview

Wikipedia (https://en.wikipedia.org/wiki/Software_design_pattern, accessed 1/12/2018) defines that "In Software Engineering, a software design pattern is a general reusable solution to a commonly occurring problem within a given context in software design. It is not a finished design that can be transformed directly into source or machine code. It is a description or template for how to solve a problem that can be used in many different situations". Such a design pattern represents best practice for solving common problems, without specifying the final application classes or objects that are involved. Framing any given problem in terms of design patterns forces a deeper understanding of the problem, and leads to readability, flexibility and stability (Bersini, 2012).

Our design pattern is presented using UML, a notation which is commonly used for object oriented analysis and design in Software Engineering. A concise introduction to UML, sufficient for understanding the following, can be found in Bersini (2012). A more comprehensive coverage of the topic is provided in Booch (2007).

For capturing structural aspects we use a class diagram and for capturing behavioural aspects we use a state machine diagram. Figure 1 shows the structural aspects of the design pattern. Initially we used the ESFM as the driving force for our people and intelligent moving objects. Later we decoupled the ESFM from the rest of the design pattern so that it can be easily exchanged with a more advanced version of the SFM without breaking the design pattern (as for example the HSFM).

3.2 A Closer Look at the Individual Classes within the Design Pattern

In this section we only explain the most relevant details. For a better understanding of the details the reader is advised to take a look at the illustrative example in the next section and the models related to the illustrative example, which can be downloaded from www.openabm.org.

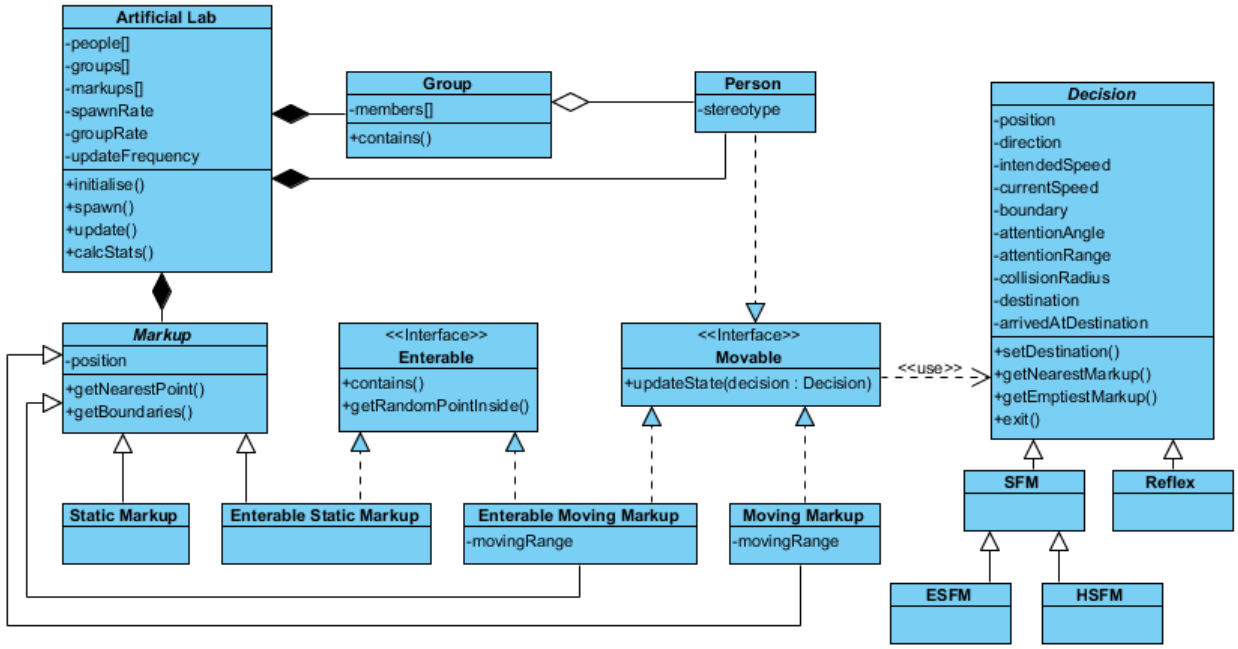

Figure 1 *Structural aspects of the design pattern*

3.2.1 Person, Group, and Decision

The design pattern shows that the "Person" class implements the "Movable" interface. In practice this means that it needs to implements the method "updateState(decision:Decision)" which receives as a parameter an object of type "Decision". This "Decision" object is tailored towards the jobs it has to do within the "Person" instance. The "Group" class simply maintains a collection of "Person" agents, and allows checking if an agent is a member of a specific group. The "Person" and "Group" agents are generated by the "Artificial Lab".

From a behavioural point of view in the simplest case a person is walking towards a desired position and once the desired position is reached the person is resting either for a period of time or until a condition becomes true. This is expressed in the state machine diagram in Figure 2.

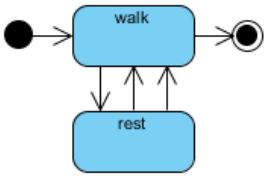

Figure 2 *Simplest behaviour of a "Person" agent*

During the initialisation of the "Person" agent a destination is defined using the "setDestination()" method and the variable "arrivedAtDestination" is set to false. The "Person" agent keeps waking until the destination is reached. The path the person agent is following is directed by the SFM, which is embedded in the "updateState(…)" method which is constantly called in the background. The pseudocode of this method is shown in Figure 3. Note that the "continue" keyword causes the loop to immediately jump to its next iteration. Until the "Person" agent has arrived at its destination, the social forces, acceleration, speed, position and direction are updated at every time step. Once the "Person" agent is at its destination or is in the process of approaching its destination, it will stop moving (the state in the state machine diagram will change from "walk" to "rest") and therefore stop calculation all movement related factors. As soon as a

new destination is set (which happens outside the "updateState(…)" method), the variable "arrivedAtDestination" will be reset, the state in the state machine diagram will change from "rest" to "walk", and the "Person" agent starts moving again. At some point the "Person" agent will have reached the final destination (e.g. the exit of a building) and will be destroyed by calling the "exit()" method.

```
For each time step do
   If arrivedAtDestination // check if it has been reset externally
      Continue
   Else if distance (position, destination) < threshold
      arrivedAtDestination=true // register arrival
      Continue
   Else
      Calculate social forces // using ESFM
      Calculate acceleration
      Calculate speed
      Calculate position and direction
   End if
End do
```

Figure 3 *Pseudocode of "updateState(decision:Decision)" method*

A very useful feature that allows us to model the behaviour of heterogeneous crowds (in a controlled way) is the possibility of defining stereotypes within the "Person" class. These stereotypes could influence things like speed, collision radius, and group membership of a person agent.

3.2.2 Markup, Enterable, and Movable

Markup shapes are usually used to represent boundaries (e.g. walls) or obstacles (e.g. pillars) or spaces (e.g. areas). Some of these markups are not enterable (walls and pillars) and some of these markups are enterable (areas). If they are enterable the concrete classes need to implement the methods listed in the "Enterable" interface (i.e. "contains()" and "getRandomPoint()"). As we had several examples where we also had to represent moving markup shapes (e.g. moving walls, moving display windows) we decided to also add a "Movable" interface to our design pattern, which requires the concrete classes to implement an "updateStats()" method. This method receives a "Decision" object as parameter which allows the concrete movable object (including "Person" agents) to make autonomous decisions (by using the SFM or any other method) about their next step. While Markup is an abstract class, "Enterable" and "Movable" are both interfaces. The reason behind doing it this way is that some programming languages (e.g. Java) only support single inheritance. By using interfaces we avoid issues, when it comes to the implementation of the design pattern in these languages.

3.2.3 Specialised Markups

The classes "Static Markup", "Enterable Static Markup", "Enterable Moving Markup", and "Moving Markup" are examples for concrete classes that extend the abstract class "Markup" and might implement one or both of the interfaces - "Movable" or/and "Enterable". In practice these concrete classes would represent and get names like "Wall" for a "Static Markup" class, "Area" for an "Enterable Static Markup" class, "MovingPlatform" for an "Enterable Moving Markup" class, and "Robot" for a "Moving Markup" class. By applying this toolbox approach we produce specialised markup objects with additional functionality. Using inheritance and interfaces helps to enforce contracts and therefore the whole becomes easier to maintain. In addition, abstract classes help to reduce code duplication (and consequently bug propagation) as they provide default implementations for shared methods.

3.2.4 Artificial Lab

The artificial lab hosts all elements of the model (people, groups, and markups) and is responsible for providing statistics. It controls the creation and deletion of the elements during initialisation and runtime of the simulation (indicated by the composition relationship). The "initialise()" method is used to set up the simulation (i.e. to create all relevant elements and to set all parameters, including the ones for the ESFM). The "update()" method drives the simulation forward and is triggered automatically on a regular basis (e.g. defined by "updateFrequency", e.g. every second of simulated time) once the initialisation is completed. It uses the algorithm presented in the pseudocode in Figure 4.

```
Loop over time horizon
   If required, create new agents
   Loop over shuffled list of agents. For each agent A in list:
      Execute agent A behaviour
      Update state of agent A (based on agent A's state, the states of agents that
      interact with agent A, and the state of the environment).
      Update other agents states and the environment (if appropriate)
      Update agent A's stats and group stats (if appropriate)
   End loop over randomized list of agents
   Update collective stats
Increment t in time loop and repeat until end of simulation time horizon
Calculate final stats
```

Figure 4 *Pseudocode of "update()" method*

As Figure 4 shows, statistics might be calculated and displayed or stored during runtime or at the end of the execution. There are different types of statistics that can be collected: statistics for individual agents (e.g. current state); statistics for groups of agents (e.g. "currentGroupSize"); collective stats per round (e.g. "currentNumberOfAgents"); results at the end of the simulation run (e.g. "averageNumberOfAgents" or "averageTimeInSystem"). The "spawn()" method simply creates a new "Person" agent at an average interval of 60/spawnRate, where "spawnRate" is the average number of people arriving per 60 seconds.

4 IMPLEMENTATION OF THE DESIGN PATTERN

When it comes to implementing design patterns there are often constrains dictated by the software packages and libraries used to implement the pattern. While the pattern works well on a conceptual basis we have to consider some trade-offs for the implementation. Here we demonstrate the application of the design pattern for an exploratory illustrative example that we have implemented in two different simulation packages that are commonly used in the field of agent-based modelling. These are Repast Simphony 2.5 (https://repast.github.io, accessed 1/12/2018) and AnyLogic 8.1 PLE (www.anylogic.com, accessed 1/12/2018). While both implementations are based on the same design pattern their implementation differs considerable. The implementations provided here are still work-in-progress and through further refactoring we should be able to get them even closer aligned to the structure of the design pattern. However, this is outside the scope of this paper. Here we only provide a proof-of-concept implementation.

Due to space constraints we are only able to present a description of the illustrative example and then we focus on the constraints that we experienced with the different packages during the implementation phase. Both models are available for download from www.openabm.org. They are well documented and should therefore be easy to understand.

4.1.1 Illustrative Example from Adaptive Architecture

The following illustrative example describes the development of an exploratory simulation model of a non-existing human-centric system. Here we are conceptualising some ideas on adaptive architecture, using the design pattern as a basis for building an artificial lab. The aim is to develop a model that can help to study the impact of an adaptive wall (a wall that moves forwards and backwards) inside a room. The adaptive wall separates the main room into two smaller room parts while there are gaps which people can still move through to another room. The general objective of such an adaptive design is to maximise the space utility, making the room with more people inside larger and vice versa.

Here we do not only use the ESFM for modelling the movement of people but also for modelling the movement of a movable markup (i.e. the adaptive wall). This means that the adaptive wall's movement is guided by social forces. Such modelling capability will be particularly useful if we think about the fact that we will be surrounded by intelligent moving objects in the not so distant future, and the design pattern supports the model design for such systems. Walls and the pillar are modelled as static markups.

To model the behaviour of the people inside the building we consider the states and transitions shown in Figure 5. After entering a room, a person agent (or a group) moves to a target space inside the room part it entered and holds the position for a period of time. At the end of that period the person agent (or group) decides to either move to the other room part or to exit the room. In the prior case finding the door to that room part and moving through the door will be the next actions. In the latter case exiting and eliminating itself will be the next actions. The transaction arrows inside some of the states provide the person agent (or group) with the capability of updating their destination while in those states under consideration of the movement of the adaptive wall. The double transaction arrow pointing to exiting state is responsible for the different transaction condition of an individual and a group. The adaptive wall does not require a state machine as it is constantly moving under a static restriction. In other words, it stays in a single "working" state throughout the entire simulation runtime and therefore no state machine is needed.

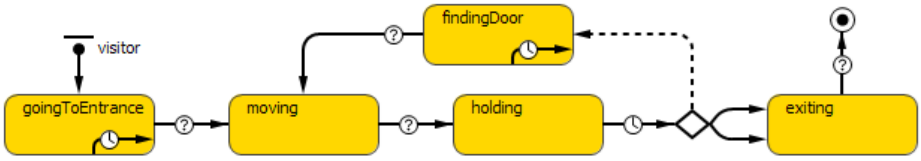

Figure 5 *Behaviour of a "Person" agent for this specific example*

While we were running the simulation we discovered emerging behaviours that we did not expect to see. When there are more people in total inside the room, the adaptive wall seems to move much slower and react milder to density change in both room parts. This is a good example of where emergent properties within the simulation can lead to new insights. An explanation might be that the social force becomes larger as more people are staying inside the room, and the impact of the same amount of people leaving or entering the room will not be as large as if we have an almost empty room. Further experiments will be necessary to confirm this observation and the initial explanation.

4.1.2 Repast Implementation

Repast Simphony is an advanced, free, and open source Java based agent-based modelling and simulation platforms (https://repast.github.io, accessed 1/12/2018). Amongst others, it provides the following support for the implementation of simulation models: rendering, scheduling, continuous space representation, and a statechart library. For everything else we have written our own Java code. At the time we developed the implementation we felt that the decision algorithms for the different types of movable subjects/objects (i.e. the person and adaptive wall) differ quite a bit, although they are both driven by the ESFM. Therefore, our initial approach was to embed the decision algorithm within the classes that require the

decisions, rather than extracting the decision process into a separate class. Some further research is needed to come up with a separate decision class, as proposed in the design pattern. Other than that the Repast implementation is closely aligned with the original design pattern. We are currently in the process of refactoring the model to derive a separate decision class, and therefore get the implementation closer aligned with the original design pattern. The expected big benefit is of course that we avoid code duplication and that we make the software much easier to maintain and extend. Figure 6 shows the current implementation class diagram for Repast.

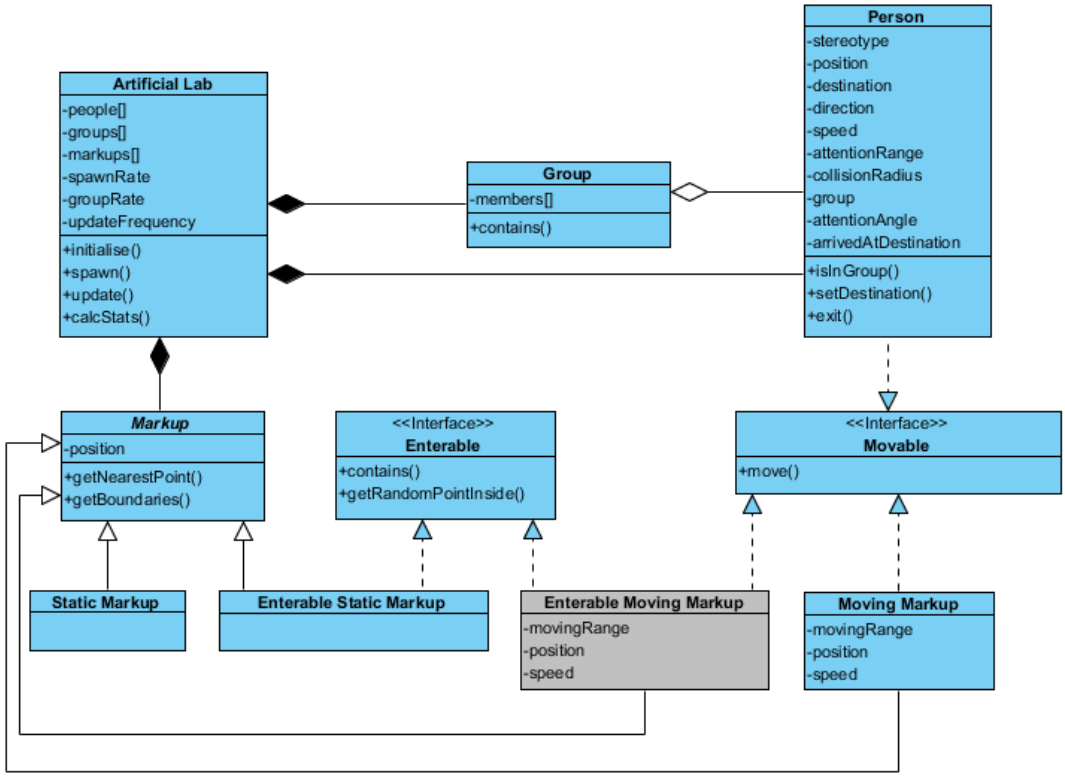

Figure 6 *Implementation class diagram for Repast (blue = our classes; grey = non implemented classes)*

4.1.3 AnyLogic Implementation

AnyLogic (www.anylogic.com, accessed 1/12/2018) is a proprietary multimethod simulation modelling tool, supporting amongst others agent-based modelling. It offers an edition that is free for self-educational and educational purposes. In AnyLogic the "Agent" class found in the package *com.anylogic.engine* is a base class for all agent classes created by the user. It is the main building block of AnyLogic models and can have parameters, variables, ports, events, statecharts and embedded agents and/or agent populations. In addition an abstract "Markup" class is provided in the package *com.anylogic.engine.markup*, including the concrete sub classes to represent static and enterable static markups. We have used these classes as a basis for our implementation. Similar to Repast, AnyLogic also offers support for rendering, scheduling, continuous space representation, and a statechart library.

For "everything that moves" we had to develop our own subclasses that all inherit some base functionality from the "Agent" class. The same is true for the artificial lab. Figure 7 shows the resulting implementation class diagram for our AnyLogic model. As with the Repast we will attempt to make it even closer aligned with the original design pattern in the future (e.g. extracting the SFM/ESFM specific functionality from the "Decision" class).

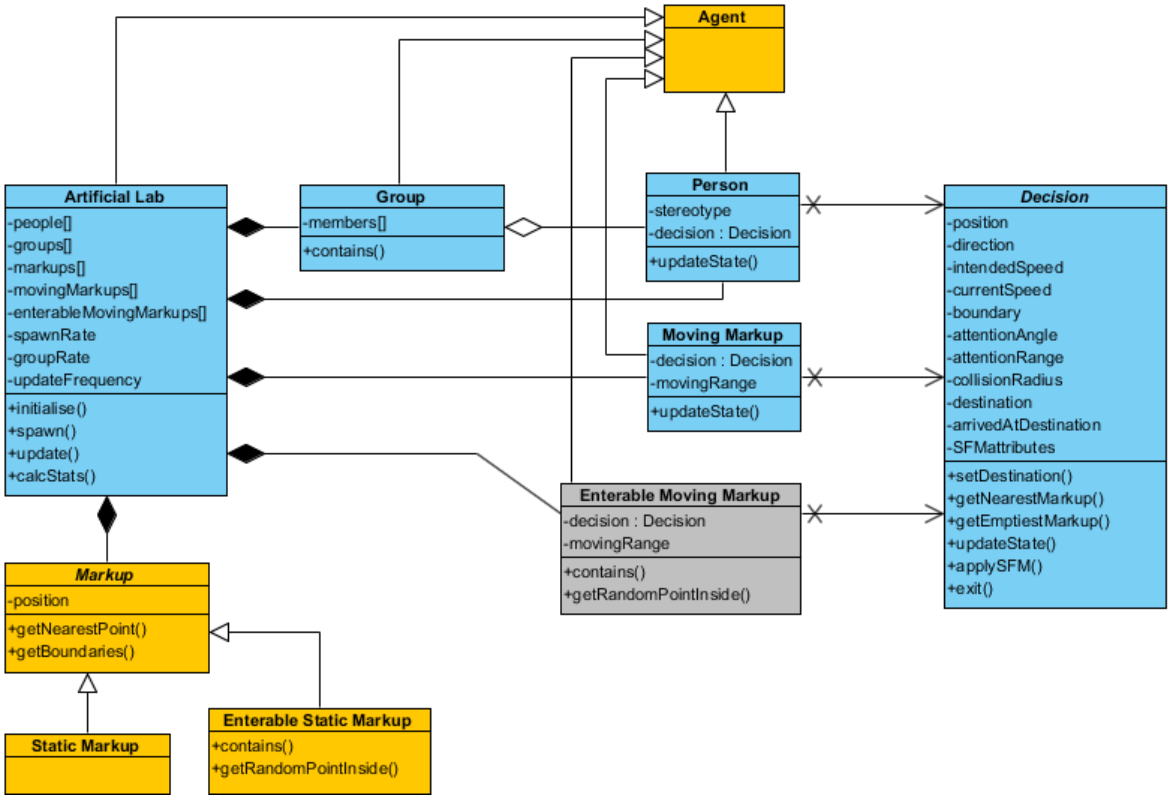

Figure 7 *Implementation class diagram for AnyLogic (orange = AnyLogic classes; blue = our classes; grey = non-implemented classes)*

5 CONCLUSION

In this paper we have presented a design pattern for modelling people's and objects' motion behaviour and for modelling the environments they are moving in. Furthermore we demonstrated its application in two different simulation packages, using an illustrative example from the field of adaptive architecture.

In the future we aim to continue refactoring the current implementations in Repast Simphony and AnyLogic to get them even closer aligned with the original design pattern. We would also like to test the design pattern applicability in other domains. Furthermore, we also consider to provide an extension to the ESFM. We realised that there are some shortcomings, when the ESFM is used to model enclosed spaces. These include things like the "artefact of sudden notice" or the "artefact of visual oblivion". In the ESFM the agents are only aware of what is happening inside their vision area. The "artefact of sudden notice" refers to the situation when an agent changes direction by turning around and a huge crowd suddenly appears in its vision area. Due to the strong force instantly received from the crowd the agent will flee from the current position. The "artefact of visual oblivion" refers to the situation when an agent sees multiple obstacles standing between it and its destination and decides to turn around. If there are no obstacles in the vision area after turning around it will instantly turn back to head toward its destination. This can end in an infinite loop. We produced some ad-hoc solutions to overcome these issues, but we want to see if we can create solutions that are more related to what happens in the real world. We are currently thinking about adding interactions one cannot see (e.g. hearing but perhaps also other senses).

Another big challenge is the calibration of the ESFM for different social and cultural contexts (e.g. a museum visit in comparison to a dance club visit requires very different calibrations. We aim to develop a library of settings for the different contexts.

Overall we believe that what we presented in this paper has opened up many opportunities and we are looking forward to see that others will use the design pattern or the proposed implementations for their own model development.

REFERENCES

Bersini H (2012) UML for ABM. *Journal of Artificial Societies and Social Simulation*, 15(1), p.9.

Booch G, Maksimchuk R, Engle M, Conallen J, Houston K, and Young B (2007) *Object-Oriented Analysis and Design with Applications*. 3e. Addison Wesley

Castle CJE and Crooks AT (2006) *Principles and Concepts of Agent-Based Modelling for Developing Geospatial Simulations*, Working Paper 110, UCL Centre for Advanced Spatial Analysis, London.

Duives DC, Daamen W, and Hoogendoorn SP (2013) State-of-the-Art Crowd Motion Simulation Models. *Transportation Research Part C: Emerging Technologies*, 37, pp.193-209.

Farina F, Fontanelli D, Garulli A, Giannitrapani A, and Prattichizzo D (2017) Walking Ahead: The Headed Social Force Model. *PloS one*, 12(1), p.e0169734.

Helbing D and Molnar P (1995) Social Force Model for Pedestrian Dynamics. *Physical Review E*, 51(5), p.4282.

Helbing D, Farkas I, and Vicsek T (2000). Simulating Dynamical Features of Escape Panic. *Nature*, 407(6803), pp.487-490.

Xi H, Lee S, and Son YJ (2011) An Integrated Pedestrian Behavior Model Based on Extended Decision Field Theory and Social Force Model. In: Rothrock L and Narayanan S (eds.) *Human-In-The-Loop Simulations: Methods and Practice*, London, UK: Springer, pp.69-95.

AUTHOR BIOGRAPHIES

PEER-OLAF SIEBERS is an assistant professor at the Intelligent Modelling and Analysis (IMA) research group in the School of Computer Science, University of Nottingham, UK. He has an interest in object oriented Agent Based Modelling (ooABM) for simulating collective human behaviour. His email address is peer-olaf.siebers@nottingham.ac.uk.

YUFENG DENG is a student, currently studying for a BSc in Computer Science with Artificial Intelligence in the School of Computer Science, University of Nottingham, UK.. His research interest evolves around "machine learning" and how this can be applied in a Social Simulation context. His email address is psyyd1@exmail.nottingham.ac.uk.

JONATHAN THALER is a second year PhD student currently studying for a PhD degree in Computer Science in the School of Computer Science, University of Nottingham, UK. He is investigating how the pure functional programming paradigm can be used to increase the robustness and gain insights into dynamics of agent-based simulations. His email address is jonathan.thaler@nottingham.ac.uk.

HOLGER SCHNÄDELBACH is a Senior Research Fellow at the Mixed Reality Lab (MRL) in the School of Computer Science, University of Nottingham, UK. With a PhD in Architecture and more than 15 years of experience in HCI research, the interface of information technology and the built environment is his core research area. His email address is holger.schnadelbach@nottingham.ac.uk.

ENDER ÖZCAN is an assistant professor at the Automated Scheduling, Optimisation and Planning (ASAP) research group in the School of Computer Science, University of Nottingham, UK. His research interests and activities lie at the interface of Computer Science, Artificial Intelligence and Operational Research. His email address is ender.ozcan@nottingham.ac.uk.