Faculteit Toegepaste Ingenieurswetenschappen
Elektronica-ICT

Improving Scalability of Large Scale Agent-Based Simulations

Proefschrift voorgelegd tot het behalen van de graad van
Doctor in de Toegepaste Ingenieurswetenschappen
aan de Universiteit Antwerpen te verdedigen door

Stig Bosmans

Prof. dr. Peter Hellinckx
Prof. dr. ing. Joachim Denil

Antwerpen, 2021

Jury

Chairman
Prof. dr. Walter Daems

Supervisors
Prof. dr. Peter Hellinckx
Prof. dr. ing. Joachim Denil

Members
dr. Wim Casteels

Prof. dr. ing. Maarten Weyn

Prof. dr. Gabriele D'Angelo

dr. Antonio D. Masegosa

Contact

Ing. Stig Bosmans
University of Antwerp - imec, IDLab - Faculty of Applied Engineering
Sint-Pietersvliet 7, 2000 Antwerp, Belgium
M: stig.bosmans@uantwerpen.be
T: +32 265 17 84

Version: 11th June 2021

Improving Scalability of Large Scale Agent-Based Simulations

Abstract

With the increase of our cities and urban environments being connected to the internet, we see a significant increase in the scale of these, so called, Internet of Things or Smart City environments. These environments are characterized by a large amount of heterogeneous devices and actors. This makes analyzing and testing of novel applications within these environment difficult. This is because these solutions can only be properly evaluated when deployed at the full scale of the environment, taking the full heterogeneity and complexity of the real environment into account. Deploying these solutions untested is too risky. Instead different approaches are required.

To cope with this complexity we see that modeling and simulation techniques can help. More specifically, Agent Based Simulation (ABS) is a paradigm that is well-suited to simulate this type of environments, which consist out of many autonomous devices and actors. These techniques allow the behavior of these environments to be analyzed and evaluated. Furthermore, it allows for the creation of a virtual copy of the environment, which can be used to optimize certain behavior within this environment.
In the last years we have seen examples of this in practice under the name of "Digital Twins". Such a Digital Twin, relies on real-time data originating from Internet-Of-Things devices, combined with simulated environments of its physical counterpart. Many cities utilize digital twins as a representation of the current state of their city. City related properties such as traffic behavior, noise and air pollution can be easily accessed. These virtual environments can even control certain parts of the environment, which could theoretically allow to dynamically optimize the aforementioned properties. But this is currently only possible in theory because modeling and simulation of such environments is strongly limited by its scaling capabilities. This is because they rely in most cases on centralized, monolithic simulation architectures, with statically defined computationally inefficient models.

In this thesis, we tackle the scalability capabilities of large-scale simulated environments. We start with an in-depth analysis of building and testing simulated large-scale Internet of Things environments. Based on this analysis we identify two opportunities. First, we look at optimizing the partitioning of distributed agent-based simulations. In the second we look at abstraction techniques that allow to switch abstraction levels of simulation regions. This enables us to balance between various levels of detail and levels of compu-

tational cost.
Furthermore, we develop a generic methodology that allows dynamic optimization of simulation partitioning and model abstraction levels. We implemented and validated these techniques in a custom developed state-of-the-art traffic simulator, which we rigorously discuss throughout this thesis.

Samenvatting

Onze stedelijke omgevingen worden steeds meer verbonden met het internet, daarnaast zien we een significante stijging van de schaal van deze, zogenaamde, Internet-of-Things of Smart City omgevingen. Dit type van omgevingen kenmerkt zich door een groot aantal heterogene toestellen en actoren die onderling interageren. Deze complexiteit maakt het analyseren en testen van nieuwe applicaties in deze omgevingen moeilijk. Dit komt doordat veel van deze oplossingen enkel goed geëvalueerd kunnen worden wanneer ze worden uitgerold, rekening houdend met de volledige schaal en complexiteit van de echte omgeving. Deze oplossingen ongetest uitrollen is te riskant. Een andere aanpak is daarom noodzakelijk.

Om met deze complexiteit om te gaan zien we dat modellering en simulatietechnieken kunnen helpen. Meer specifiek is agent-based simulatie een paradigma dat goedgeschikt is om dit type omgeving te simuleren. Deze omgevingen bestaan typisch uit een groot aantal autonome toestellen en actoren. agent-based simulatie laat toe om het gedrag van deze omgevingen te analyseren en te evalueren. Verder laat het toe om een virtuele kopie te maken van de omgeving. Dit kan gebruikt worden om bijvoorbeeld bepaald gedrag te optimaliseren of in kaart te brengen.
De voorbije jaren hebben we een voorbeeld van een dergelijk systeem in de praktijk gezien onder de naam "Digital Twins". Zo'n Digital Twin gebruikt real-time data komende van Internet-of-Things toestellen, gecombineerd met een gesimuleerd model van zijn fysieke tegenhanger. Vele steden gebruiken nu al Digital Twins om de huidige staat van hun stad voor te stellen. Stadsgerelateerde eigenschappen zoals verkeer, geluid- en luchtverontreiniging kunnen hierdoor makkelijk geraadpleegd worden. Deze virtuele omgevingen kunnen zelfs controle uitoefenen op bepaalde delen van de omgevingen. Dit zou het theoretisch moeten mogelijk maken om de eerdergenoemde eigenschappen te optimaliseren. Maar dit is momenteel enkel mogelijk in theorie omdat het modeleren en simuleren van zo'n omgevingen sterk gelimiteerd is door zijn schaalbaarheidsmogelijkheden. Dit komt doordat ze veel steunen op centrale monolithische architecturen, waarbij statisch gedefinieerde, computationeel inefficiënte modellen en simulatoren gebruikt worden.

In deze thesis proberen we de schaalbaarheidsmogelijkheden van grootschalige gesimuleerde omgevingen te verbeteren. We beginnen met een diepgaande analyse over het

bouwen en testen van gesimuleerde grootschalige Internet-of-Things omgevingen. Gebaseerd op deze analyse, identificeren we twee opportuniteiten om de schaalbaarheid te verbeteren. Eerst bekijken we hoe we het partitioneren van gedistribueerde agent-based simulaties kunnen verbeteren. Vervolgens onderzoeken we technieken die toelaten om dynamisch te switchen tussen verschillende abstractieniveaus van simulatie regio's. Dit laat ons toe om te balanceren tussen accuraatheid van het gesimuleerde model en de bijhorende computationele kost.
Daarbij presenteren we een generieke methodologie die de dynamische uitvoering van de voornoemde twee methodes toelaat. We hebben deze technieken geïmplementeerd en gevalideerd in een op maat ontwikkelde state-of-the-art verkeerssimulator, die we uitgebreid bespreken doorheen deze thesis.

Preface

My promotor, Peter, said during the course of my PhD that the toughest part of doing a PhD is persisting. Looking back to my PhD trajectory I think this is a very accurate analysis. It felt to me personally like doing a marathon, there are many obstacles along the way, there are ups and downs, and during the downs you need to continuously find ways to refuel your energy levels in order to keep going, in order to persist.

I consider myself very lucky that I always found myself surrounded by people that helped me to refuel this energy. I'm therefore especially grateful to my wife and soulmate Yaiza, who supported me from the start to the finish line. Who gave me the energy, the motivation and the possibility to keep on going. I would also like to thank my parents, Geert and Kristel, whom have always supported me in the decisions I took, whom have supported me to follow my interests and guided me to eventually take the decisions that led to this thesis. Furthermore, I would like to thank my family, Tanguy, 'de dertigers' and the many other friends for the great times we had aside from work which helped me to get the often, so much needed distraction and resulting motivation.

I would like to thank my promotor Peter, for giving me the opportunity to start this PhD trajectory and guide me along the way with many great research advice, but also with invaluable career and life advice that I will continue to take with me. I would like to thank my co-promotor Joachim, for the many great brainstorm sessions we had and the great times we had at the various conferences we travelled to.
I want to thank my colleague Toon to be there both during and after work, to brainstorm daily about the work we were doing and to just get a good cold beer together on the sunny terrace of 'de Coyote' when needed.
Finally, I would like to thank the members of the jury: Walter, Maarten, Wim, Gabriele and Antonio for the great advice and feedback that I received during the course of my PhD.

Antwerp, Belgium
June 2021

Stig Bosmans

Contents

List of Figures

List of Abbreviations

ABM Agent Based Modeling. 3, 19

ABS Agent Based Simulation. i, 27, 60

CPS Cyber Physical Systems. 19

DES Discrete Event Simulation. 9, 39, 43, 46

DTA Dynamic Traffic Assignment. 65

EF Experimental Frame. 56

GSD Global Step Duration. 34, 36

IDM Intelligent Driver Model. 57, 58, 63

IoT Internet Of Things. 1, 15, 30

LCC Local Communication Cost. 32, 36

LE Local Entity. 18

LEs Local Entities. 15, 19, 22

LP Logical Processes. 8

LSD Local Step Duration. 36

MCC Model Computation Cost. 32, 35, 36

MSC Model Synchronisation Cost. 32, 35

MSE Mean Squared Error. 83

NN Neural Networks. 6

OSM Open Street Map. 61

PADS Parallel And Distributed Simulation. 4, 8

PEU Physical Execution Unit. 8

RBF Radial Basis Functions. 6

RCC Remote Communication Cost. 32, 36

RL Reinforcement Learning. 4

RMSE Root Mean Square Error. 70

SEs Simulation Entities. 8, 19, 22

SOM Self Organizing Map. 23

SotA state-of-the-art. 18

STGCN Spatio Temporal Graph ConvolutionalNeural Network. 81

WCT Wall Clock Time. 43

Chapter 1

Introduction

1.1 Problem statement

In this thesis, we aim to improve the scalability of large scale agent-based simulation. We initially discuss the challenges and opportunities in the context of simulation-based testing and evaluation of large-scale smart environments and applications. Later in the thesis, we focus on developing methodologies and techniques to allow dynamic improvement of scalability in state-of-the-art agent-based simulators. We validate our proposed methods on a realistic large-scale traffic simulation scenario. During this thesis, we attempt to solve the following challenges:

- How can we dynamically switch the agent distribution over multiple compute units to reduce computational cost
- How can we dynamically switch abstraction levels of simulation models to reduce computational cost
- What type of simulation architecture is needed to implement dynamic or adaptive performance optimization techniques in a clear and reusable way
- How can we validate the presented techniques in a real-life large-scale traffic simulation scenario

In the next section, we give further context about these challenges that arise when simulating large-scale agent-Based environments and how we will attempt to tackle them.

1.2 Background

Over the last years, we have seen a significant increase in large-scale smart systems. Such a smart system is characterized by its decentralized nature. It consists of hundreds or even thousands of heterogeneous internet-connected entities. Such an entity can be an actor (e.g. a person, or a car), or an Internet Of Things (IoT) device that collects and shares its data. These entities have the capability to either observe or interact with the environment. In the latter case, these entities could alter the state of the environment.

This leads to an enormous opportunity, because when a large amount of these entities interact with each-other and impact the environment, it leads to global, potentially optimized, emerging behavior that can be observed at the system level. This type of behavior, originating from local autonomous interacting entities is often referred to as emergent behavior [1].

A concrete example of this can be found in smart traffic solutions where the environment consists of many observing and acting entities. The acting entities consist of vehicles, cyclists and pedestrians interacting with smart traffic lights, variable speed limit signs, digital traffic signs etc. The observing entities are traffic loops, traffic cameras, sound level devices, air pollution devices etc. Local behavior of individual acting entities could impact the global emergent behavior. For example, switching the traffic light schedules of smart traffic lights could lead to the optimization of emergent properties such as for example traffic flow, noise and air pollution. Other examples of large scale smart environments are smart grids, smart cities, large-scale digital twins and in general large-scale Internet of Things systems.

Analyzing and properly modelling this local behavior is therefore key but not an easy task. Emergent behavior can only be properly evaluated when the local entities have the ability to interact with each-other and with the environment at an appropriate scale. With the rising maturity of Artificial Intelligence systems we could even see some of these entities acting autonomously in the interest of the resulting emergent behavior. Imagine, for example that the traffic lights learn based on the current observable traffic flow how they can collaboratively alter their behavior in order to optimize the emerging traffic flow.
D. Roca et al. [2] argues that the resulting emergent behavior in the context of such large-scale decentralized Internet of Things environments will lead to improved scalability, interoperability and cost efficiency as opposed to traditional approaches that heavily rely on extensive programming of explicit behaviors in a centralized architecture.

We believe that leveraging simulation as a means to analyze the emergent behavior of this type of systems is key. Analyzing and validating both the local and emergent behavior is impractical in real-world test setups because of the required scale. Furthermore, modern learning techniques such as Reinforcement Learning require a large amount of interactions of the agents with their environment. This makes simulation also in the optimization context a key component.
However, as mentioned, the type of smart systems discussed in the previous paragraphs are characterized by their large-scale requirements. The scale of these systems makes computational efficiency a primary condition. In this thesis, we propose various techniques and methods to improve the computational efficiency of large-scale simulation frameworks in the context of analyzing, testing and optimizing large-scale smart systems.

The computational requirements of this type of simulations are directly dependant on the amount of entities and the complexity of their models. We saw that spreading the amount of entities over more computational units (CPUs and servers) could reduce the total execution time at the cost of increased simulator complexity and increased synchronization requirements. But as stated by Amdahl's law [3] the scaling possibilities by relying on increased concurrency are limited by the additional overhead it creates and

the parts of the application that require serial, non-parallel execution. Instead, the complexity of the models could be altered based on the accuracy needs of the domain in order to reduce the computational cost. The focus of this thesis is on exploiting these observations to improve the computation efficiency. Furthermore, we identified that many of the computational requirements and constraints of the simulations strongly depend on the context of the simulation. For example, the required model complexity can change over time and based on the operating context of the model. Based on this observation, we put extra focus on optimizing the computation efficiency of the simulation dynamically by taking the simulation context into account.

Throughout the thesis we will refer to the type of system described in this section as Large-Scale Smart Systems, or simply "Smart Systems". In our work, we focus primarily on smart traffic system use cases, as we believe that this is representative for the challenges that arise when simulating smart systems in general, and where emerging behavior originates from the interactions and behavior of local decentralized entities.
Internet-of-Things is a concept that is strongly related to these smart systems, therefore we also put some extra attention on simulating and testing general Internet-of-Things systems. This is mostly discussed in chapter 2. The remaining chapters primarily focus on optimizing the computational efficiency of large-scale traffic simulations.
In the next sections, we look into further detail what methods exist to simulate smart systems and we discuss relevant techniques such as Parallel and Distributed Simulation and Model abstraction. The analysis presented in the next sections is based on a positional paper that was published and presented in the SCS SpringSim conference of 2018 [4].

1.2.1 Model representation

In order to model emergent behavior, the Agent Based Modeling (ABM) paradigm seems the most appropriate [5]. With ABM, a bottom-up modeling approach is applied. Instead of modeling the global expected behavior, the modeler describes the behavior of the individuals. The ABM paradigm allows these individuals to interact. Eventually, these interactions will lead to a global behavior. As the smart system application are deployed in a similar matter, the simulation engineer has to model the different entities involved in the simulation.

Opposed to ABM there are other formalisms often used in modern simulation applications. For example, Discrete-event formalisms such as DEVS [6] model the behavior of a system using a timed sequence of "events" either as input to a system or as a timeout within the system. These events cause instantaneous changes to the state of the system. Atomic discrete event models can be combined together into a coupled model. Furthermore, extensions exist that couple these models in a grid, e.g. Cell-DEVS [7]. These types of cellular automata can have an advantage to model geographical areas. E.g. the city is divided into a grid of cells where each cell models a part of the city.

Finally, differential equations can help to model rates of changes of relevant properties, e.g. traffic flows can be modeled with a differential equation. Different equations can also be coupled together using co-simulation techniques to allow for a divide-and-conquer approach to model complex systems.

The focus in this thesis is on the Agent-based formalism, because it is ideally to simulate emergent behavior by modeling the behavior of individual entities, which makes modelling of these smart systems easier. Furthermore, the formalism is related to how modern optimization applications, such as Reinforcement Learning (RL), are developed. The simulations developed using the ABM formalism can therefore be easily integrated in an RL environment. We will also look into exploiting other formalisms because of performance reasons and integrate them in the same simulation environment.

1.3 Overview

The main challenge in simulating large-scale smart systems is related to performance. In order to obtain some level of representative emergent behavior a great amount of interacting entities is required. Running such a simulation on a single computational unit is often not enough, instead distributing to multiple servers is a better approach to cope with the necessary scale. Example implementations are described in the Parallel And Distributed Simulation (PADS) methodology [8] and in the IEEE 1516 standard. Facilitating the proper execution of these simulation entities in a simulated environment still requires several performance optimization techniques in the simulation core. Preferably, these optimizations can be performed transparent to the simulation modeler. However, this won't be possible all the time. In these sections we provide an overview of various optimization techniques that can be applied dynamically or statically. A dynamic optimization can be applied at run-time and can be context-dependent, while a static optimization needs to be applied up-front. We made a break-out of major performance optimization techniques in Figure 1.1 below. We used a feature diagram formalism to represent the various techniques. We refer to this diagram throughout the thesis, to clarify in which area we make contributions.

1.4 Model abstraction techniques

An important aspect of generating emergent behavior is to properly describe the behavior of the individual elements of the real-world within the simulation model. Furthermore, once these systems are modeled we show which modeling techniques can help us in achieving a scalable smart system simulation framework.

The most appropriate formalism often depends on the required level of detail. Model abstraction techniques can help to reduce the level of detail when possible. This leads to a decrease of the computational cost of individual models by "simplifying" the representation of such a model while maintaining the original behavior as accurate as possible. The main objective of applying model abstraction is to make a proper trade-off between computational complexity and accuracy. Extensive taxonomies of model abstraction techniques exist in literature [9], [10]. In this work we will only discuss the most relevant techniques. As demonstrated in Figure 1.1, we differentiate between two major categories, one based on multi-formalism model abstraction and another based on single-formalism abstraction. We than discuss how we can use these techniques in a smart system simulation environment.

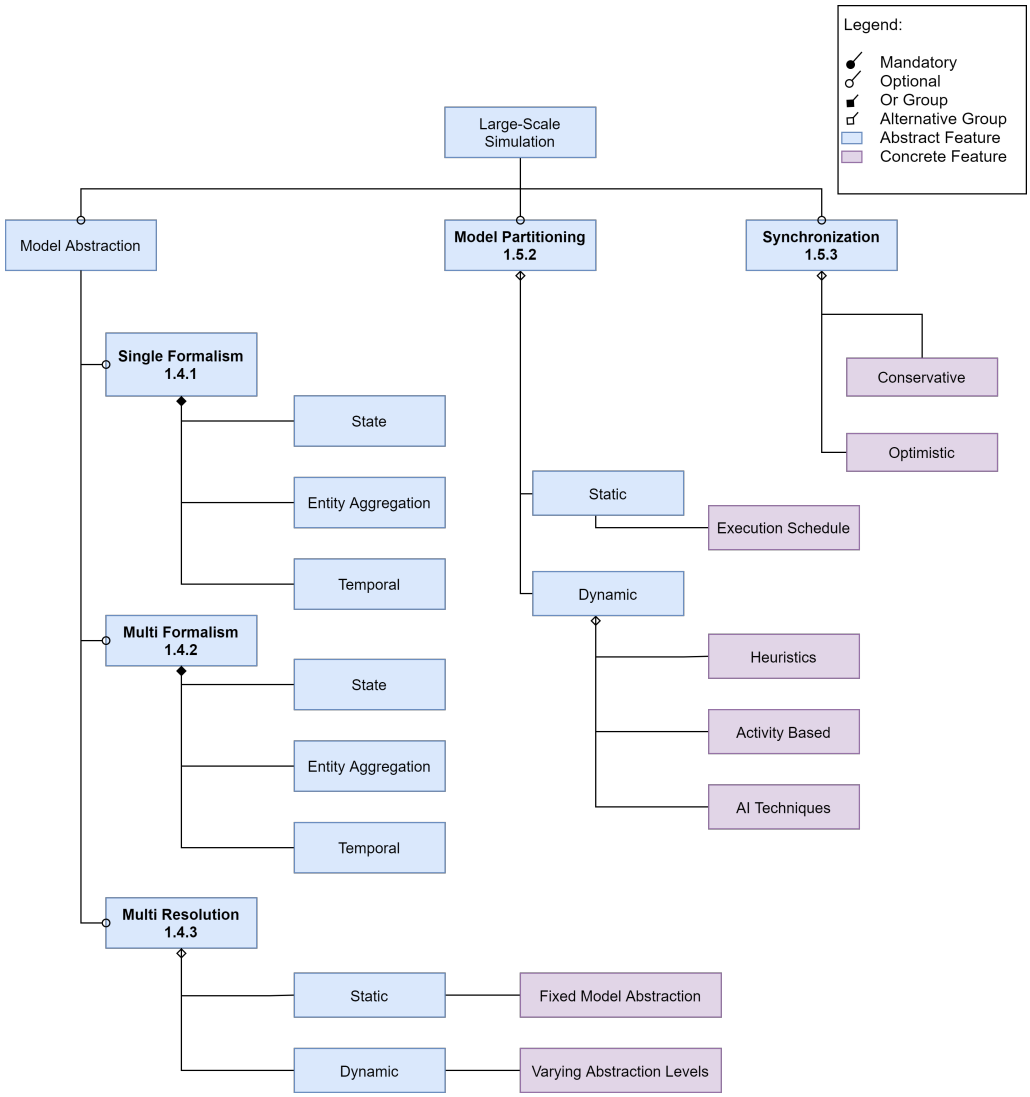

Figure 1.1: Overview of relevant simulation performance optimization techniques

1.4.1 Abstractions within a single formalism

These abstractions are applied on the model without changing the formalism to describe it. We define three types of abstractions that can happen within a single model. We will apply these types of abstractions on the individuals of the ABM but they can also be applied to models in other formalisms as well.

State abstraction This technique abstracts the state of a single individual. As the decision processes of individuals can be very complex, certain parts of the decision making process that an individual exhibits can be ignored without compromising the global

behavior of the simulation, e.g. the individual also reasons about other properties that do not influence the global behavior, or higher-order reasoning with little to no impact on the decision of the agent.

The process of creating a more abstract individual from a detailed one can be done manually. This requires insight into both the decision making process of the individuals as into the application that generates the emergence of behavior. Therefore, more automatic techniques might be more appropriate. An example technique that can be used for this is metamodeling (as in surrogate modeling). Caughlin et al. define a metamodel as a projection of the original, high-fidelity model onto a subspace defined by new constraints or regions of interest [10]. In practice, a metamodel is a mathematical approximation of a complex model. The original model is treated as a black box, and the metamodel operates as a surrogate model that replaces the original. To come to such a metamodel a detailed analysis of the input/output mapping of the original model has to be performed. Based on this analysis a surrogate metamodel should learn to represent a similar mapping. Preferably, the surrogate model is more abstract and is computationally less complex. Various methods have been used for the development of metamodels such as polynomial regressions, Radial Basis Functions (RBF) and others [11]. Most of these techniques try to approximate input functions or data. Since the goal of metamodeling is to map an input value to a specific output value we believe other supervised learning techniques could be used as well. For example, P. Symonds et al. demonstrate that Neural Networks (NN) could perform up to 15% more accurate compared to classic RBF approaches [12] [13]. We believe that evaluating various state-of-the-art deep neural networks could lead to even better results. We consider this as a promising area for further research.

Entity aggregation Another option to reduce the computational complexity of simulation entities is Entity Aggregation. The idea of entity aggregation is to combine multiple low-level simulation entities with a single high-level entity while preserving the collective behavior. For example, when simulating emergent behavior in a smart power grid application, a single agent could represent a single household but depending on the required level of accuracy a single agent could also represent an entire neighborhood. Rodriguez et al. demonstrate that NN's could be used to transform a collection of low-level entities to an aggregated model [14] . From an architectural perspective, a mechanism in the simulation kernel can be created to detect and analyze clusters of agents that interact a lot and pose homogeneous behavior. Using metamodeling techniques such a cluster of agents can then be replaced by a single surrogate model. Conversely, the modeler can also manually model an aggregation of individual simulation agents.

Temporal abstraction Finally, temporal abstraction can be used to limit the granularity of simulation entity state updates over time. Since, each state update results in a cascade of simulation events the overall computational cost could be significantly reduced.

Engineers can themselves easily change the temporal granularity (time-step) of their simulation. However, choosing a too large time-step can result in an unstable simulation. Automatic techniques to adapt the time-step also exist [15]. E.g. for solving differential equations, several techniques are available that automatically adapts the time-step of the simulation based on the estimated error, e.g. Runge-Kutta 4-5 embeds a higher-order method in its solver to estimate the error and adapt the time-step.

1.4.2 Multi-formalism modeling

Changing formalisms during abstraction can help to achieve better performance results as we can leverage the strengths of different types of formalisms. For example, as explained in section 4.1, in the domain of traffic simulation, queuing theory might be a more appropriate, less computationally expensive formalism compared to agent based modeling. The formalism change allows for an easier state and aggregation abstraction.

1.4.3 Multi-resolution modeling

Multi-resolution modeling can be used to vary between model abstraction levels (and possibly formalisms) within the same simulation. The advantage of this is that we can increase abstraction levels in simulation areas where less accuracy is required. This will consequently reduce the computational complexity of the simulation. Oppositely, we could also increase levels of detail in certain simulation areas when an increased level of accuracy is required. The multi-resolution model can be static or dynamic.

When applying a static approach, the most appropriate formalisms and levels of detail have to be specified upfront and will be used during the entire course of the simulation. Due to the dynamic nature of smart system applications, the potential computational benefit is limited. Conversely, with a dynamic approach we can switch between various levels of abstraction during the course of the simulation. This can be done by predefining when and where an abstraction switch should occur. Domain knowledge is necessary to detect and abstract/refine the (part of the) model.

Learning approaches are also possible to switch levels of abstraction, by learning areas of opportunity and applying the necessary level of abstraction. For this, additional mechanisms in the simulation kernel are required that detect areas of opportunity based on various parameters. For example, simulation entities that have a limited amount of interaction with the rest of the simulation could be marked as candidates for an abstraction level increase. The advantage of such an approach is that we can adaptively detect opportunities to switch between abstraction levels in order to decrease necessary computational resources or oppositely, use all available computational resources.

Applying a dynamic multi-resolution modeling approach leads to additional challenges that need to be taken into account [16]. For example, reinitializing a more abstract model from the current state of the original model is possible as there is enough information available in the original. However, when switching from a more abstract to a refined model, extra information is required. This should be mended by modeling additional domain-knowledge to fill this information gap.
In chapter 4, we propose a mix of entity aggregation and multi-formalism modeling, i.e. dynamically switching from an agent-based formalism responsible for modeling behavior of a single entity to a discrete event formalism responsible for modeling multiple entities. In chapter 5, we expand on this work with a framework that allows to switch between multiple resolutions and formalisms to balance both computational cost and validity of the global emergent behavior. In this chapter we also discuss a method that allows dynamic switching of model abstraction levels. Finally, in chapter 6, we discuss a first attempt to combine traffic simulation with data-driven models that where the behavior was not explicitly described but learned from data.

1.5 Techniques related to simulation architecture

In this section, we will discuss techniques that could improve the performance capabilities of the simulator itself. Most of these techniques have been studied in various domains such as parallel and distributed simulation (PADS) and distributed discrete event simulation.

1.5.1 Simulation architecture

Most of the current Internet of Things and traffic simulators have an underlying monolithic architecture which limits its scalability capabilities. D'Angelo et al. recognize this limitation in state-of-the-art IoT simulation in their work. Instead, they propose their custom built large-scale simulator Gaia/Artis which is based on the PADS paradigm. PADS allows a simulation to be executed among multiple distributed devices or Physical Execution Units (PEU's), which will significantly improve scaling capabilities. Each Physical Execution Unit (PEU) consists of a collection of Logical Processes (LP). An LP represents a part of the simulation and contains a collection of Simulation Entities (SEs). A simulation entity, as the name implies, represents an individual simulation model or agent (in the context of agent-based simulation). A disadvantage of PADS and distributed simulation is that it leads to additional difficulties with regards to synchronization, simulation partitioning and transparency.

1.5.2 Model partitioning

Model partitioning aims to optimally distribute agents across multiple servers to decrease the overall communication cost. A significant part of the computational interaction cost is characterized by remote communication across multiple PEU's [17]. This computational cost of remote communication is much higher than the local communication cost between SE's located in the same region. Much efficiency can be gained by analyzing communication patterns and optimizing the distribution of the simulation entities, so that local communication is maximized and as a result the communication cost will be lower.

Static model partitioning: We make a distinction between a static and a dynamic approach. With a static partitioning approach it is assumed that the interaction dynamics between SEs remain stationary, as soon as these dynamics change over the course of the simulation the benefit of the partitioning decreases. For example, a fixed partitioning could be configured by specifying the exact time of when a model entity should migrate from one LP to another. Such a schedule could be performing well at first, but as soon as the communication dynamics change it will lead to suboptimal simulator performance [18]. In the domain of IoT and traffic we can assume the presence of stochastic dynamic nodes and devices. This will inherently lead to unpredictable changes in interaction dynamics. Furthermore, because of the required scale to simulate emergent behavior we can conclude that static partitioning methods are impractical and not preferable in order to simulate this type of application [19].

Dynamic model partitioning using heuristics: When compared to static partitioning, the dynamic partitioning methods will be more adequate, as they are able to automatically migrate simulation entities when their communication patterns evolve. In

the context of PADS, many research has been done in the area of dynamic simulation partitioning. Most of these techniques rely on solutions based on heuristics [20]. In most cases these heuristics don't have a complete view on the global simulation state because this would be computationally too expensive, both from a model computation and communication point of view. Instead, most heuristics rely on limited data directly available on the individual hosts.

Dynamic model partitioning using domain knowledge: A possible disadvantage in the heuristics approach is that it doesn't take domain knowledge into account. This can lead to a number of undesired side effects. For example, migrations might occur that need to be undone in a later phase. This could result in a higher computational cost (introduced by these consecutive migrations) than the obtained gain which was only temporal. The heuristics presented above do try to prevent oscillating migrations by introducing a threshold that prevents immediate undoing a migration. However, another solution would be to inject domain knowledge that could prevent these unwanted migrations to occur in the first place. Furthermore, it could perform more optimal migrations that couldn't have been achieved by solely analyzing local communication patterns of individual SEs.

Van Tendeloo and Vangheluwe demonstrate in their work that significant performance improvements can be obtained by injecting domain knowledge in their Python-based distributed DEVS simulator PythonPDEVS [18]. Their work builds further on the abstract notion of activity introduced by Muzy et al. which represents a measure for a number of events in the context of a Discrete Event Simulation (DES) system [21]. This activity concept can refer to various resources such as time, memory or energy which are relevant in a DES system. In the context of performance optimization the time resource will be most relevant. One can look at activity from various perspectives, either from a computational load perspective or from a communication perspective. The heuristics discussed in the previous paragraph look at activity from a communication perspective whereas activity-based on the computational perspective. The concept of activity prediction allows simulation models to provide hints to the simulator kernel about both their current and anticipated activity and how they should be distributed. Consequently, these hints will be exploited to decide when a SE migration should occur. This approach leads to a more optimal distribution of computational load across servers. The idea of activity prediction based on domain knowledge is also relevant when looking at the activity concept from a communicational load perspective, in that case, the goal would be to reduce the overall communicational load over the network. A disadvantage of this approach is that a level of transparency has to be sacrificed, breaking down the boundary between model and simulation engine in order to obtain better performance.
In chapter 3, we present a generic and reusable method for dynamic/adaptive partitioning by leveraging the concept activity in order to solve load imbalances in a distributed simulation setup. Our method tries to manage the breaking of the boundary between the model layer and the simulation engine layer by presenting a clear and generic framework that allows information to be shared between the various layers within the simulation architecture. We validate this technique on two agent-based use cases.

1.5.3 Synchronization

Finally, another important decision that needs to be considered when developing a large-scale distributed simulator is synchronization. This section gives an overview of the most relevant techniques described in state-of-the-art research.

Synchronization is an important topic in the field of distributed simulation. Its goal is to maintain validity of a distributed simulation system by preventing causal inconsistencies. Such an inconsistency occurs when events that depend on each-other are executed in the wrong order [22]. For example, imagine an event B that depends on the results of an event A. If event B were to be processed before the execution of event A, a causal inconsistency occurs and simulation results could become invalid. Various synchronization techniques are discussed in literature to maintain this causal consistency, a distinction is made between two major categories 1) optimistic synchronization techniques and 2) conservative optimization techniques. Each technique will lead to different performance results.

Conservative synchronization One way to maintain causal consistency among various LPs in a distributed simulation is to prevent a simulator to move to the next event only when it is certain that no other LP will insert an earlier event. [23]. This approach is called a conservative synchronization approach. The best known conservative techniques are based on the Chandy Misra Bryant algorithm [24]. In their work, each LP should have an individual communication channel for all other LP's it communicates with. Each LP is assumed to post timestamped events in the right order. An LP is only able to accept and process events with the lowest time-stamp as soon as all messages in all channels are received. This could lead to deadlocks when some LPs are not generating events, leading to some empty channels and an LP consequently remaining in a waiting/blocking state. As a solution, the idea of null messages is added, where null messages can be sent to channels when no events are assumed for a particular LP. The CMB technique allows the simulation to progress at various rates, however it introduces high overhead because of the null messages in terms of computational overhead and network load [25].

Optimistic synchronization The conservative synchronization method blocks the simulator and results in some LPs waiting in an idle state on other, slower LPs, which is not always necessary. Instead, the optimistic synchronization method introduces a non-blocking approach where each LP continues on its own rate, not waiting for other LPs until an inconsistency occurs, e.g. an event is received with a time-stamp earlier than the local time of the LP. When such a inconsistency is found the simulator will turn back its state to a checkpoint prior to the time-stamp of the received event before continuing processing. As a result, the simulation remains valid. This non-blocking optimistic synchronization mechanism as described above, is called the time warp protocol, first introduced in Jefferon's paper [26]. In cases where not many causality conflicts are expected, it could lead to better simulation performance and a reduced execution time compared to the conservative approach.

The synchronization solutions are a out of the scope of this thesis and will not be discussed further.

1.5.4 Scalability

Throughout this thesis we refer to the concept of scalability and how we aim to improve it. It is therefore important that we clearly define this concept as meanings may vary between various research domains. We define the concept of simulation scalability from the perspective of the simulation execution time, where an improved scalability is equal to an overall improved performance of the simulation execution time. That is the time interval between the end of the simulation and the beginning of the simulation.
This can be in a single-core computational environment but also in a multi-core or distributed environment.
There are no clear benchmarks available that we can leverage to compare the improvement of the scalability with. This is because the execution time of a simulation strongly depends on the implementation and level of detail added to the simulation models. And this is very application dependent.
Therefore, in the experiments presented in this thesis we compare the impact of our presented methods on state-of-the-art models executed in the same simulation framework. This simulation framework is called Acsim, and has been developed throughout the course of this thesis. The architecture of this framework is also implemented based on state-of-the-art frameworks and literature. We describe the implementation and the design decisions of the framework throughout this thesis.

1.6 Contributions

In the previous section, we presented an extensive background to the problem of simulating and evaluating large scale Internet of Things environments. In this research project we aim to analyse techniques and methods to test and validate large-scale smart systems using agent-based simulation and modeling.

Furthermore, we identified possible solutions discussed in literature that allow to improve the scalability of such complex environments: Model abstraction and Model partitioning. We identified the problem that most of these solutions rely on static implementations. This can however lead to inefficiencies occurring at simulation run-time. Therefore, in this thesis we propose a methodology to dynamically switch model abstraction levels and dynamically optimize agent distribution in the context of distributed agent-based simulation.
We validate our method with implementations in a state-of-the-art agent-based traffic simulator. We consider traffic simulation as the perfect validation use-case of the methods presented in this thesis. This is because traffic simulation is often used in smart city solutions. It requires a high amount of complexity and scale. It leads to emergent behavior that can best be validated at appropriate scale. Therefore, we believe that the methods presented in the context of traffic simulation can also be used in simulation of other large-scale applications. We compare each of the dynamic approaches to a static baseline to evaluate its performance.

The contributions of this thesis can be summarized as such:

- Overview of challenges and solutions to simulating large-scale smart environments and Internet-of-Things systems in general
- Method for dynamic partitioning of agent-based simulations

- Method for dynamic abstraction of agent-based traffic simulations
- The implementation of these methods on a state-of-the-art large-scale traffic simulation framework

The contributions made in this thesis are shown in the feature diagram below. This feature diagram is a selection of the features presented in Figure 1.1. Our contributions are mostly related to multi-formalism abstraction where we abstract a collection of concrete agents and replace them by a more abstract formalism. Furthermore, we always rely on a conservative time stepped synchronization. This is because we want to have the full simulation synchronized at each discrete timestep. With opportunistic synchronization some of the simulation areas could be running behind or ahead compared to the rest of the simulation. Because of these reasons we didn't research synchronization methods further. A lot of research has already be done in this area, and we consider it to be outside of the scope of this thesis. All chapters in this thesis implement the same step-based conservative synchronization method.

Our main contribution of this thesis to the state-of-the-art is not only in the various fields we show in Figure 1.2 but also more general: we look broadly to the domain of scalable simulation and we present a method that allows adaptive behavior of the simulation engine to better balance performance and accuracy. Such adaptive method, as presented in this thesis, have not been described in literature. Our method goes beyond the state-of-the-art by the facts that it generalizes among various agent-based simulation applications and that it can be applied both in the domain of model abstraction as in the domain of distributed load balancing. Furthermore, we validate this research using a custom developed large scale traffic simulation framework which is further discussed in the next paragraph. This allows us to validate our method beyond small-scale toy examples as is often the case in related work. The combination of these contributions make this thesis unique and offer significant contributions to the state-of-the-art in the domain of large-scale agent-based simulation and modeling.

Acsim simulation framework Throughout this thesis we discuss the architecture of the Acsim simulator. Acsim is the name of the simulation framework that we developed during the course of this research. Each chapter might reintroduce parts of the simulation architecture, but always from a slightly different perspective. We used the framework to validate our experiments and the various techniques that we present. It has grown as an important contribution of this thesis. It started as a very basic simulator and eventually grew to a state-of-the-art simulator. We hope that the framework will continue to add value beyond the scope of this dissertation and that it will be used in related research trajectories. It must be noted that the implementation details of the simulation framework our outside the scope of this thesis as they don't offer a direct research contribution. A major amount of work during the executions of this thesis was however focused on the implementation of this simulation framework. There are no similar frameworks available, so we had to build this simulation framework from scratch in order to test and validate the methods proposed in this thesis.

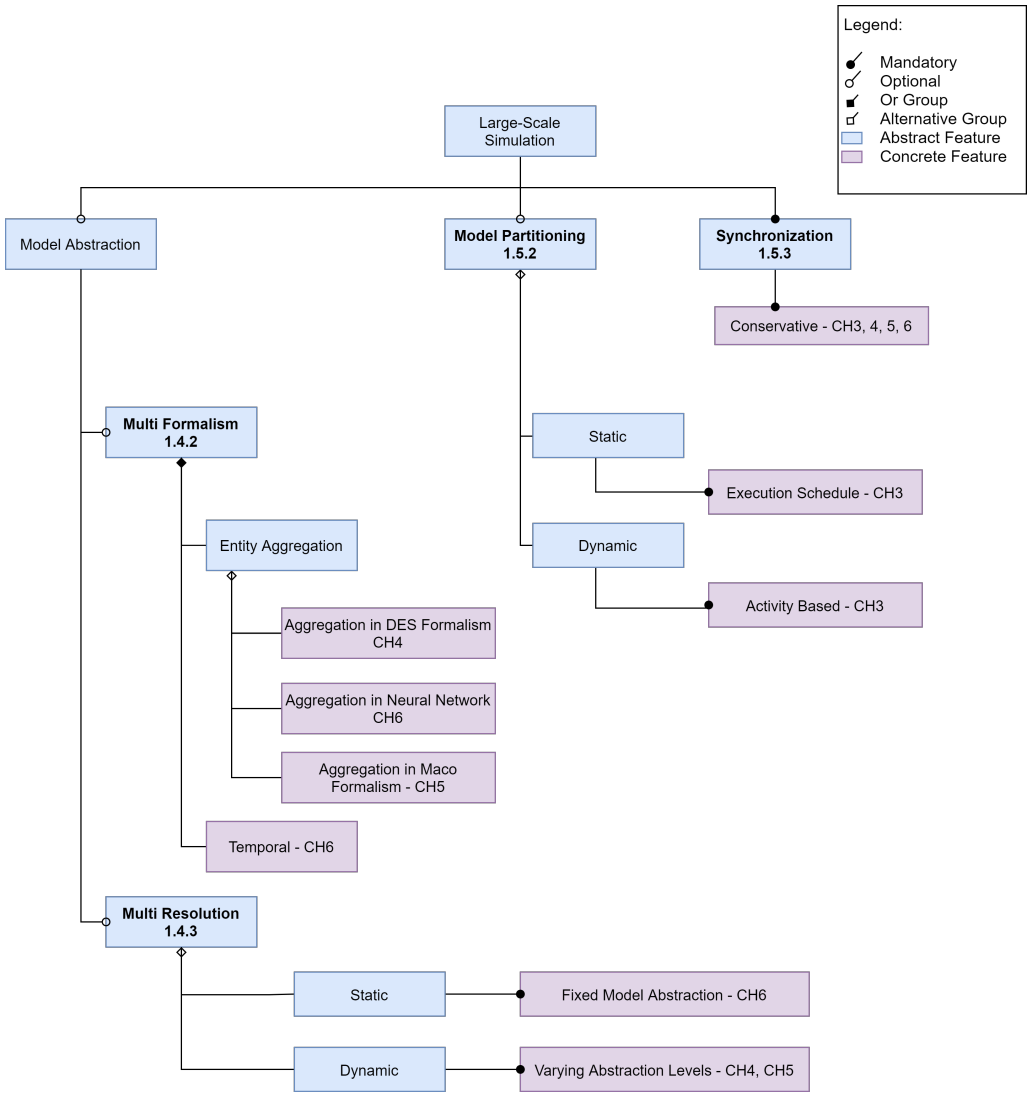

Figure 1.2: The various features and contributions of the thesis

1.7 Outline

This thesis is presented as a collection of bundled papers that were published the last four years. Each chapter corresponds to a single paper.

In chapter 2, we present a framework that enables hybrid testing of Internet of Things environments. We discuss the implementation of this framework using a participatory sensing use case. We further align some problems that arise with regards to synchronization and scalability. this chapter was published as *Testing IoT systems using a hybrid simulation based testing approach*, in the 2019 Springer Journal of Computing [27].

In chapter 3, we discuss techniques to adaptively distribute agents among multiple computation units. We present the MAPE-K framework as part of our simulation architecture to allow for this adaptivity. We validate our method on two use cases. A realistic micro-traffic use case, where we dynamically migrate parts of the traffic network. And a sugarscape automate where we migrate agents between CPU units to reduce the remote communication cost. This chapter was published as *Adaptivity in Distributed Agent-Based Simulation: A Generic Load-Balancing Approach*, in the 2020 International Workshop on Multi-Agent Systems and Agent-Based Simulation as part of the AAMAS conference [28].

In chapter 4, we present a technique to approximate model behavior on an agent-based micro-traffic use case. We dynamically identify approximation opportunities by measuring the entropy of speed distributions at the street level in order to approximate the entire agent-based street-level behavior. This chapter was published as *Reducing computational cost of large-scale simulations using opportunistic model approximation*, in the 2019 Spring Simulation Conference [29]

In chapter 5, we discuss a state-of-the-art large-scale microtraffic implementation and leverage the MAPE-K paradigm to dynamically switch model formalisms and reduce computational cost while taking model accuracy into account. We allow switching between micro and meso-level traffic models. This chapter is submitted for publication as *Adaptivity in multi-level traffic simulation* at the 2021 Elsevier journal of simulation modeling practice and theory.

In chapter 6, we discuss the implementation of data-driven model based on a novel graph convolution architecture within our micro-traffic simulation framework. We show how it can more effectively approximate driving behavior while taking the layout of the traffic network into account. This chapter is a progress report and will be submitted later this year to a yet undetermined conference.

In Appendix B, we present a publication that describes the early state of the Acsim simulator architecture. This work was published in 2017 [30]. Finally, in Appendix C we present an expansion of my master thesis related to scheduling of computationally intensive calculations in a cloud environment [31]. This work was published in 2018 in the International Journal of Grid and Utility Computing.

Chapter 2

Testing Internet of Things Environments using a hybrid testing approach

As mentioned in the introduction of this thesis, Large-scale Smart Systems are closely related to IoT systems. In this chapter, we focus on the challenges that arise when testing IoT applications using simulation and how emergent behavior relates to this type of challenges. This chapter doesn't contain the main contributions of this thesis, but it makes a strong case on testing of IoT systems using large-scale agent-based simulation techniques which is relevant in the scope of this work. This chapter is based on the work that was published in 2019 in the journal of Computing [27].
The interactions of local entities, such as IoT devices or people interacting with the IoT system, eventually leads to an emergent behavior. Both the emergent behavior and the local behavior need to be taken into account when testing IoT systems. Therefore, we present a hybrid simulation-based testing approach that is able to facilitate both high scalability for emergent behavior testing with real life on device testing. Furthermore, we introduce various solutions to the challenges that arise when implementing this hybrid method. These challenges are mainly related to the IoT development pipeline, synchronization between real-life and simulation environment and the scalability constraints of modern simulation techniques.

A very important aspect when testing Internet of Things systems at the system level is to include the behavior exhibited by Local Entities (LEs) that leads to emergent behavior. In order to orchestrate such LE behavior, the LE will need to be able to interact in real-time with the environment and with other entities. This eventually leads to a global, emergent behavior. Emergent behavior is the overall, global behavior posed by a smart system. It inherently depends on the behavior and interactions of LEs with each other and with the environment. Mataric et al. define emergent behavior as a collection of actions and patterns that result from local interactions between elements and their environment, which have not been explicitly programmed. When tested and calibrated correctly this emergent behavior can lead to various global optimizations at the system level. However, a large amount of real-time interacting LEs, posing realistic behavior will be required to properly test this.

In the next section, we present the participatory sensing use case that serves as a running example throughout the chapter. In section 3, an overview of classic IoT testing techniques is provided and the importance of local and emergent behavior is clarified. In section 4, we present various techniques that can be used to simulate behavior of local entities. Section 5 introduces the hybrid simulation-based testing methodology and proposes various solutions to implementing this methodology. Finally, section 6 takes a deeper look at scalability constraints that arise when implementing large real-time IoT simulations.

2.1 Participatory sensing use case

To better explain the concepts presented in this chapter we will use a running example throughout the remainder of this chapter. The running example is based on the SeRGIo project, which is short for mobile sensing services for developing geospatial IoT application. It is a participatory sensing project that is currently being developed with multiple academic and industrial partners as a proof of concept. The goal of the project is to collect qualitative data from citizens or workers, such as mail(wo)men, by targeted distribution of small sensing tasks. SeRGIo is different compared to other participatory sensing frameworks in that it focus on the collection of qualitative data instead of quantitative data. An example of such data is the user's perception of safety in a particular neighborhood. A sensing task is typically received on the smartphone of the user and contains a small questionnaire or single question such as "How clean is the street?" or "What is the quality of the street bench in front of you?". SeRGIo will also take the spatio-temporal aspects of a user into account when sending tasks. This means that the system only sends tasks related to a certain place or area when a user is actually nearby.

It should be clear that testing this type of system poses significant challenges. This is mainly due to the fact that the functional aspects of the IoT system can only be tested with actual participants. In current state-of-the-art, there isn't a clear approach to handle such test cases.

2.2 IoT testing

IoT applications are inherently heterogeneous systems. They consist of many different sensors, actuators, communication protocols and operating systems. This makes the testing of these systems a challenging task. In this section, we provide an overview of classical testing techniques used in IoT projects and identify the gaps that remain in current testing practices.

2.2.1 Classic testing of IoT

Given, that there is a strong cohesion between hardware and software in IoT projects, the testing approach used in state-of-the-art Internet of Things projects is often based on best practices used in classic software or hardware development.

Software testing Software testing is mainly used for the validation of both functional and non-functional aspects of IoT middleware software or more low-level code such as the

software logic deployed on IoT sensors and actuators [32]. Two major types of testing are described:

1) White box testing: the tests have full transparency to the inner structure of the software. Unit tests are often considered a white box testing approach. Within the context of IoT testing, this method is used to test the functionality of low-level pieces of code, such as single methods and classes [33].

2) Black box testing: With this testing method the software system is considered a full black box [34]. These tests have no knowledge of the internal structure of the system. Black box tests are concerned with more high-level aspects of the software, typically on the system-level. E.g. in SeRGIo we could apply a black box system level test to verify whether a user correctly receives a sensing task on his smartphone when he is located in a certain area that is below the data relevance threshold. In practice, these type of tests are difficult to run, especially when developing IoT applications. Because, as mentioned before, IoT systems typically contain many different sensors, actuators and software parts interacting with each other.

Within the context of this chapter we only focus on the black box testing method.

Prototype setups Prototype testing is used to test the functionality of hardware components in a lab-based environment. These tests are often limited to single devices. Given the diversity of IoT systems and environments it is best to rely on more large scale test setups when testing entire systems. Various initiatives are described in literature to facilitate such large scale test beds. One example is the imec City of Things testbed in Antwerp [35]. Another example of such a testbed is the Smart Santander project [36]. It is a city-wide, real IoT testbed. It offers many thousands of IoT devices readily deployed in an actual environment. Most of the testbeds described in literature are ideal to test non-functional requirements of an IoT system, such as testing the interoperability between various devices and their communication protocols or operating systems.

Simulation based testing Instead of writing custom test cases, simulation models are used to interact with the system. Although real testing is often desirable, simulation testing is a more flexible and cost-efficient approach. It allows for a more controlled and reusable environment that can be tweaked much easier. Within the IoT domain simulation testing is most often used to test technical aspects of the system such as network related features, power consumption etc. The most well-known examples of such simulators are NS-3 [37] and Omnet++ [38]. Various IoT operating systems such as Contiki and TinyOs also offer their dedicated simulator, Cooja [39] and Tossim [40] respectively. Some simulators are focused on testing more high-level IoT setups such as the iFogSim simulator [41] which is used to test IoT edge or fog architectures. Finally, D'Angelo et al. demonstrate the use of the Gaia/Artis specifically to run large-scale IoT simulations [42]. Also here, the focus is mainly on testing technical aspects such as power consumption and network utilization.

The problem however with the testing approaches described in this section is that they typically do not take the impact of behavior into account when testing IoT systems and are often focused on testing only non-functional requirements. Many of the classic testing practices isolate certain parts of the system and test it in isolation. We want to

test the behavior of the system at the system level and evaluate the global functional requirements of the systems. Therefore, we look at the IoT system as a black box.

2.2.2 Role of behavior in IoT systems

To better understand the importance and the role of behavior in IoT systems it is important to clearly define different types of behavior. We differentiate between local behavior at the local or entity level and emergent behavior which arises at the global level of the IoT system.

Local entity behavior This type of behavior is exhibited by the Local Entity (LE) level of an IoT system. With the term local entities we refer to local actors that operate autonomously and have some type of behavior. This behavior can be very different ranging from a simple actuator that responds to certain inputs, a sensor forwarding inputs towards devices whose behavior is powered by an AI. But also human actors can be seen as local entities. Actually, any connected entity that is able to communicate with other entities in the IoT system can be seen as a local entity. In the SeRGIo use case, mail(wo)men and citizens are also LEs. We refer to these entities as LEs in the remainder of this chapter. The local behavior exhibited by these LEs has an impact on the overall behavior of the system. Therefore, this behavior is an important part of the system and needs to be taken into consideration when evaluating the system.

Emergent behavior A special type of behavior in IoT is emergent behavior. Emergent behavior is the overall, global behavior posed by a system. It inherently depend on the behavior and interactions of LEs with each other and with the environment. Mataric et al. define emergent behavior as a collection of actions and patterns that result from local interactions between elements and their environment, which have not been explicitly programmed [1]. It is loosely based on the emergent behavior that is observed in bird flocks where birds apply three local rules which result in emergent flocking behavior (e.g. remain at a fixed distance to neighbor birds). Roca et al. argue that this emergent behavior will lead to improved scalability, interoperability and cost efficiency of ultra-large-scale IoT systems as opposed to traditional approaches that strongly rely on extensive programming of explicit behaviors [2]. Emergent behavior typically originate from autonomous entities, with adaptive or evolving behavior, that are interacting with each other and with the environment. This type of behavior particularly benefits the IoT areas which require the interaction of an enormous amount of devices where relying solely on a centralized architecture is insufficient. Examples are such as smart power grids, autonomous car flocking and smart traffic lights.
In the context of this work we are interested in testing functional aspects of IoT applications at the global system level. Furthermore, we want to evaluate how we can include behavior when testing functional requirements of an IoT system. We believe that the dynamic impact of local entity behavior can not be neglected in IoT systems. As in many cases, there is a strong connection between human actors and the IoT system as a whole. Furthermore testing systems from the bottom up reduces the complexity of evaluating the entire system, as you start from evaluating the local behavior opposed to evaluating the complex global behavior. This reduction of complexity leads to reduced error made by the tester and makes it computationally more feasible. The state-of-the-art (SotA)

literature lacks clear methods to validate, verify this behavior and its relationship to the IoT system as a whole. In the next sections, we present various techniques and list open challenges that arise when testing behavior in IoT systems. Furthermore, when testing emergent behavior in IoT systems, real-time interactions between LEs and the environment (e.g. a IoT middleware system) are required. Solutions to facilitate this are largely ignored in SotA literature. Later in this chapter, we look at how we can leverage large-scale simulation techniques to do this.

2.3 Simulating behavior in IoT system testing

Many IoT services such as participatory sensing, adaptive traffic navigation and more, inherently rely on human behavior. Furthermore, some of them even depend on the behavior posed by people interacting with each other and with various IoT devices in order to provide qualitative services. As pointed out by Nunes in the context of Cyber Physical Systems (CPS), most CPS are however human-centric applications where humans are an essential part of the system and most of these systems still consider the human as an external and unpredictable element [43]. The same argument holds for Internet of Things systems and behavior posed by other LEs such as smart actuators and sensors.

For example, in the participatory sensing use case, human actors are responsible for collecting sensing tasks. These actors behave according to certain semi-predictable spatio-temporal patterns, e.g. a mail(wo)man walks the same route each day and an average citizen goes to work and returns back home at roughly the same time using the same route. Furthermore, the actors also add an element of stochasticity to the system. For example, not every SeRGIo user will always be able or willing to respond to sensing tasks.

One can not rely directly on human actors and readily deployed IoT LEs to evaluate the behavior of the system prior to when the system is fully operational. Therefore, simulation can be seen as an effective alternative to real LEs during the test phase. Opposed to solely simulating technical, non-functional aspects of IoT systems as described in the previous section, we propose the simulation of LE behavior in order to evaluate the functional requirements of IoT systems. These SEs should be able to interact in real-time with the IoT system middleware, as if they were actual real-life entities. The following techniques can be used to model human behavior:

Explicit modeling The behavior of human actors or smart devices can be explicitly modeled. A technique often used for modeling the Internet of Things is ABM. With ABM each IoT device or actor is considered an agent that is autonomous, dynamic and has the ability to interact with other agents or with the environment. These properties make ABM also ideal for modeling the behavior of human actors [44]. Within the Sergio project we implemented an ABM approach to simulate citizens that navigate through the city. These citizen agents are able to walk a predefined route and broadcast their location to the middleware system. Furthermore, the agents were able to respond to task sensing requests based on a predefined probability estimate. This technique allows for an easy, cost-effective and reusable testing strategy compared to the classic testing approaches described in the previous sections. However, the level of detail in the modeled behavior is limited. It is therefore very likely that not all variations of behavior are included in the models which could lead to imperfections or errors in the demonstrated behavior.

Data replay Many IoT projects leverage existing datasets to replay behavior. For example, the participatory sensing system presented by Marjanovi et al. uses a public dataset that contains discrete, timestamped GPS locations of 65 users [45]. For each user the data will be replayed at a predefined interval. The IoT middleware under test will perceive the incoming GPS logs as real-time behavior posed by an actual user. The same strategy could be used to test the Sergio project, to model the navigation behavior of citizens through the city. However, this would require the collection of qualitative data, which is not always available depending on the type of data and the required quality. Furthermore, explicit modeling of behavior is still needed in many cases in order to respond to stochastic events. For example, with Sergio when a sensing task request is sent to a user, the system needs to have a response within a limited timeframe. Since the exact time of the creation of these events is not predictable upfront, we can not rely on replaying data event responses and thus, some logic is required to handle, process and respond to the event

Both explicit modeling and data replay can be effective strategies to model human behavior. They definitely can be used to include LE behavior in the IoT testing strategy. Compared to relying on a group of actual human testers, the simulation strategy is more scalable, more cost-effective and reusable. But, they require additional efforts and don't guarantee that the simulated behavior is realistic.

2.4 Hybrid testing

Using simulation can be an effective testing strategy. However, it is difficult to fully rely on it during the entire development process. As pointed out by Crisan et al. the data generated by classic simulation methods, as presented in the previous section, cannot model precisely the traffic bursts and noise of real-world activities of people [46]. Real-world testing is still preferred as it will better represent the broad variations of human behavior. Therefore, we present a hybrid testing methodology. The methodology combines both the advantages of simulation, which is a cost-effective, scalable and reusable technique, with the advantages of real-life testing.
The concept of hybrid simulation has been described in related literature. Mainly, within the domain of structural construction simulation
[47]. It allows experiments to be conducted in which structural components with complex responses can be modeled experimentally and more well-known components can be represented within an analytical model. The motivation for using hybrid testing in the Internet of Things domain is similar. Combining complex behavior, represented by real-life testing with more homogeneous behavior represented by simulation. In more related work, Arora et al. present a hybrid approach for their Kainsei, wireless sensing simulator [48]. However, in their work they consider the real world and the simulated world as two separate environments where no real-time interactions are required. Instead, data gathered in the real world need to be transferred manually to the simulated environment. In the following sections we present three novel solutions or methods that should be taken into account when implementing a hybrid testing approach.

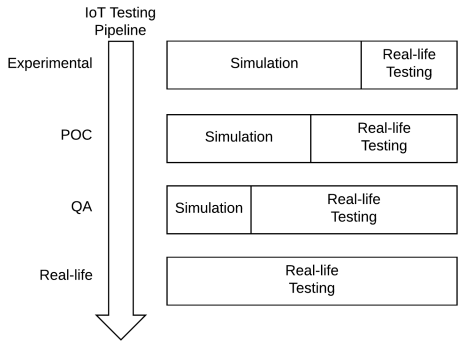

Figure 2.1: IoT testing pipeline: Moving from experimental setups to real-life tests

2.4.1 Hybrid testing pipeline

The core of our methodology is based on a dynamic mix of both simulation and real-life testing as can be seen in Figure 2.1 below. In the early stages of development, functional testing of the IoT project will rely mainly on simulation and only for a small part on real-life testing. The amount of simulation will decrease when moving to later stages in the development pipeline.

In the experimental phase it makes sense to almost fully rely on simulation. At that point, fast development progress is key. Later phases move closer to deployment and correspondingly more accurate behavior is required. In the SeRGIo project, for example, a lot of focus during the early development stages was on building a stable simulator. Once the simulator was completed, we had a testbed environment that allowed us to easily experiment with various task distribution algorithms without worrying about the effort and cost that would occur when deploying a testbed in real-life. In later phases we focused on finalizing the most promising algorithms and gradually testing these in real-life environments.

2.4.2 Uniform communication interface

An important aspect of our hybrid testing methodology is that an IoT middleware system under test or other IoT entities are not aware whether the entity, which is generating specific behavior, is simulated or actually operating in real-life. The underlying method should be fully transparent as illustrated in Figure 2.2. The simulated environment should be able to interact with the middleware and with other entities in real-time. This has as a consequence that a clear separation of concerns is required and that the communication interfaces are clearly defined and documented. The Sergio project relies on a variety of interface techniques such as REST and AMQP. In the context of IoT also COAP [49] and MQTT [50] can be used as more resource-efficient alternatives.

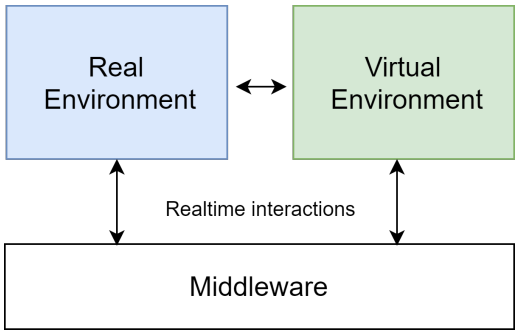

Figure 2.2: Interactions between virtual and real environment with the IoT middleware

2.4.3 Synchronization challenges

By implementing a hybrid testing approach, there will be a mix of data generated by either SEs or real LEs. It is important that there are no unnecessary inconsistencies between the data generated by the various sources. Synchronization faults in time and space can add fake inconsistencies to the testing environment. For example, in the Sergio use case it is possible that a SE responds to a sensing task request with a message that marks a specific area as "dirty" while a real LEs would mark it as "very clean". As a result there will be a strong variation between the observed data points. This is can causes significant problems. Let us for example assume that the SeRGIo project would adapt its behavior when a large variation in data points is detected in a certain area. Such variations could indicate that there is a lot of uncertainty about this area and additional sensing points are required to increase certainty. When more sensing points are added in the area, the load on the servers or edge devices located near the area will increase as a consequence. This in turn, could lead to a reconfiguration of reserved compute resources which is actually unnecessary and is only triggered as a consequence of the lack of synchronization between simulated and real LEs.
Clearly, a proper synchronization mechanism is required when implementing a hybrid testing approach. We present two techniques below. Synchronization can be implemented over multiple dimensions. We limit ourselves to synchronizing over the time and space dimension. In the context of IoT projects these are the most relevant dimensions. Within the space dimension we are concerned to maintain consistency between data sensed at a specific location. Within the time dimension we are concerned to maintain consistency between data sensed at a specific time. Preferably, both dimensions are take into account.

Proxy based synchronization: One of the requirements for implementing hybrid synchronization is that the IoT middleware is unaware whether the LE that is transmitting messages is real or simulated. Therefore, the synchronization mechanism cannot be implemented as part of the middleware. Instead, we propose a proxy synchronization mechanism as illustrated in Figure 2.3 below.

The proxy intercepts all messages sent to the middleware. Messages originating from the real environment are directly forwarded to the middleware and also used to train a prediction algorithm which tries to match the space and time dimensions to the data

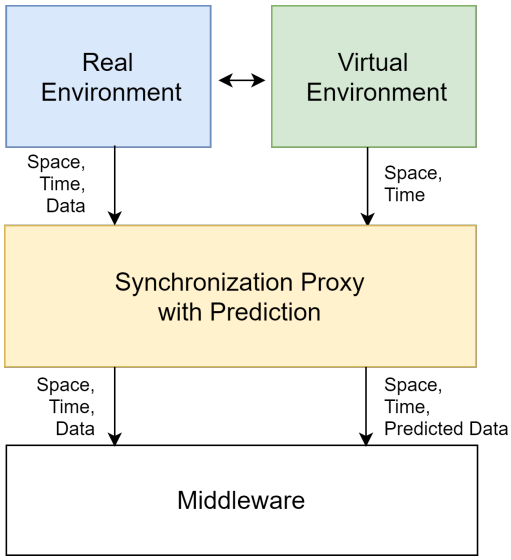

Figure 2.3: A proxy based solution that synchronizes the observed data from the real environment with the virtual environment

values. Messages originating from the virtual environment will only contain a space and a time value. The proxy uses these values to estimate a data value that matches the data coming from the real environment. When no messages are received from the real environment, the prediction will be random. When time passes, and more and more messages are processed from the real environment, the predictions made by the proxy will better match the actual data distribution. An example can be seen in Figure 2.4 below where a Self Organizing Map (SOM) prediction technique is used to match the actual data distribution. After training the SOM neurons are able to better match the actual temperature distribution. The illustration only shows the time dimension, also the space dimension should be used in practice but is now left out for visualization purposes. Suppose that a virtual LE sends a message at midnight. The SOM will try to match this time value to the closest neurons, in this case the SOM will output a value somewhere near ten degrees Celsius and send it to the middleware. In some specific cases it might even be useful to feed the sensor data originating from the virtual environment back to the real world that could impact the behavior and actuators of these real-life LEs.

Reducing synchronization requirement by space isolation: Another technique is to isolate certain geographical areas. These areas can then be limited to only serve real LEs or only simulated LEs. An example is illustrated in Figure 2.5 below. The left area is fully simulated, all data sent from this area to the middleware originates from the simulation. Whereas, the right area contains a physical test bed, all data sent from this area originates from the real world. The advantage of this is that it reduces the need for an additional synchronization mechanism between real and simulated environment, like the one presented in the previous section. There will however, remain a mismatch in data variability at the border between areas (illustrated by a red line in the image

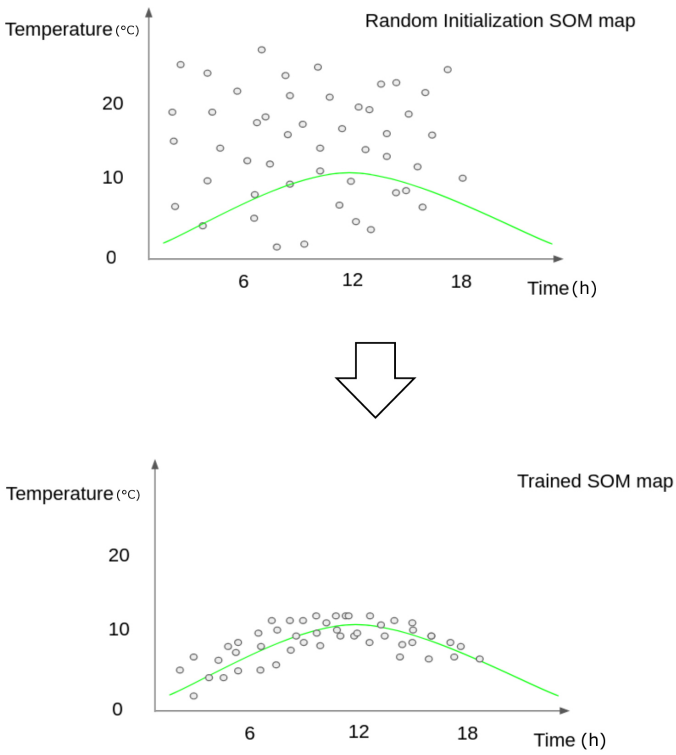

Figure 2.4: Automatic matching of virtual observations with real observations using a Self Organizing Map

below).

2.4.4 Evaluation of hybrid testing

To further motivate the advantages of implementing a hybrid testing approach we present our practical experiences gained from implementing a hybrid testing method in the SeRGIo project. During the early stages of middleware development we tested our sensing task distribution algorithm primarily in a simulation environment. One of the evaluation metrics was to verify the effort required by humans actors to perform a given sensing task. Based on that we could then calibrate the optimal refresh rate of sending out sensing tasks to obtain an optimal spread of data coverage in the city. It is hard, almost impossible to estimate this only based on assumptions of human behavior modeled in a simulation environment. This is because we can't estimate upfront how much time a task would take and if people would either accept or reject the tasks. Instead we moved a stage further and tested the algorithm in a real environment using a basic smartphone

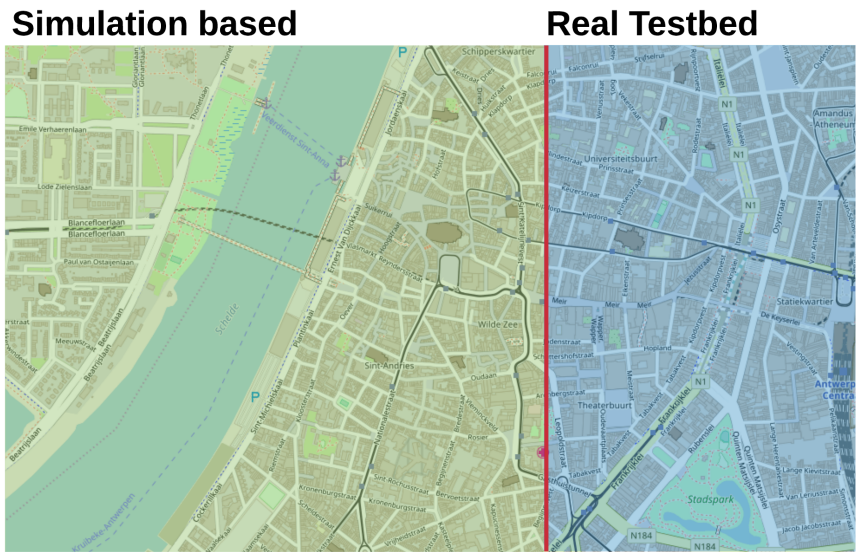

Figure 2.5: Space isolation: separating which part of a region is either real or virtual

application that was able to interact with the middleware under test and receive sensing tasks. An example of this task was "What is the cleanliness of the street at location X (which was in a max range of 400 meters)?".

2.5 Conclusion

When testing large IoT systems the amount of LEs that need to be simulated will be very high. This is especially the case when simulating emergent behavior where a large amount of interacting entities are required. Most state-of-the-art simulators are based on a monolith architecture which does not take advantage of parallelization or multi-server distribution that will be required to run many thousands of virtual LEs in parallel. In the first optimization technique we try to reduce computational cost by better balancing agents over multiple computational units. This work is presented in chapter 3. In chapter 4 and 5 we discuss a dynamic model abstraction technique that allows to dynamically switch between models with various levels of accuracy and computational complexity.

Chapter 3

Adaptivity in distributed agent-based simulation

In the previous chapters, we discussed the problem of scalability in the context of large-scale agent-based Internet of Things simulations. We showed that in related work a parallel and distributed simulation approach is often used to cope with these scalability issues.

But distributed and agent-based simulations often suffer from an imbalance in computational load, leading to a suboptimal use of resources. This happens when part of the computational resoures are waiting idle for another process to finish. Self-adaptive load-balancing algorithms have been developed to use these resources more optimally. These algorithms are typically implemented ad-hoc, making re-usability and maintenance difficult. In this chapter, we propose to organize the distribution of agents adaptively by dynamically reacting to imbalances in computational load, synchronization load, and communication load.
We present a generic self-adaptive framework. The methodology is evaluated with the Acsim framework on two simulations: a micro-traffic simulation and a cellular automata simulation. For each of these scenarios a scalable and adaptive load-balancing algorithm is implemented, showing significant improvements in execution time of the simulation.
Although ABS is a relatively new simulation paradigm [5], it has been used as an effective tool in a wide range of research domains [4], [51]–[53]. The main characteristic in ABS is the concept of an agent, which is a self-contained autonomous entity, with the ability to interact with other agents and with the environment. These interactions can lead to complex emergent behavior [54]. ABS is, therefore, one of the most powerful and natural tools to simulate emergent phenomena using a bottom-up approach.

ABS has been used to evaluate and analyze behavior of complex large-scale dynamic systems such as traffic systems [53] or complex Internet of Things systems such as smart city environments [4]. However, traditional monolithic ABS simulations quickly run into problems when the scale of the simulation increases. This becomes especially problematic when the application of these simulations is time-critical. Therefore, reducing the computational cost and the run-time of these simulations is vital.

With this motivation, researchers have replaced the classic monolithic set-up by a distributed architecture. This can be achieved by partitioning the simulation into separate logical processes. This allows the simulation to be divided among multiple processors and servers, thus allowing to simulate larger systems and reducing the simulation runtime. This however also increases the complexity and may add inefficiencies such as the need for synchronization and slow remote communication between simulation partitions. Furthermore, the inherently dynamic aspect of agent-based simulation makes static partitioning inefficient because the computational load of each process changes during the simulation. This can lead to a significant waste of resources, for example, the simulation can start perfectly balanced, but over time the distribution of these agents can become highly imbalanced. Such distribution imbalance is often due to agents that change their locations, increase communications or change their internal load. A direct consequence of such imbalances is a significant increase in run-time and under-utilization of computational resources. As stated by Long et al. it is likely that such load imbalances occur in distributed agent-based simulations [55].

Most state-of-the-art load-balancing mechanisms are implemented in an ad-hoc manner, making them hard to reuse and maintain. The contribution of this chapter is a generic framework to implement self-adaptivity in distributed agent-based simulators. We evaluate this method using two different implementations: a large-scale micro-traffic simulation with a graph-based environment and a cellular automata simulation with the Sugarscape model.

In the feature diagram in Figure 3.1, we show that our contributions in this chapter are mainly in the Adaptive Model Partitioning area where we compare a static partitioning approach to a dynamic activity based approach.

The first section of this chapter discusses the concept of adaptivity and related work. Section two presents the architecture of the distributed agent-based simulation framework Acsim, that will be used to evaluate the experiments. Section three presents the main principles of a MAPE-K loop and its implementation in Acsim. Section four presents the specific examples and the conclusions are drawn in section five.

3.1 Adaptivity in Agent-Based Simulation

Adaptivity in agent-based simulations can be related to the notion of activity which was introduced by Muzy et al. as a measure of the number of events occurring during a discrete event simulation [21]. As stated by Van Tendeloo et al. activity can be interpreted depending on the particular resource one wishes to focus on (time, memory, energy,..) [18]. Therefore, both the communicational load and the computational load can be seen as types of ‘activity‘. For example, from a communicational load perspective, an agent has high activity if it generates many messages in a fixed time window. From the computational load perspective, an agent has high activity if its step duration takes a long time to process.

Given this definition of activity, we can go ahead and define adaptivity as the prop-

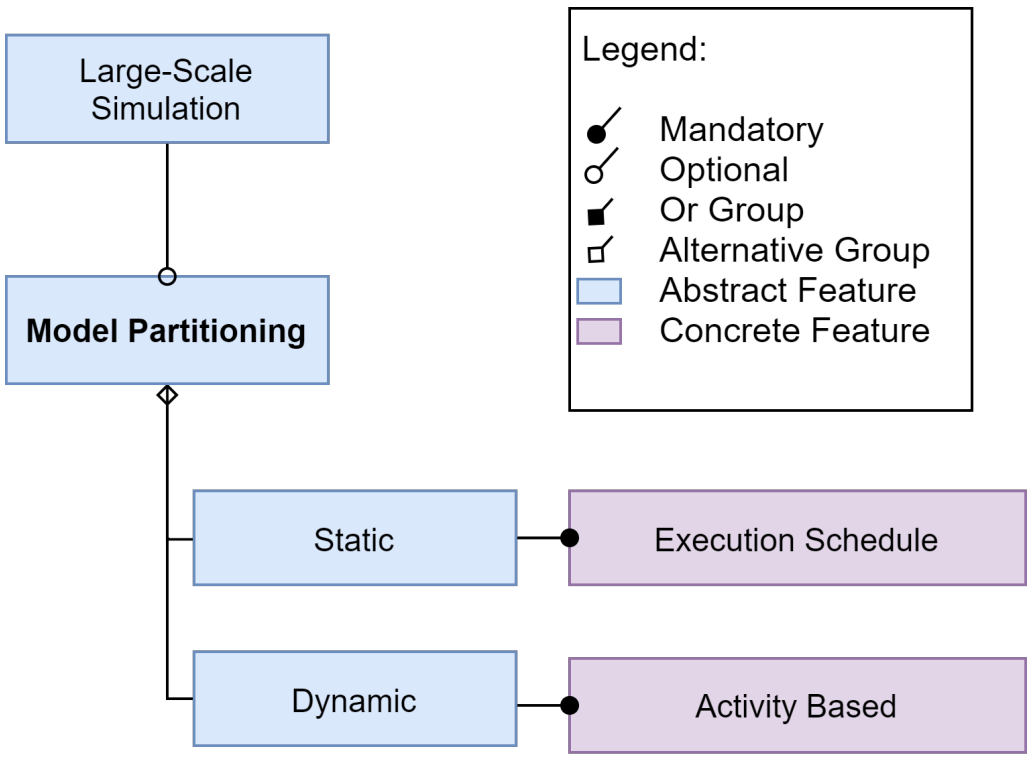

Figure 3.1: Feature Diagram - Contributions Chapter 3

erty of a distributed simulation framework to dynamically react to imbalances of activity with the aim to restore the balance and improve overall simulation run-time.

Adaptivity is typically implemented as a load balancing optimization problem based on global information [56] [57]. The activity is defined as a function of computational load, synchronization load and communication load. The disadvantage of these approaches is that they require global information to be stored or synchronized centrally and that the optimization algorithm is computationally intensive and thus less scalable. It is also possible to use heuristics that only require local information, making these solutions computationally much more efficient, but the obtained optimum might be local. For example, D'Angelo et al. present in their work a range of heuristics that trigger agent migrations based on local and remote communication patterns [20] and Q. Long et al. present a distributed load balancing algorithm based on partial local information [55]. But adaptivity is not constrained to solving load balancing problems only. In [58] and [29] the authors show that adaptivity can be used to dynamically switch abstraction levels of a single agent or a collection of agents. Switching to a higher abstraction level leads to a reduction in the computational load at the cost of losing accuracy.

Most of the related work relies on ad-hoc implementations of adaptivity. An exception is the work of Franceschini et al. who are using a MAPE-K control loop to implement an automatic simulation abstraction solution [58]. In the following sections, we expand on

this work and present the integration of a MAPE-K control loop in the Acsim distributed simulation framework. Furthermore, we show that MAPE-K can also be used effectively for adaptive load-balancing.

3.2 Distributed simulation architecture: Acsim

Acsim is a distributed Python-based agent-based framework, developed by the authors, inspired by Mesa [59]. It has been developed as a prototyping simulation framework. The goal of the framework is not to be a production-ready simulation framework but to allow for the validation of state-of-the-art techniques regarding simulation scalability. We hope that these techniques will eventually inspire production-ready distributed agent-based simulation frameworks.
One of the main motivations for the development of Acsim is the observation that there is an increasing need for large scale simulators in the context of IoT and Smart Traffic applications. Due to the increase of connectivity of smart devices and the availability of real-time data, simulation platforms provide the opportunity to simulate entire cities. Simulation technology enables the creation of a virtual testbed of large-scale IoT applications and allows for real-time simulation-based optimization. An application of this technology is, for example, a real-time city-wide and simulation-based traffic light optimization platform. But state-of-the-art simulators are limited in their scalability capabilities to support such technology. This is the challenge that Acsim tackles. Although Acsim focuses on large scale IoT and traffic simulation, it can also support other agent-based simulations.

Acsim relies on a conservative time-stepped synchronization mechanism. Where time

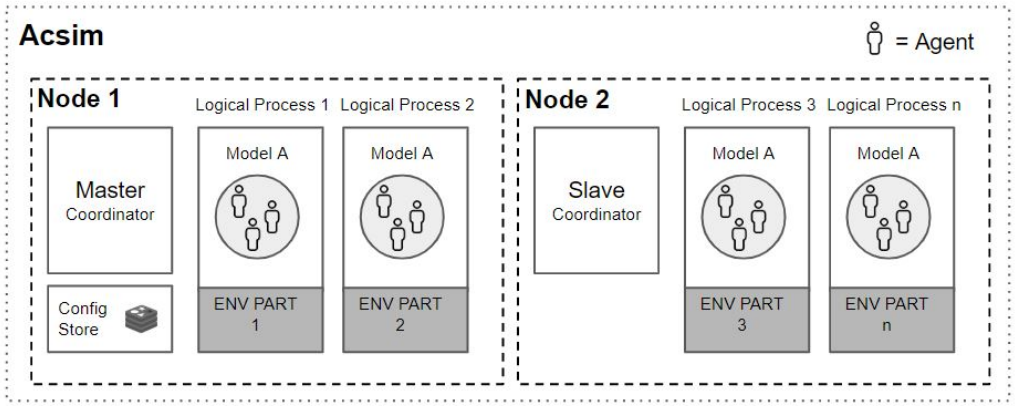

Figure 3.2: Acsim - Distributed simulator architecture. Acsim contains a cluster of nodes and a node represents a physical device with one or more CPU cores, connected to other nodes via the network.

is collectively progressed after the completion of each individual agent step. The architecture of the simulator is displayed in Figure 3.2. Acsim consists of three main building blocks: 1) Agent: represents an entity at its highest granularity, an agent contains a state, can adapt its state at each time-step and has the possibility to interact with other

agents using message-passing and interact with the environment. 2) Model: a model serves as a container for a specific type of agent and is responsible for the initialization of all agents of this type. For example, a class of car agents will be part of a car model. This car model will initialize all cars, generate routes and collects car-related logging information. 3) Logical Processes: Acsim consists of multiple sub-simulator or LP's. Each LP manages a part of the environment and a collection of agents that are located in this partial environment. It runs a dedicated process and is responsible for low-level simulation tasks such as handling agent migrations, managing message-passing between local and remote agents, collecting logs and initiating agent steps. An agent step is a discrete step forward in time. Only as part of a step can an agent adapt its state or communicate with other agents and the environment. The global synchronization is managed by the master coordinator. The coordinator orders all LP's to execute the next step. Furthermore, the coordinator collects and stores logs generated by the LP's. Finally, Acsim has extensive monitoring capabilities, enabling an in-depth analysis of local and global simulator performance.

3.3 MAPE-K as a generic framework for adaptivity

Due to the ever-increasing complexity of computing infrastructure, a shift to self-managing systems is observed in the field of software development. In 2005, IBM introduced MAPE-K loops to deal with this complexity [60]. Measure Analyze Plan Execute - Knowledge (MAPE-K) loops are closed feedback loops which can handle the complexities of self-adaptivity. More recently, [61] described templates on how to utilize MAPE-K control loops to different distributed applications. The implementation of most adaptive optimization strategies in a simulation is ad-hoc and cannot be reused efficiently. We propose the application of a MAPE-K control loop as a generic solution that will allow existing adaptivity strategies to be efficiently implemented and maintained.
As mentioned above, the Acsim framework is step-based which results in the simulation being as fast as the slowest simulator in the distribution. There is no guarantee that this local optimization leads to a global optimum. The overhead of calculating the global optimum, at a master node, increases with the scale of the simulation. Because of the varying load-distribution over time, the global optimum shifts and a new optimization iteration is needed. Our approach focuses on a solution to partitioning and merging distributed environments. Our approach is generic, each simulator can easily implement its specific logic as part of the MAPE-K framework implemented in Acsim. Execution of the MAPE-K loop is handled by the Acsim framework. We put extra emphasis during development that the MAPE-K framework is implemented in a modular way, as part of the simulation coordination engine. Its architecture can therefore be transferred to most agent-based simulators and be integrated without significant changes. This is because the MAPE-K framework breaks the barrier between simulation application and simulation engine. We can refer to this as leaky abstraction. This design choice has been made in order for the framework to be implemented in other simulation engines without breaking existing simulation applications. The trade-off however is that a simulation application developer needs to be aware of low-level aspects of the simulation engine when developing a MAPE-K implementation for its application. Next, we will go in-depth on the structure of the MAPE-K framework integrated into Acsim:

1. **Measure**: During this phase, logs regarding the model, simulator and environment are retrieved from each subpart of the Acsim framework. When a MAPE-K iteration starts, these logs are stored to the shared knowledge. This knowledge base is located at the master node. To enhance the scalability, only computationally less expensive algorithms are used at the master level.
2. **Analyze**: This phase has access to the shared knowledge base to identify bottlenecks and flag optimization opportunities. These identifications do not provide a solution but an indication of the performance of a certain entity in the framework.
3. **Plan**: This step collects all flags and generates an optimization plan without execution. There could be multiple optimization plans in a single MAPE-K loop.
4. **Execute**: This phase of the loop runs distributed after receiving one ore more optimization requests from the planning phase. This phase has the highest computation requirement in the loop. The optimization algorithms used can vary for each application. When a local optimization is complete, a synchronization message is sent to all relevant entities involved in the optimization.
5. **Knowledge**: This part is shared between the first three steps of the loop. The execute step does not need the knowledge base as it only executes the plans created during the previous step. During each iteration, the knowledge can be expanded to store relevant information for future MAPE-K loops.

Each simulation will have access to the simulation logs, these are stored in the knowledge class. The MAPE-K framework implemented in Acsim allows for easy implementation of the phases and allows for reuseable, maintaineable and application-specific adaptivity behavior. The loop can be executed both locally and centrally. Also a hierarchy of multiple loops, affecting each other is supported by the framework.

3.4 Motivating examples

In the previous sections, we introduced the concept of adaptivity and how we can implement it generically in the Acsim framework using the MAPE-K framework. In this section, we validate this approach on two different agent-based simulations. In both scenarios we implement a novel activity load balancing heuristics. We differentiate compared to classical adaptive load balancing algorithms by making sure the heuristics are not performed centrally but at the level of a LP to ensure scalability. In the experiments our aim is to improve the global step duration GSD of the entire simulation. We can express it as follows: $GSD = max_i(LSD^i)$, where LSD^i is the local step duration of LP i. In other words, the global step duration is always equal to the worst LP step duration. The reason for this is that Acsim relies on a conservative time-stepped synchronization algorithm, as discussed in section 3.2. In the examples below the goal is to improve the activity balance with each MAPE-K iteration. To gain insight in how LP's are performing, we distinguish the different contributions to the step duration (as discussed in detail in [20]): the Model Computation Cost (MCC), the Remote Communication Cost (RCC), the Local Communication Cost (LCC) and the Model Synchronisation Cost (MSC). The weight of each contribution is application-specific. When an imbalance occurs, for each variable a different optimization strategy could be used. When optimizing on a local level, each LP calculates their cost balance using only local information.

3.4.1 Adaptive local optimization of compute cost: a micro-traffic example

Introduction

In this example we perform a micro-traffic simulation of a 20km by 20km urban area where cars are making random trips. Each car is an agent, managing its state and adapting its acceleration based on speed regulation and the acceleration of leading cars. The implemented models are based on the Intelligent Driver Model [62], which is a state-of-the-art car following model and the lane-changing model MOBIL (Minimizing Overall Braking Induced By Lane Changing) [63]. This implementation leads to both realistic local behavior and realistic emerging behavior. All cars comply to standard traffic regulations and priority rules.
The environment is represented by a directed graph datastructure. Edges are roads (with single or more lanes) and nodes are junctions. A car agent can interact with the environment by requesting where nearby cars are located. Car agents can also interact with each-other to request acceleration and related information, or with traffic light agents to request the state of a traffic light.
During initialization the environment is partitioned based on the number of available cores. The partitioning algorithm is a multilevel recursive algorithm for multi-constraint graph partitioning as presented in [64]. It attempts to balance node cost, which represents the computational cost observed at a node in the graph, of the graph partitions and minimizes edge cut. A single LP will manage a single environment partition and the agents located in this partition. When agents leave the environment partition they will migrate to an LP that manages one of the neighboring partitions. At the edges of a partition, car agents require state information of agents that are located in the neighbouring partition. Therefore, we include a synchronization mechanism. This mechanism broadcasts the state of an agent, located at a border area, to neighboring partitions after each state update. In this scenario the cost of a step depends on two activity parameters: Model Computation Cost, MCC and Model Synchronization Cost MSC. In the remainder of this section we elaborate on how we can dynamically load balance these activity parameters using local information only in order to reduce the global step duration.

Description of the optimization algorithm

A significant amount of research has been done in the context of distributed micro-traffic simulation. The load balancing problem is one of the most discussed problems within this context. As stated in [65] it is necessary for all simulation processes to consume similar amounts of computing power in order to run at the same speed. Ramamohanar et al. [66] introduced a spatial workload balancing approach where they partition the environment in grids. As pointed out by the authors, this approach is static, and unable to react to changes in computational load introduced by agent migrations. Cordasco et al. presented a distributed extension to the Agent-Based Simulation framework MASON [67]. In their work they put extra emphasis to the partitioning and load balancing problem. But also their implementation is not generic nor dynamic. Instead, other work, such as Xu et al.'s work, presents an adaptive graph partitioning approach [56]. They essentially execute the graph partitioning algorithm multiple times, on the entire traffic

network, when imbalances are detected. The problem with this approach is that the algorithm runs on the entire network, making a distributed approach difficult.
To solve this problem, we propose a heuristic-based approach, powered by the MAPE-K framework presented in the previous section, that is able to run in a distributed way. The global idea of the algorithm is that we keep track of activity using an activity graph. For example, assuming car agent computational load is homogeneous, we keep track of the number of cars located on the incoming edges of a node. When imbalances are detected between neighboring environment partitions we allow an overloaded partition to migrate a collection of its border nodes and edges to a neighboring, less occupied partition. This is visualized in Figure 3.3. The amount of nodes and edges that get migrated depends on the amount of activity that needs to be transferred in order to reestablish the activity balance.

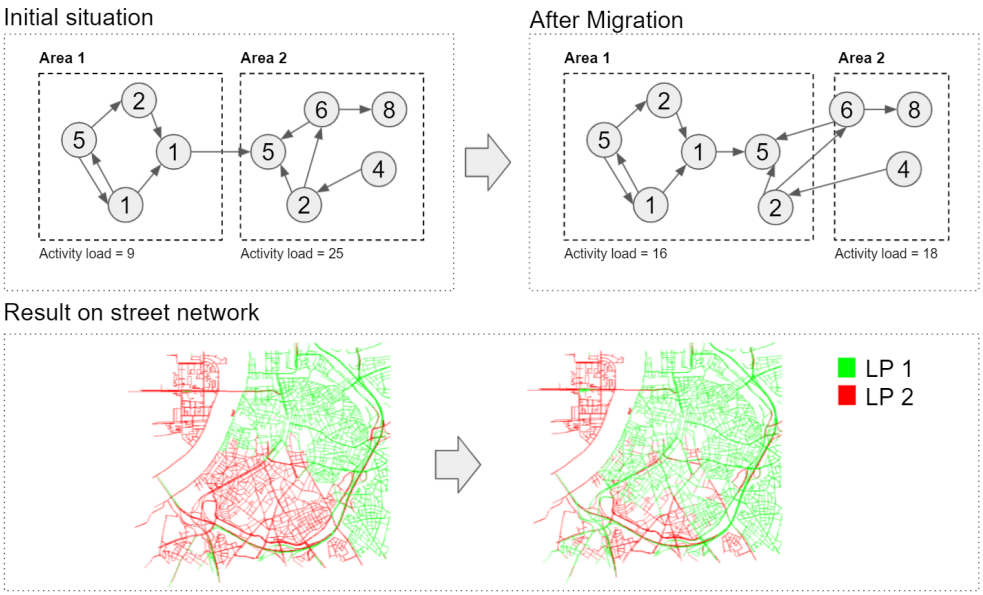

Figure 3.3: Heuristic: Load balancing using local activity graphs

MAPE-K Implementation

The implementation of the computational load balance algorithm in the MAPE-K framework is explained below:

1. **Monitor**: We keep track of the Global Step Duration (GSD) and the local step durations of the simulators (LSD^i).
2. **Analyze**: The average LSD is calculated. When one of the LSD exceeds the average by 20% or more, the algorithm evaluates if part of the computational

activity can be offloaded to the neighbors (this is achieved by migrating nodes, edges and agents). If this is the case a ‘migration flag‘ is set.

3. **Plan**: When a migration flag is found, a plan of execution will be created. This plan orders the overloaded area to migrate a given amount of activity to one of its neighboring areas that has been selected in the Analyze step.

4. **Execute**: The overloaded area will calculate which nodes it can offload. Consequently, both the originating area and the destination area will update their graph datastructure accordingly.

5. **Knowledge**: This part is shared between the first three steps of the loop to keep track of the global and local step durations.

Results

We ran an experiment to test this implementation. In the experiment we randomly generate trips in a city center. We introduced an initial imbalance of 1/10. This could be a realistic scenario when people are leaving a residential area to an industrial area in the morning. We expect the algorithm to restore this imbalance over time. Thirty runs of this experiment were executed, the average and standard error are displayed in Figure 3.4. In both graphs we compare a non-adaptive approach with an adaptive approach. The MAPE-K optimization is performed at time-step 250. Note that this time-step has been chosen based on the application related observations and requirements. As this is mostly a domain-specific decision, the time-step interval can be easily adapted by the simulation developer. We observe a significant reduction of step duration when the optimization occurs.

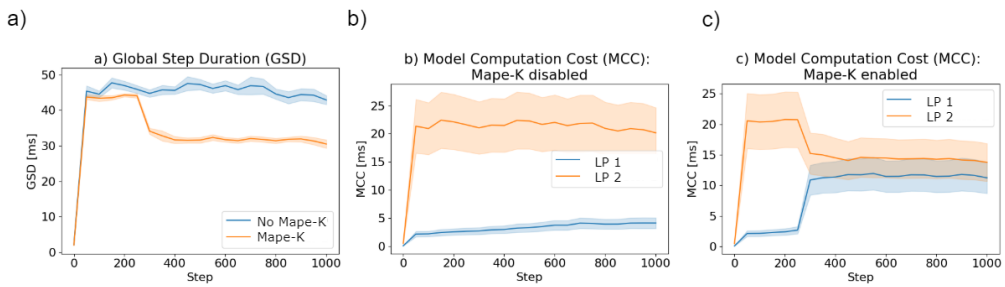

Figure 3.4: Results - with and without MAPE-K adaptive optimization, micro-traffic simulation

Future work: Balancing Synchronization Cost

As explained in the introduction, the step duration not only depends on MCC. It also depends on MSC. The impact largely depends on the scenario. When there is a large amount of traffic at the border areas of environment partitions, the MSC will be significant and cannot be ignored. Therefore, further optimization will be required. We propose a technique that can be explored in future work. The general idea is that we

can measure the synchronization cost based on the amount of agents located in a border area. When an imbalance in synchronization cost is observed between areas, we can simulate the synchronization cost after incremental walk in the graph. This is similar to incremental expansion demonstrated in Figure 3.3. When the synchronization cost of the incremental expansion is lower than the initial cost, we can perform a migration of nodes and edges.
In conclusion, the proposed synchronization heuristic combined with the computational cost balancing heuristic is expected to lead to a further reduction in step duration. The proposed heuristics will improve upon sub-optimal scenarios where imbalances are observed in neighboring areas, in a scalable and computationally efficient manner. But, it is limited to finding local optimal solutions, not a global optimal solution.

3.4.2 Adaptive local optimization of communication cost: a cellular automata example

Introduction

In this example we use the agent-based simulation Sugarscape [68] with a cellular automata environment. This example was chosen as it is a well-known agent-based simulation and because the type of the environment Sugarscape uses is used by many other agent-based simulators. This shows that ideas presented here are transferable to similar agent-based simulations. These simulations typically lead to emergent behaviour and can be used in, for example, biology [51]. In Sugarscape, sugar is grown in each cell of the environment at a certain rate and the goal of the agents is to survive by collecting enough sugar. If an agent cannot satisfy his metabolism, he is replaced by a randomly initiated agent at a random vacant position. The agents are characterised by a metabolic rate and range of sight. At each step they search for sugar by looking in the four perpendicular directions and move one step towards the cell with the highest sugar level, collecting the sugar at their new location. The environment regrows sugar at each step in the cells according to a fixed rate until a maximal sugar level is reached. The MCC of the agent is relatively small but instead the step duration depends mainly on the LCC and the RCC (with RCC being significantly more expensive). The *RCC* is the result of agents that are close to the edge of a simulator and searches within the next simulator for sugar. This consists of two messages that are sent between the simulators: one to ask for the amount of sugar on the cells of interest and one with the corresponding answer.

Implementation

Our optimization algorithm keeps track of a communication-based activity graph, where imbalances in *LCC* and *RCC* between simulators are monitored and dynamically improved by migrating parts of the environment. The MAPE-K cycle is implemented as follows:

1. **Monitor**: The local and remote (both contributions due to received and sent messages) compute time for each cell is logged.

2. **Analyze**: Bottleneck simulators are identified by comparing their Local Step Duration (LSD) to the GSD.

3. **Plan**: A plan is created to partition certain sections of the simulator's environment to restore imbalances that might have manifested over time. Each section is evaluated using its logged *LCC* and *RCC*. During partitioning, the algorithm can decide to migrate a section which will switch *LCC* to *RCC* (and possibly vice versa), and as a consequence reduce overall cost.

4. **Execute**: Each simulator executes his part of the established plan and possibly migrates parts of its environment to another simulation process.

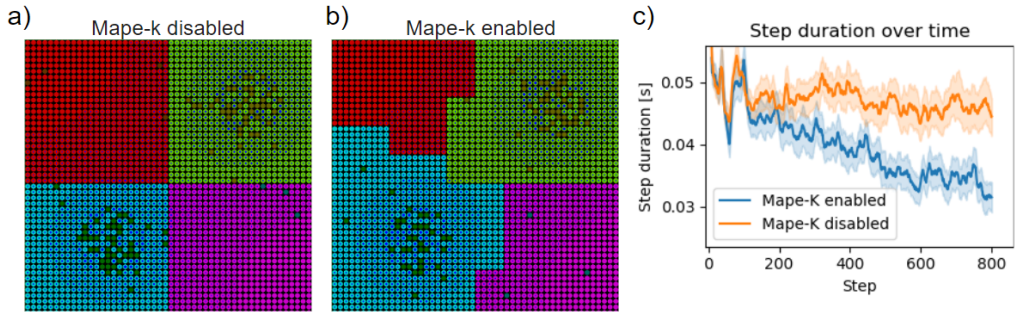

Figure 3.5: Results - with and without MAPE-K adaptive optimization, Sugarscape.

Results

To evaluate the performance of the adaptive approach, we ran 30 random initiated Sugarscape simulations and compared them to 30 non-adaptive simulations. Both simulations are initiated with four LP's managing each a quarter of the environment. The simulation stops after 800 steps, the MAPE-k framework is executed every 50th step. The results are illustrated in Figure 3.5. Image a) represent the initiated environment divided over the four LP's. Image b) shows the same environment where the borders are optimized based on the current activity. Finally, c) illustrates the average step duration and the standard error. From these results we can see that once the random activity of the agents is replaced with emerging behaviour, the adaptive approach improves performance.

3.5 Conclusion

In this chapter, we presented a MAPE-K loop as a generic and effective framework to implement a self-adaptive distribution for agent-based simulations. We evaluated this framework by implementing two examples: a distributed traffic simulation and a sugarscape simulation. We showed that MAPE-K can be used effictively in multiple simulations to implement adaptivity. Furthermore, the results of the proposed adaptive load-balancing heuristics show a significant reduction in computational cost while being executed decentralized.
In chapter 5, we will leverage the framework presented in this chapter and apply it to a dynamic model abstraction problem.

Chapter 4

Dynamic Approximation

In this chapter, we look to reduce the computational complexity of large scale simulations using model approximation. In the previous chapter, we discussed how dynamic partitioning could improve scalability of large scale agent based simulations. In this chapter, we apply model abstraction by approximating the model of simulation regions by computationally less complex counterpart.

Transforming certain models to a less computationally intensive abstraction/approximation, will lead to a reduction of the computational cost of the simulation which in turn allows us to increase the overall scalability of the simulation. More specifically, we present an opportunistic abstraction/approximation approach, that is able to dynamically transform low-information areas to more abstract representations. We show that by using this method we are able to reduce the computational cost significantly.

The abstraction/approximation approach we present in this work is mainly based on the work that has been done in the area of model abstraction which has been studied extensively during the last 20 years. This research domain of model abstraction is in related work also referred to as meta-modeling, surrogate modeling or model approximation depending on the research domain of the author. In the context of this thesis we use these names interchangeably when we refer to certain related work, but in the scope of this thesis they have the same meaning. Model abstraction allows us to represent models at an appropriate level of abstraction, which often comes down to making a proper trade-off between computational cost, modeling effort and accuracy. In the scope of this chapter we are interested in the entity aggregation abstraction technique, where multiple models are replaced by a single approximated agent, that abstracts the overall behavior of the collection of models. For example, Rodriguez et al. present various statistical techniques that can be used to aggregate simulation models [14].

The technique that we present in this chapter dynamically approximates the behavior of multiple agents in a certain area by moving from a simulation area containing multiple individual car agents to a computationally more DES area. It is partly related to state abstraction and entity-aggregation, in the sense that we can approximate multiple models

by a single approximation and that we replace the initial behavior with an abstraction.

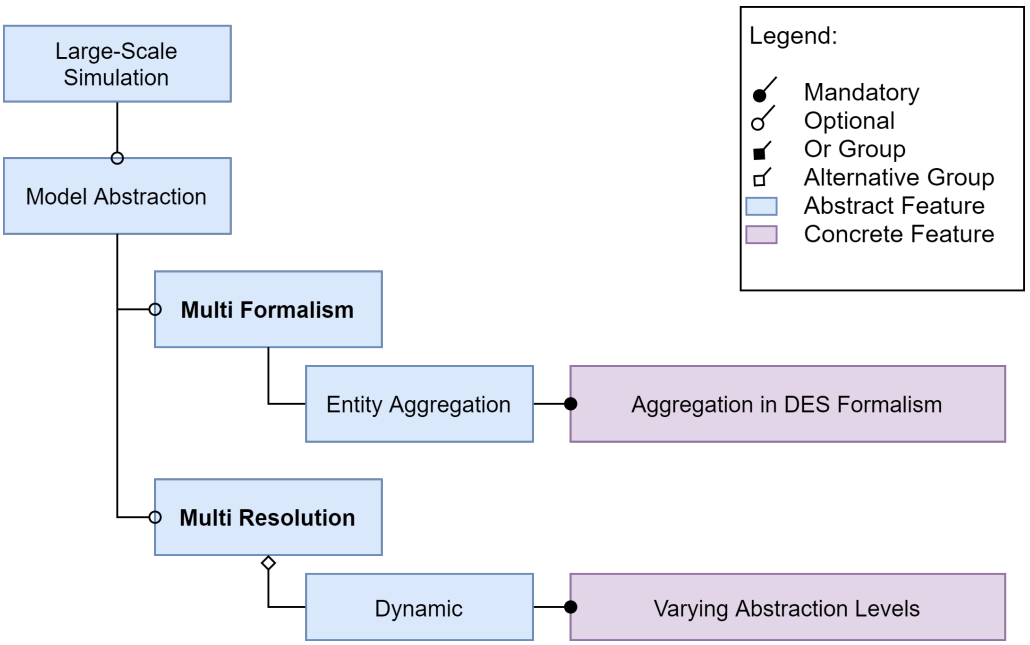

Figure 4.1: Feature Diagram - Contributions Chapter 4

The feature diagram in Figure 4.1 shows that our contributions in this chapter are mainly in the Multi Formalism area, where we focus on entity aggregation by switching multiple agent in the Agent Based Formalism to the DES formalism. Furthermore, we apply a dynamic multi resolution approach where we can vary abstraction levels during simulation run-time.

This chapter is structured as follows: in section 2, we introduce the core idea of switching between simulation formalisms in order to switch between abstraction levels. In section 3, we introduce a traffic simulation use case that will help to build intuition about the problems we're trying to solve. In section 4, we introduce the idea of opportunistically approximating simulation areas based on entropy. Section 5 shows the impact of the our technique both on computational cost and on the simulation validity. Finally, we position our approach from a more theoretical perspective.

4.1 Dynamic model approximation driven by information theory

To improve the overall scalability of our simulation, we propose to dynamically approximate models. Our hypothesis is that when the approximation is less detailed than the original model, the overall computational cost will be lower. However, at the same time we expect the models approximated with less detail and at a higher level of abstraction to lose information. In this work we look to opportunistically approximate models, to reduce the computational cost while maintaining the overall validity as good as possible.

Furthermore, we assume that not all areas of the simulation environment are equally important. The behavior in some areas have less impact to the global simulation behavior. In this thesis we attempt to exploit that assumption by trying to identify those less important areas and consequently remove details by replacing the original model with an approximation.

To further clarify this we will first introduce some notations:
When we consider the full set of simulation models in our simulation to be M , the set of low abstraction models, to be L and the set of high abstraction models to be H. Then $M = H \cup L$. With high or low abstraction models we refer to models that are expressed at a higher or lower level of abstraction. Explicitly modelled models are considered to be part of the set of low abstraction model L, and the set of approximated models to be part of the set of high abstraction models. The diagram below illustrates the hierarchical topology of our conceptual simulation framework. The Full simulation consists of both the set of high abstraction models and low abstraction models. In the remainder of this chapter we look at how to facilitate a proper transition from a low abstraction model to a high abstraction model. We want to maximize the size of set H while maintaining the validity of the global simulation.

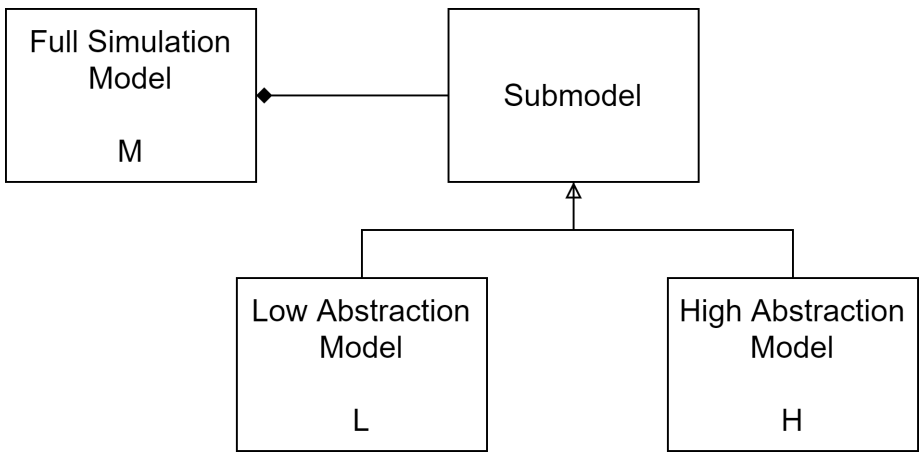

Figure 4.2: Topology of simulation models

4.2 A traffic use case

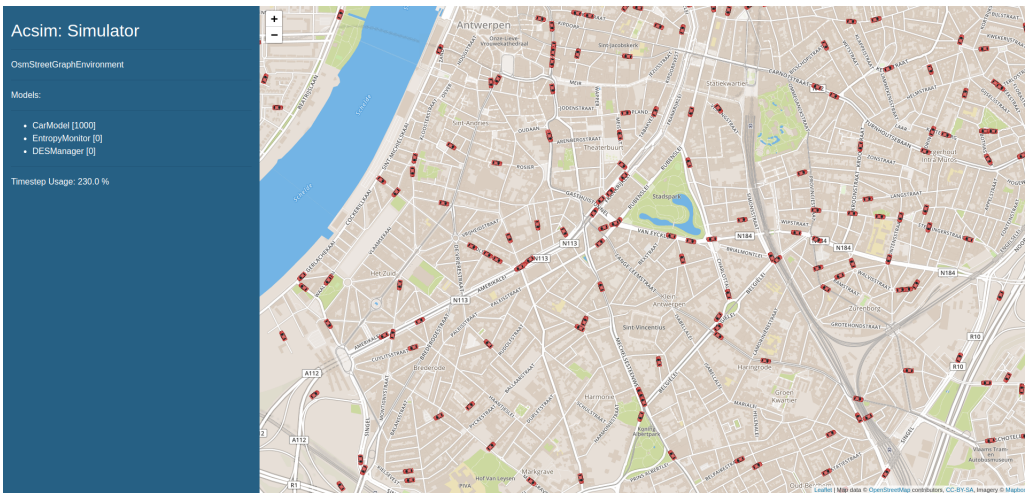

Figure 4.3: Traffic simulation in Acsim.

To validate the computational impact of opportunistic entropy-based transformation we introduce a basic traffic simulation use case using the agent-based IoT simulator Acsim [69]. This will also help to build intuition of the problem we are trying to solve. The simulation use case we present, consists of 1000 cars driving at various speeds as shown in Figure 4.3. The average speed of a car depends on the speed limit of the road the car is driving on. Based on the road type, the speed may deviate more or less from the given speed limit. Furthermore, we consider different speed distributions depending on the type of road the cars are driving on. These distributions are shown in Figure 4.4 below.

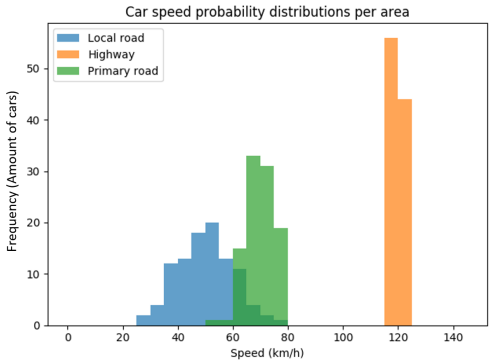

Figure 4.4: Car agent speed distributions per road type as observed in the simulation.

We expect more variability in the average speed on local roads due to traffic lights and road priority rules. Whereas, the variability in speed is lower on highways. These

assumptions give us a proper use case to evaluate the problem we're trying to solve, where, depending on the area, different types of behavior can be observed.
To model this simulation we use the agent-based Acsim simulator. The environment is represented as a (directed) graph-based open street map environment. Roads are represented by edges and road crossing are represented by nodes. Furthermore, the environment allows to efficiently run GIS range queries and calculate routes between various locations. By default, each car is represented by a single agent. The simulation is time-driven, this means that the simulation state is updated at discrete time-steps. For example, at each time-step the car agents update their location and run some computations to simulate a realistic, constant computational load. We do this because at this stage in the simulator the computational complexity of the car models is limited, in later chapters this is not the case. Therefore we simulate an increased computational complexity for now.

4.2.1 Area approximation

To move certain areas to a less-detailed approximation we allow the simulator to transform specific agent-based simulation areas to a DES representation of the area. The DES area approximation is also implemented as a single agent, which is running a discrete event simulation internally. It leads to an aggregation of multiple agents into a single model and an approximation given that we lose information regarding the individual velocity of each agent. This enables us to reduce the computational cost significantly but at the cost of losing local interactions. In the traffic use case, a simulation area is a single road section. As can be seen in Figure 4.5 below, each time a car agent arrives at a DES road-section area, it will be deactivated. This road section agent will then schedule an internal event when the agent can leave the area. The duration depends on the agent speed, which is sampled from the historical data of agents driving on the specific area. We can say that the DES road-section area agent is approximating the behavior of incoming agents by averaging its speed. Therefore, these agents are temporarily approximated and are temporarily part of the set of high abstraction agents H.

4.2.2 Computational cost

We consider the computational cost to be a measure that illustrates the computational cost to progress the simulation a single timestep t to the future. Many empirical, indirect execution time measuring techniques are described in literature to measure the computational cost of a simulation. An example is the Wall Clock Time (WCT) which measures the duration of the entire simulation from start to finish. However, as priorly explained our simulation will need to run in a hybrid mode, where virtual, simulated IoT entities need to interact with real-world IoT entities or IoT middleware systems in real-time. Therefore, our goal is to satisfy the predefined amount of P simulation timesteps per seconds in order that the simulation remains responsive to outside interactions. In many cases this minimum required amount of timesteps P depends on the context of the simulation. We apply the 'step-usage' metric to indirectly measure the computational cost of our simulation model. It is similar to the WCT of a single simulation step. The step-usage compares the observed computation time O_t of a single timestep t of the set of all simulation models M to the preferred timestep duration $d_t = \frac{1}{P}$.

$Step - Usage = \frac{O_t}{d_t}$.

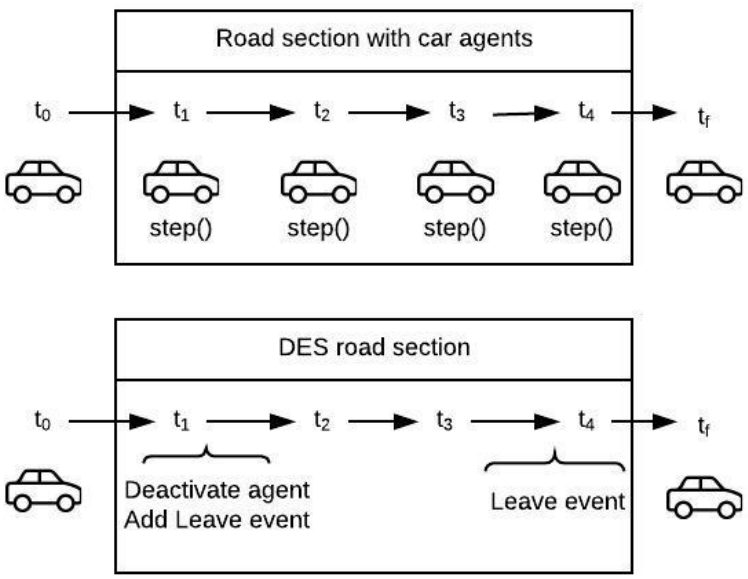

Figure 4.5: Road section with individual car agents vs. a Discrete Event Formalism agent that approximates the entire road section.

We can express the computational cost of the simulation at timestep t as a function C of set M, where we assume:

$$C(t, M) = C(t, L) + C(t, H) \text{ and,}$$

$C(t, H_i) \leq C(t, L_i)$ where H_i is an abstract approximation of L_i and $H_i \in H$ and $L_i \in L$.

4.2.3 Computational cost over time

Figure 4.6 shows the computational cost at each time-step of the simulation. The computational cost is expressed as a step-usage ratio C, as introduced in section 4.2.2 . This metric is sensitive to noise introduced by other processes, to remove this noise we applied a running average of 100 timesteps before plotting the data. It is expected that the computational cost remains constant over time. Figure 4.6 below shows that the computational cost during the first scenario remains constant over time as is expected. However, the computational cost exceeds the 1.0 time-step usage ratio. This could lead to the state of simulation agents being updated too late and as a consequence, to inconsistent or incorrect behavior of the simulation models.

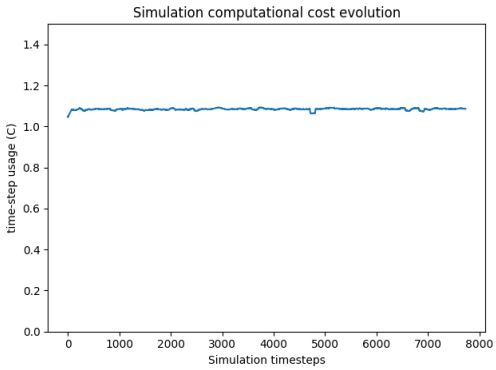

Figure 4.6: Computational cost over time without transformations to event-driven areas

4.3 Entropy based transformation

The overall properties of global emergent behavior of a simulation are created by the properties of the individual simulation models. In order to maintain the validity of the overall simulation, we will have to maintain the properties of the overall simulation and the properties of the individual models or the collections of models. Also note that the approximations we want to make are applied on an individual property of a model, in this case the velocity. We assume that some areas of the simulation contribute more (A_{H_C}) to the properties of the simulation compared to others. Therefore, it would be interesting to target simulation areas that contribute less to the overall behavior (A_{L_C}). We can assume that if we transform these low-contributing areas to more abstract approximations (A'_{L_C}), the overall emergent behavior is less sensitive to the loss of information and the inaccuracies introduced by these approximations. In other words, we expect the simulated global behavior to be more accurate when more low contributing A_{L_C} areas are approximated opposed to when more high contributing areas A_{H_C}.

Identifying low contributing simulation areas
When we continue to reason about this, we can conclude that after identifying the low contributing areas A_{L_C}, we can opportunistically transform these to more abstract representations which helps us to achieve our main goal, which is to reduce the computational cost while maintaining overall validity. This brings us to the core of this paper: finding a technique to identify these low contributing simulation areas. To do this, we propose to use the entropy measure from information theory [70]. Entropy is a measure that can be used to determine the amount information or the amount of randomness in a dataset, as expressed in formula 4.1:

$$E = \sum_{i=1}^{n} [p_i log_2(p_i)]. \tag{4.1}$$

There is a connection between the amount of information that can be observed in a given simulation area and its contribution to the global emergent behavior of the entire simulation. Furthermore, low information areas are often easier to predict and

therefore, the properties of a more abstract approximation of the area are easier to be maintained after a transformation. Another advantage of using entropy is that it is easy to measure. We can easily extend the simulation to keep track of the entropy value of individual areas. When comparing this with the entropy of the global simulation is should allow us to dynamically transform low-entropy areas to higher abstraction levels. This will decrease the computational cost while maintaining the validity. Furthermore, entropy is a general-purpose metric, it can be used for multiple simulations in multiple contexts without requiring expert domain knowledge. In related work, P. Lamarche, leverages similar information theory inspired techniques to evaluate the information loss of abstractions in the context of a multi-agent system [71]. The entropy-based technique we present in this work, is similar but is not used to evaluate the information loss after the abstraction, but rather estimates the possible impact on the global simulation behavior when a given area is abstracted or transformed.

Measuring entropy in the traffic use case

To decide when and where we can transform standard areas to DES areas we need to extend the simulation framework used in section 4.2 to measure entropy levels. We do this by regularly measuring the speed driven at each area and categorize these measurements at intervals of 5 km/h. This interval unit based on our observations because any value below didn't add a lot of information regarding the observed velocity distribution. It allows us to measure discrete probabilities and as a consequence to measure entropy as expressed by equation 4.1. When the entropy level exceeds a certain threshold we transform the standard area to a DES area. We use a heuristic to determine the threshold, it is set $1/3^{th}$ of the global entropy value. This value is selected based on our experiments. In practice, threshold value will strongly depend on the application. So an area will switch from a standard to DES simulation when the area's entropy is below $1/3^{th}$ of the global entropy value. In the section below we present the results on the computational cost of the simulation after applying entropy-based transformations.

4.4 Results

In this section we verify the impact of our entropy-based approximation strategy on both the computational cost and the validity of the traffic simulation use case. The setup is similar to the results presented in section 4.2.3, except this scenario does perform transformations to event-driven areas based on the entropy measures introduced in the previous section.

Transforming time-based simulation areas to event-driven areas

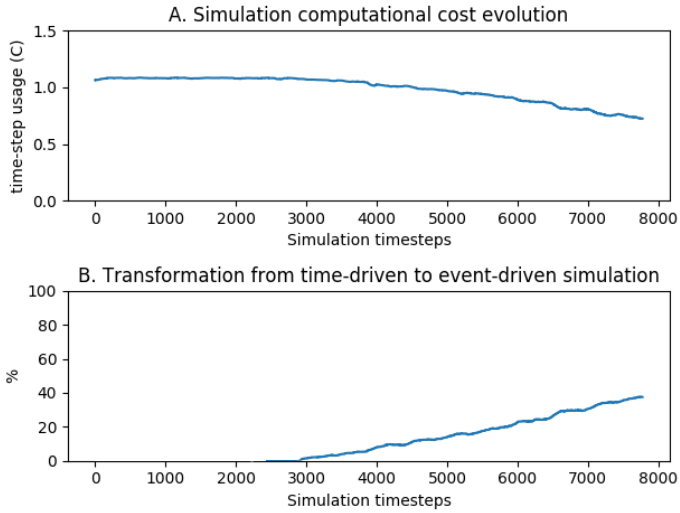

Figure 4.7: Evolution of the conversion of time-driven agents to event-driven approximations

Figure 4.7.A shows the computational cost over time of the second scenario. Figure 4.7.B shows the evolution of standard areas being transformed to DES areas over time. As can be seen at the beginning of the simulation all areas are running in the Agent-Based formalism. However, after about 2000 timesteps the simulation has gathered enough measurements and starts to slowly transform areas to DES areas. Note that this only occurs when enough measurements are gathered and when the local area entropy is below $1/3^{th}$ of the global entropy value as mentioned in section 4.3. This directly leads to more agents being approximated by the event-driven formalism. As a consequence, the computational cost decreases. After 8000 simulation time-steps, 202 areas switched to the DES formalism. Figure 4.7 B shows the gradual increase, starting from 0 areas switched at the beginning of the simulation to 202 DES areas after ca. 7500 timesteps. By then around 37% of agents are active in the approximated areas, the other 63% remain in the agent-based formalism. Furthermore, the time-step usage ratio C decreases significantly from 1.08 at the beginning of the simulation to .72 after about 8000 timesteps. That is a 33% decrease in computational cost, which can be used to scale up to even more agents. Furthermore, after about 4500 timesteps the computational cost of the simulation is below 1.0 time-step usage ratio, which is considered to be safe.

Impact of approximation strategies on simulation validity

To test the impact of approximating simulation areas on the overall validity of the simulation we compare a random area selection strategy versus an entropy-driven area selection strategy as presented in section 4.3. To quantify the validity we will monitor a global simulation property that we expect to be invariant after multiple runs of different simulations. The invariant property selected for the traffic use case is the average distance driven per car. When no approximations are applied we expect the difference in this average distance to be minimal. However, after approximating certain simulation areas we expect this difference to be larger because we're losing accuracy in individual car speed. The graph in Figure 4.8 shows the average error in distance at each simulation timestep for all active agents. The random transformation strategy is our baseline, it

approximates random simulation areas with a discrete event area simulation. The reason we selected this as the baseline is that there are no other techniques described in literature that can be used in the context of this problem. The entropy based transformation strategy only approximates simulation areas with a low entropy value as described in section 4.3. After about 2000 timesteps the first areas are starting to be approximated. After 8000 timesteps the error in distance of the random strategy is significantly larger compared to the entropy based strategy. Therefore, we can conclude that the validity of the simulation is substantially larger when we use the entropy-driven strategy compared to a random uninformed strategy.
Note that for each traffic simulation the routes driven by the agents are deterministic, but the speeds of the agents are not. The speed is sampled from a random normal distribution with a mean that corresponds to the speed limit of the road the car is driving on, as shown in Figure 4.4. Therefore, the distance driven by each car will be slightly different for each run of the simulation. In order to only visualize the average error introduced by the random approximation strategy and the entropy approximation strategy, we subtracted the average error of simulation runs that didn't approximate any areas. This explains why the strategy and entropy graphs in Figure 4.8 are not exactly zero before 2000 timesteps, as would expected in a fully deterministic simulation run.

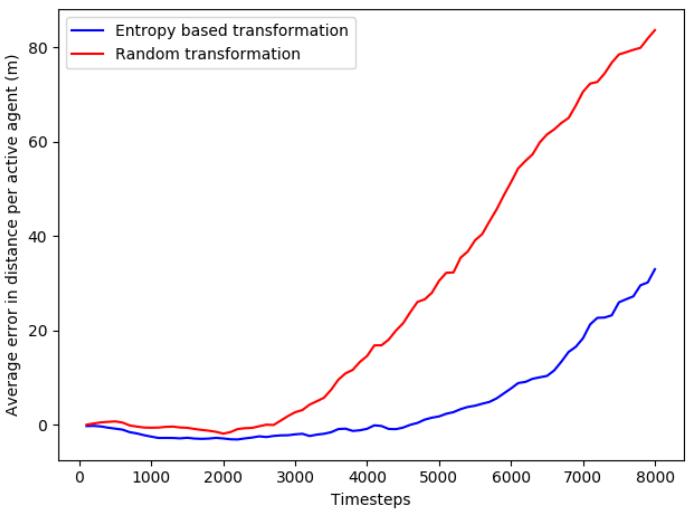

Figure 4.8: Approximation error after a random and entropy-driven transformation strategy - average of multiple runs of various simulations.

4.5 Discussion

In this section we present the proposed method in a more rigorous way. This analysis was added to situate how this work and its contributions compare to the related work published in the community of theory of simulation and modeling. From a more theoretical point of view we can consider the assumptions, guarantees and constraints that are inherent in individual simulation models or agents to be part of a frame. An experimental frame is defined by Zeigler et al. as a specification of the conditions under

which the system is observed or experimented with [6]. A frame provides the necessary contextual information for a model. A frame specifies where the model is valid and the results are considered valid [72] [73]. In our work, our simulation model is a composition of different sub-models. The sub-models all have specific frames to denote the validity. The composition also has a frame to specify for which properties this is a valid model. When a sub-model is replaced by another model with the same frame or a *bigger* frame (a frame with a model that is valid in a broader context, and gives more accurate results), the composition is still valid. This is because the properties in the sub-model on which the composition relies are at the same approximation (or better). However, this is not what we consider in our work. We expect that the behavior after the transformation to a more approximated model ($M_{LA} => M_{HA}$), to be similar (within given boundaries) and is included under the same frame of the overall composed model.

In Figure 4.9 the difference in model behavior after transformation is represented by a performance distance value D, which is the difference between two performance values (that result in the property) P and P'. Ideally, this distance is minimal.

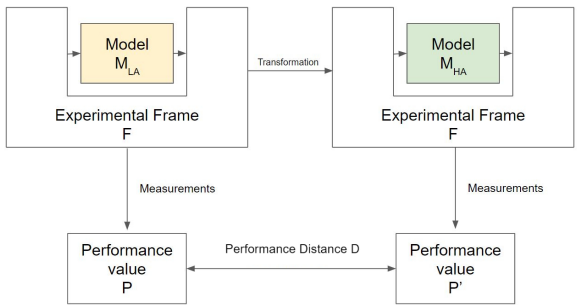

Figure 4.9: Model transformations and experimental frames: keeping track of the validity of a model using experimental frames and performance values

We can more explicitly reason about this by looking into the different properties of the sub-model and composed model. We use the framework proposed by Barroca et al [74] to look at properties of models. A model defines its meaning through a semantic mapping, notated as [[.]]. With simulation this usually results in a set of traces. These traces are related to certain performance values (e.g. the speed of a specific car on a certain road section). These performance values are translated to the ontological domain where they are transformed to a Boolean value (e.g. no traffic congestion exists in the city). The ontological domain consists of the properties we are interested in.

When reasoning about the sub-models of our simulation, we need to be aware of the properties of interest of the sub-models on the properties of the composed model. Figure 4.10 shows this: both the sub-model M_{LA} and its approximation M_{HA} have a semantic mapping to its corresponding performance values as illustrated in Figure 4.9. Both of these performance values (P and P') need to hold in the ontological domain in order for the composed model and its simulation to be considered valid.

The previous allows us (in theory) to explicitly reason about the distance function required for the two models to still be valid. This can be used to our advantage as mostly more approximate models are usually better with respect to computational cost.

We thus need to find a trade-off where the computational cost is much better but at the same time the distance value D is within the bound of the validity so that the global simulation remains within acceptable boundaries.

The approach that we present in our work replaces the use of this explicit reasoning over performance distance and ontological properties with an entropy measure over the performance values needed by the overall simulation. It is a pragmatic metric and we assume that the representation of amount of information is indirectly related to the validity of the model. A model that does not produce a lot of information with respect to a certain performance value, will also not have a big difference in the distance function between a more or less approximate model.

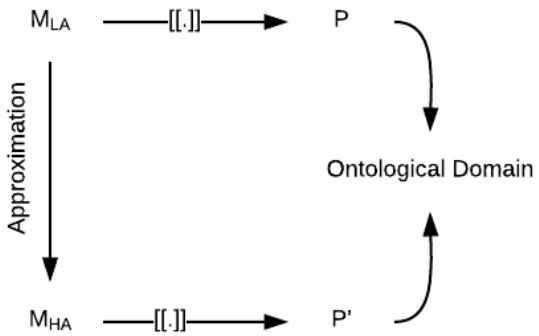

Figure 4.10: Semantic transformations to be evaluated in ontological domain

However, entropy is limited as a metric because it only represents a statistical measure which loses domain-related context about the original model. Also, our experiments were limited to unimodal distributions, which is unrealistic in practice. Many models can only be represented by a multimodal distribution. For example, the unimodel highway velocity distribution presented in our use case will in many cases only be valid outside rush-hour when there are no traffic jams. When we want to incorporate rush-hour in the highway distribution the outcome will become a multimodal distribution with a higher variation. The measured entropy of this distribution will be much higher, and as a result of this our proposed method will not be effective. Either, the entropy will be too high to move to another model, or the model will not produce correct results with respect to its properties (e.g. on the highway it will let the cars drive around 60 km/h instead of 20 km/h during rush hour or 120 km/h outside rush hours). Instead we could measure entropy-levels based on context-information. For example, when we make a distinction between the highway velocity distributions based on time our entropy value will become valid again. This will then result in a unimodal distribution at rush hour and a unimodel distribution after rush hour, opposed to a multimodal distribution when time is not taken into account.

4.6 Conclusion

In this chapter, we presented a method to opportunistically increase abstraction levels of low-information areas to decrease the computational cost of our simulation. Our hypothesis is that we can use entropy as a metric to identify these low-information areas and that the global simulation behavior is less sensitive to noise in these low-information areas. This makes these areas ideal to approximate by a less computationally intensive proxy while maintaining the overall simulation behavior. Our experiment shows that we're able to reduce the computational cost by about 33% in our use case and that we're able to reduce the global approximation error when we opportunistically transform simulation areas with a low entropy value. This highlights the potential of our approach and it shows that our technique succeeds in reducing the computational cost while maintaining the validity of the simulation to a certain level. In the next chapter, we will expand on this work, by developing a more realistic, state-of-the-art traffic simulation scenario and work towards more measurable techniques to evaluate the model validity after approximation. Furthermore, we will implement the MAPE-K paradigm presented in the previous chapter into the simulator architecture to allow a more generic implementation of adaptive approximation behavior.

Chapter 5

Dynamic model abstraction

As we've seen throughout this thesis, the scale of agent-based simulations is limited by the availability of computational resources. Especially in the case of traffic simulations, the limited amount of computational resources leads to the fact that the simulation is either slow or that the amount of agents is limited. As a consequence, most commercial traffic simulators reduce the amount of detail of their simulation models [75]. Traffic simulation can then be modeled at the macro or meso level. At the macro level traffic is simulated as a flow propagating through a street network. As a result, the notion of an individual car disappears. This level of abstraction is often most appropriate for non-urban simulations, where the interactions of individual cars are less important. At the meso level, often groups of cars are simulated together. The car properties, regarding speed or accelerations of spatially nearby cars could be shared instead of being calculated for each individual car.

Choosing the most appropriate level of abstraction is a difficult task. It highly depends on the context and the scale of the experiment, the availability of computational resources, and the required accuracy of the models. For example, a macro-level traffic simulation will be most appropriate to simulate highways, whereas a micro-level traffic simulation is most appropriate in urban environments. As a consequence, most state-of-the-art solutions rely on a multi-level simulation when large environments need to be simulated [76]. It allows to statically combine multiple levels of abstraction. This however limits the flexibility during the simulation as the most appropriate level of abstraction for a certain region often changes over time. A sub-optimal static selection of abstraction levels could therefore lead to either under- or over-utilization of the computational resources or an unnecessary amount of inaccuracies introduced by a higher level of abstraction than needed.

In this chapter, we look at tackling this lack of flexibility introduced in multi-level agent based traffic simulation. We propose a framework that allows dynamic switching of abstraction levels in a multi-level, more specifically a micro and meso level, traffic simulator. We present the architecture of a state-of-the-art traffic simulator that includes the dynamic switching of abstraction levels. Our proposed method is inspired by related work in the theory of simulation and modeling regarding model abstraction and by the work of (dynamic) multi-level traffic simulation described in section 6.3. We validate the

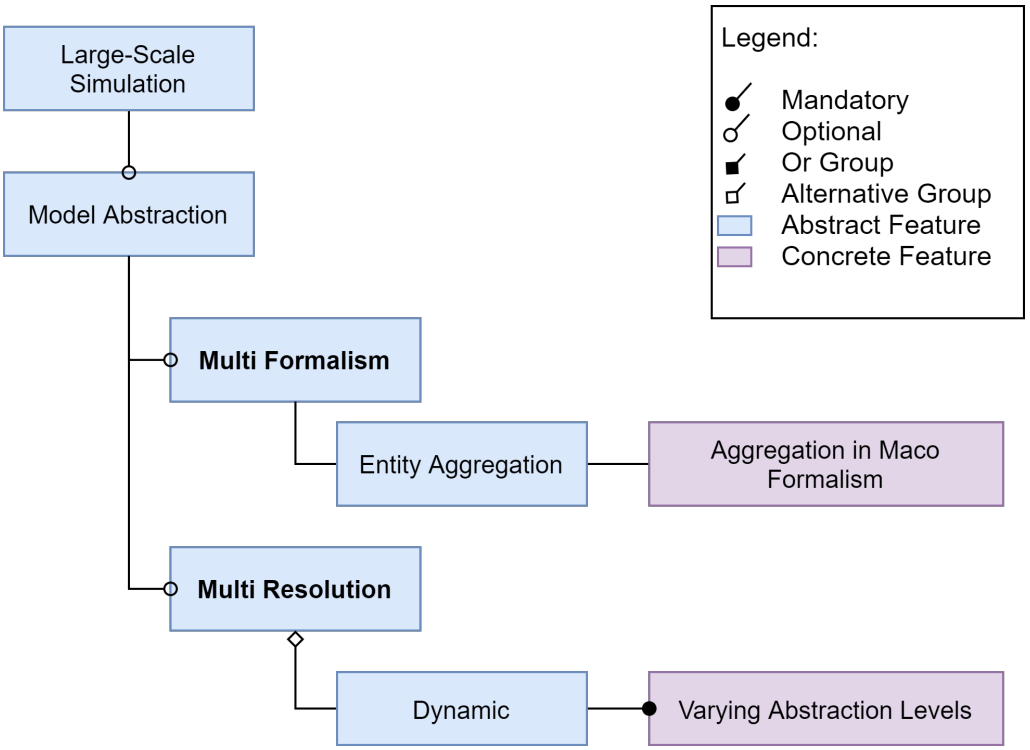

Figure 5.1: Feature Diagram - Contributions Chapter 5

accuracy and validity of our method on a realistic urban traffic simulation. We show that our method can dynamically balance the trade-off between simulation accuracy and computational cost. We do this by leveraging the concept of experimental frames [77], this helps to determine if the context of a model is compatible with the model. When this not the case we can switch models to improve accuracy while taking computational cost into account.
The main contribution of this chapter is the methodology that we present to allow dynamic switching combined with a detailed discussion of how this methodology fits into a state-of-the-art multi-level traffic simulation architecture. We discuss this simulation architecture, together with the developed traffic models in detail. Furthermore, we emphasize the reusability of the implementation of our method within the simulation architecture. The feature diagram in Figure 5.1 shows that our contributions in this chapter are mainly in the Multi Formalism area, where we focus on entity aggregation by switching multiple agents in the Agent Based Formalism to an explicit macro model. Furthermore, we apply a dynamic multi-resolution approach where abstraction levels can vary during simulation run-time. In this chapter, we expand on the previous, but instead of relying on an opportunistic approach where both approximation and area selection are performed automatically we now take a different approach. We switch to an explicit, state-of-the-art formalism and the decision where and when to switch is more driven by domain knowledge. Furthermore, we present a method to encode this domain knowledge into the simulation engine.

In the first section, we discuss the related work regarding multi-level abstraction and model validity, multi-level traffic simulation, and adaptivity in agent-based traffic simulation. In section 5.2, we present a use case that will be referred to throughout the chapter and which defines the requirements needed for our simulator architecture. This simulation architecture is introduced in section 5.3. We also provide more background to the meso and micro traffic models that we use in our example. In section 5.4, we present our method to allow dynamic switching in the simulator. Finally, in section 5.5, we evaluate the proposed method and simulation architecture.

5.1 Background

In this section, we discuss the broad area of related research. The contributions discussed in this chapter cover a wide area of topics. To keep things organized, we split this section in multiple subsections, each discussing the related work of a certain topic. In subsection 5.1.1 we provide background about multi-level abstraction in simulation. In subsection 5.1.2, we discuss how we can define and evaluate the validity of a model formalism in a given context. In subsection 5.1.3, we discuss how multiple levels of model abstractions can be combined in the context of traffic simulation. Finally, in subsection 5.1.4 we review techniques discussed in literature that allow to dynamically switch between these abstraction levels.

5.1.1 Multi-level abstraction

Choosing the appropriate level of abstraction of a simulation model is one of the most difficult challenges that occur during modeling and simulation. The level of abstraction defines the amount of information that a model contains. The most appropriate level depends on the context of the simulation and the questions that the model developer is trying to answer. In an ideal scenario, each simulation model contains the maximal amount of information. In other words, the model is represented at the highest possible fidelity or at the lowest level of abstraction. But this leads to a significant increase in modeling and validation effort. Furthermore, the impact on the computational cost cannot be neglected. In practice, we want to avoid including unnecessary details in the model, to both make the model analysis process less complex [78] and to reduce the computational complexity. Therefore, abstraction is a core part of the modeling process. During the modeling process, a modeler chooses a fixed abstraction level that seems most relevant. In many scenarios, a single abstraction level is however not sufficient. During the run time of the simulation, the context that the model operates in changes constantly. A change of context might lead to situations where the abstraction level of a model is not ideal. For example, a model that does not describe a car's conflict behavior on traffic intersections will not perform well at busy urban roads. This leads to an increase in inaccuracies of the simulation results. To overcome this problem, a significant amount of research has been published in the modeling and simulation community regarding multi-level abstraction. The idea of multi-level abstraction is to use multiple models at various abstraction levels and/or formalisms. Each abstraction level can then be used depending on its operating context. Morvan et al. summarizes the three major challenges that need to be solved during multi-level agent-based modeling [79]:

- Modeling of interactions between abstraction levels
- Coupling of heterogeneous models
- The dynamic adaptation of the level of detail of simulations to save computational resources or use the best available model in a given context

A large amount of literature is available regarding the first two bullets. But limited work has been done regarding the last bullet, especially in the traffic simulation community. In this work, we attempt to fill this gap by building a framework that allows for efficient dynamic switching of abstraction levels by taking computational cost and accuracy into account. We discuss the related work of this topic in section 5.1.4. We will focus in this chapter on integrating two models, a micro-level traffic model and a meso-level traffic model. For the integration of these models we apply one of the multi-level design patterns discussed by Mathieu et al. [80]. In their work, they present four patterns to integrate simulation models at multiple levels of abstraction. One of them, the puppeteer pattern, offers a solution to keep track of micro-level state while increasing abstraction levels. This pattern freezes the micro-level state when agents switch from the micro-level to a meso-level. Meanwhile, at the meso-level, a hierarchically higher level agent, a 'puppeteer' agent, takes control of the micro level agents and adapts the state of these agents simultaneously. The micro-level behavior is thus delegated to the puppeteer agent. This will reduce the computational cost of the model, but we will lose the ability to model the individual behavior of agents based on their observation. This will lead to a reduction in accuracy. When we switch back from the meso-level to the micro-level the frozen state of the micro-level agents is restored, and they will regain the ability to control their behavior.

5.1.2 Model validity and experimental frames

To decide from a validity point of view which level of abstraction is, most appropriate in a given context we will leverage the framework of modeling and simulation (M&S) as described by Zeigler et al. [77]. In their work, they introduce the concept of experimental frames to specify the conditions under which a model can be observed in a given context. We will leverage this concept in this chapter to define the levels of the validity of a model. But before we discuss this idea it is important to understand some of the related definitions and concepts introduced in Zeigler's M&S framework.
They define the concept of a system as "the real or virtual environment that we are interested in modeling". An Experimental Frame (EF) is then a specification of the conditions under which the system is observed or experimented with. An EF can be seen as a definition of the variables that specify a system. The EF can be specified at multiple levels of abstractions. In the context of traffic simulation, one EF can include the driver acceleration, whereas another might include only street-level density and velocity characteristics.
The M&S Framework defines a model as a specification of a system.
The concept of model validity can then be defined as the relation between a model, a system, and its experimental frame. A valid model is a model where, for all the experiments possible within the experimental frame, the behavior of the model and system agree within an acceptable tolerance.

What makes this framework interesting within the concept of this work, is that for our experiments we can define an experimental frame for a system. And multiple models, at various levels of abstraction, are able to comply according to the experimental frame. Taking efficiency into account, it makes more sense to leverage the computationally less complex model from the set of available models that comply to the experimental frame. However, when a given model is not able to adhere to the experimental frame, we will need to switch to another model that does comply, to maintain the validity.
This framework provides the context to decide when we should select a model from a set of given models, based on the pre-defined experimental frame. We will leverage this idea to dynamically decide at run-time when we can switch models, to reduce the computational cost, while maintaining the validity of the simulation.

5.1.3 Multi-level traffic simulation

In the previous section, we discussed the general idea of using multi-level abstractions. In this section, we look in more detail at how these ideas are being applied in the context of traffic simulation. Traffic simulators are used to model and simulate the dynamics of complex traffic systems. They can typically be categorized based on their abstraction level. These levels range from:

- Micro level: This simulation model describes the detailed behavior of a car at the individual driver level. It is used to capture detailed behavior and interactions of cars. This type of model typically requires a significant calibration effort, since the amount of parameters is large compared to other abstraction levels. Because of this, and because of the computational cost, this level is most often used on small-scale simulations. State-of-the-art examples of these models include Car following models such as the Intelligent Driver Model (IDM) [62] and lane changing model such as the MOBIL model (Minimizing Overall Braking Induced By Lane Change) [63].
- Macro level: The macro-level simulator models traffic at a continuous level. These models are often inspired by flow theory. These models don't include individual car entities.
- Meso level: This level is somewhere in between the previous levels. It keeps track of individual models but as aggregates. For example, they use speed-density relationships at street sections. We use this model as our higher abstraction level in our experiments. It can be used on urban environments and allows for the puppeteer pattern to be implemented since the individual entities delegate their behavior to a single entity, the meso model, which is hierarchically higher.

Each of these models has its limitations and trade-offs. The micro-level model can only be used at a limited scale and the meso- and macro-model are not able to simulate the detailed behavior of individual cars. To better balance these limitations and advantages a hybrid model can be used. Burghout et al. have described a number of methods to combine these different levels of abstraction in their work [76]. Their methods are however limited to static multi-level traffic simulation. This makes it impossible to check the validity of the models while they are being used. In this work, we will expand on this static multi-level approach and discuss methods to allow dynamic updating of abstraction levels by taking the validity of the model into account given a certain context.

5.1.4 Adaptivity in agent-based traffic simulation

In this section, we discuss the state-of-the-art research regarding dynamic switching of abstraction levels. There is a large body of research in the modeling and simulation community that discusses the problem of model abstraction. More recently, we see an increased interest in the dynamic switching of these abstraction levels.

In recent work, Franceschini et al. present an adaptive abstraction method where they switch abstraction levels of a basic traffic simulation within the same simulation formalism [81]. They use a predefined trigger to decide when to switch between abstraction levels. In earlier work, the author presents a dynamic abstraction simulation that switches between an agent-based formalism and a discrete event formalism [29]. The decision to switch abstraction levels is motivated by statistical analysis of the observed emergent behavior. In this work we expand on this and develop a more rigorous framework to decide when and where to switch between abstraction levels.
In other related work, we see multiple positional papers that aim at introducing a dynamic multi-level traffic simulation method. Bouha et al. [82] present a conceptual framework that allows dynamic switching between traffic simulation formalisms. They present a simulation framework that supportsIDM Micro-simulation and a Metanet macro-level simulation model [83]. The framework allows switching between those levels, and describe how to integrate both levels in a single simulator. The experiments focus on a highway environment, the results are however limited as this paper was presented as a positional paper. Haman et al. also present a method that allows dynamic switching between micro- and macro-level traffic simulation [84]. Their method is inspired by the concept of a Holonic organizational meta-model [85][86] leveraged in the work of Gaud et al. which allows dynamic multi-level switching between abstractions applied to a pedestrian simulation [87]. Gaud et al. present a generic method to switch abstraction levels based on indicators that estimate the deviation of a simulation model in comparison with the most accurate level.
In summary, most of the related literature, regarding dynamic switching attempts to present a solution to decide when to switch between abstraction levels and how to integrate multiple abstraction levels. Furthermore, the experiments presented in the literature are limited in their complexity and scale. We present a framework that determines to switch abstraction levels based on the validity of the experimental frames applied to a realistic, large-scale use-case. Furthermore, we attempt to implement this as part of our simulator architecture in a clear and standardized way. Regarding the transition from a higher abstraction level to a lower abstraction level and vice versa, we aim to leverage the puppeteer pattern discussed in section 5.1.1.

5.2 Use case

We want to combine the ideas of dynamic multi-level traffic simulation to balance between validity and computational cost. We want to bring these ideas, based on the theoretic foundations discussed in section 6.3, into a realistic simulation. To do this we define a use case scenario in this section, that we will refer to throughout the remainder of the chapter. The goal of the use case is to come up with a realistic, large-scale multi-level traffic simulation example where we can show that dynamic switching of abstraction levels will directly benefit the computational cost and overall validity compared to traditional

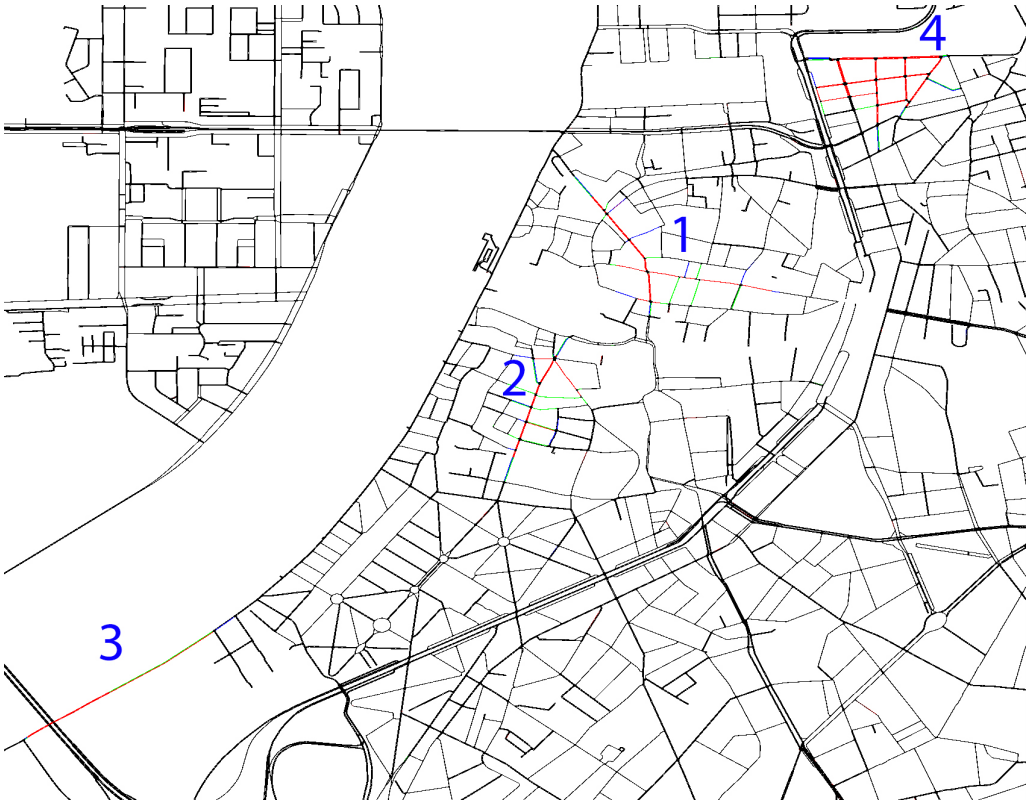

Figure 5.2: Multi-level traffic simulation - colored regions can dynamically switch abstraction levels. This is a screenshot of the simulation framework presented in this work.

single or static multi-level traffic simulators.
The simulation simulates urban traffic in a city center. The cars are simulated primarily at the micro-level. The micro-level simulation will be the ground truth, the behavior observed at this level is used to calibrate and validate levels that contain less detailed behavior. This allows us to objectively compare the observed data between various levels. An example environment is represented in Figure 5.2. This exact environment is used in experiment 2 presented in section 5.5.2.

To simulate realistic emergent traffic patterns, the micro-level car agents have predefined routes based on their origin and destination zones. The cars need to comply to traffic regulations, have different driving characteristics, and can switch lanes. The simulation includes traffic lights leading to interesting emergent behavior.
Besides the micro-level, the simulation use case also supports a meso-level model. The meso-level model is solely based on a speed-density relationship. In other words, the average velocity observed at a street section directly depends on the observed traffic density of the street. Each street that is part of the meso-simulation is calibrated on raw velocity and density data that is observed in the baseline micro-level simulation. The meso-model does not directly take traffic priority rules into account at intersections and does not directly take traffic light states into account. This is an important difference between the

two abstraction levels. Because as a result, the validity of the meso model decreases in contexts where traffic lights or priority rules have an impact on the observed behavior. This context-dependent information can be specified in the experimental frames. With these specifications we keep track of when the model is valid or not in a specific context. In this use case, we leverage the experimental frames to decide during the simulation when one of the two models will be selected. In the next chapter, we discuss the implementation of our custom simulation framework, the use case specifications and the constraints discussed above, and how these ideas can be implemented in the simulation engine. Finally, we validate these ideas in two experiments presented in section 5.5.

5.3 Dynamic multi-level simulation framework

In order to execute our dynamic multi-abstraction use case described in the previous section we need a simulation framework that adheres to the following requirements:

1. ABS methodology: from a modeling perspective, we find that traffic can be best described using the ABS paradigm. Where an individual agent represents a single car at the micro-level or a single street at the meso-level. Each agent keeps track of its own state, which it can update at discrete time steps. It has to ability to observe and interact with its surrounding environment and with nearby agents.

2. Access to the simulation engine to be able to add custom functionality to switch abstraction levels and manage transitions from one abstraction level to the other.

3. Ability to import geographical data that fully describes a road network. This includes the number of lanes on a street, their direction, the speed limit, the allowed turns, the possible priority conflicts at junctions, and so on.

4. A routing module that is able to determine shortest routes in a road network. These routes are used by the car agents to navigate throughout the city environment.

5. A state-of-the-art micro-level traffic model that calculates driving parameters such as acceleration based on the driving behavior and surrounding cars. Furthermore, lane changing behavior is required.

6. A state-of-the-art meso-level traffic model that is computationally less complex compared to the micro-level model. Furthermore, the limitations of the model should be clear. We exploit these aspects in order to evaluate the performance of our model using the predefined experimental frames.

7. We need to have detailed logging information regarding step duration in order to measure performance bottlenecks.

8. We need the ability to collect aggregated statistics at both the car- and road-level. This data includes total trip time, the average velocity at a road segment, average density and average flow. We need this to calibrate meso models based on a micro simulation baseline, and to compare validity between meso models.

9. Visualization engine: in order to debug, test and evaluate the simulator a visualization engine will be required.

10. High performance: in order to reach city-level scale during our experiments, we need to be able to simulate a high amount of agents at acceptable durations.

We were not able to find an open-source simulator that fits all of these criteria. There are many commercial micro-level traffic simulators such as Aimsum [75] and Vissim [88]. Although that these simulators adhere to most of our criteria, they even support a static multi-level traffic simulation functionality, they lack the ability to access and adapt the simulation core which is a hard requirement in order to implement dynamic multi-level functionality.

There are however some very good open-source alternatives. The most famous open source micro-traffic simulator is definitely Sumo [89]. Sumo is a powerful microscopic simulator that is fully open-source. It does not support any multi-level functionality and does not have any built-in macro-traffic models. Because of this, we decided to build a custom traffic simulator which fully adheres to the functionality requirements described above. We decided to leverage the network parser, Netconvert, from Sumo. This parser has the ability to interpret and parse raw Open Street Map data [90]. Open Street Map (OSM) offers open user-generated street maps of the entire world. Netconvert converts this raw OSM data to a human-readable XML based format that stores detailed street information such as lanes, priority rules, traffic light locations and configurations, possible turns, etc.

In the sections below, we dive deeper into the architecture of the simulation framework that we developed.

5.3.1 Custom simulation framework architecture

Figure 5.3 shows the most relevant components that were tailored to the scope of this chapter. We discuss each of these components in more detail in the following subsections.

5.3.2 Simulation core components

The components at the bottom of our layered architecture diagram in Figure 5.3 contain supporting functions to represent a street **environment**. As mentioned in the previous paragraph, we start from raw Open Street Map data that is parsed using the Sumo netconvert engine. We then developed a custom parse, the **Net Parser module**, to interpret this raw data and feed a traffic environment datastructure that can be used throughout the simulation framework. This data structure contains both relevant street meta data such as length, speed limit, geographical shape, lanes etc. But it also describes intersections and the path on these intersections to connect incoming and outgoing steets. It describes the priority rules and possible conflict situation on the intersection. The relationships between roads, lanes, and junctions are described using a directed graph data structure. A road is represented by a node and a junction connection between roads by a directed edge. This enables the car agents to take turn regulations into account. **The routing module** is responsible to calculate the shortest path between a given origin and destination based on its distance. It leverages the directed graph to do this.

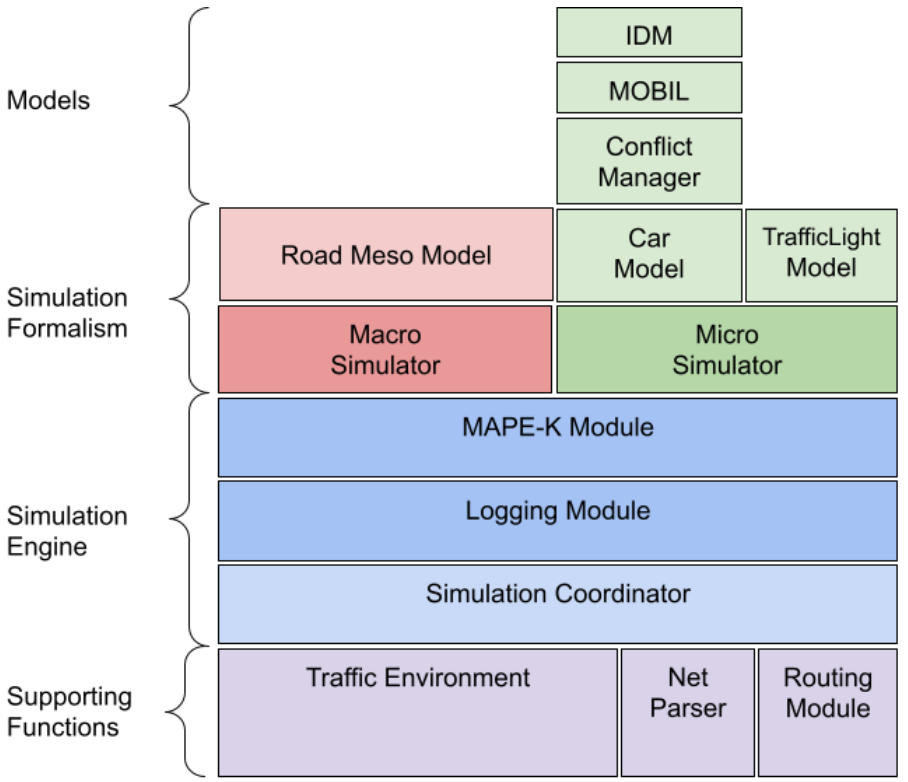

Figure 5.3: Simulation Architecture

The core of the simulation is the **simulation coordinator**. This component is responsible to initialize the various simulators. We distinguish between a micro-level simulator and a macro-level simulator. Both of these simulator entities share the traffic environment data structure. This might lead to conflicts between threads, because each simulator level is executed on a given thread, but this is prevented by the fact that each environment region is controlled by a single simulator entity. Simulation time progresses in discrete timesteps. These timesteps are triggered by the coordinator. When a step is triggered, the coordinator waits until all simulator threads are finished executing their calculations before it triggers the next timestep.
The **logging module** is responsible for collecting road-based statistics related for short periods of time. The variables that we capture are traffic volume, density, flow, and average velocity. All of this data is logged into files.

5.3.3 MAPE-K: Adding dynamic behavior to the simulation engine

The **MAPE-K module** is the core of the dynamic abstraction methodology. MAPE-K is short for Measure, Analyze, Plan, Execute, and Knowledge [91]. Each of these words describes a phase that is part of an iterative loop that gets executed at regular intervals during the simulation execution. The MAPE-K framework is used in many engineering applications to be able to appropriately react to dynamic changes.

In our context, we use this framework in combination with the validity frame specifications to detect when certain regions do or do not adhere to the validity frame. This method has been used in the context of generating dynamic behavior in simulators. It is a clear, application-agnostic method that allows model developers to specify dynamic performance-optimizing strategies without building ad-hoc, non-reusable solutions directly into the simulation core.

During the measure phase, we keep track of raw statistics (average velocity, density, flow, amount of intersection conflicts, etc.). In the analysis phase we aggregate these raw metrics and try to detect certain events. During the planning phase, we try to take steps to react to these events, for example deciding to switch to a given level of abstraction. And finally, during the execution phase, we execute these steps in the simulation environment. In our use case, the MAPE-K loop gets executed every 60 timesteps during the simulation. This resolution is application specific. In the use case presented in this chapter this resolution is sufficient based on our observations.

5.3.4 Micro simulator

The micro simulator is based on the agent-based paradigm. More specifically we implement a structure that is inspired by the MESA simulator [59]. It is a very intuitive modeling framework, that leverages the Object-Oriented Programming and applies it to the Agent-Based Simulation paradigm. Within this structure, we differentiate between four main entities: a simulator, an agent, a model and an environment entity. The simulator entity is responsible for triggering steps. The agent entity describes the behavior of a single agent (e.g. a car or a traffic light), the model contains a collection of an agent class and is responsible to collect statistics on this group of agents. Finally, the environment describes the position of the agents and the relations in space between agents.

Micro models

The car agent represents the state of a single car on a road. It has a source and destination, it determines its route based on a shortest path algorithm which is implemented as part of the routing module. The driving model is based on the IDM as described by Treiber et al. [62]. IDM is a typical car-following model, where the velocity and the acceleration are determined by the velocity of the car in front. The driving characteristics of the car are described given the parameters in Table 5.1. The changes in velocity are calculated given the IDM equations described in Equations 5.1 and 5.2.

$$\frac{dv}{dt} = a(1 - (\frac{v}{v_0})^{\delta} - (\frac{s^*(v, \Delta v)}{s_\alpha})^{2}) \tag{5.1}$$

$$\begin{gathered} \text{with } s^*(v, \Delta v) = s_0 + vT + \frac{v\Delta v}{2 * \sqrt{ab}}, \\ \text{and, with} s_\alpha = v_\alpha - v_{\alpha-1}, \text{ for vehicle } \alpha \\ \text{where } \alpha - 1 \text{ is the vehicle in front of } \alpha. \end{gathered} \tag{5.2}$$

Variable	Description	Unit
l	Vehicle length	m
v_0	Desired Velocity	m/s
a	Maximum acceleration	m/s^2
d	Maximum deceleration	m/s^2
s_0	Minimum distance	m

Table 5.1: Intelligent Driver Model Parameters

Furthermore, a lane-changing model was implemented inspired by the Treiber et al. MOBIL model (Minimizing Overall Braking Induced By Lane Change) [63]. The model takes two criteria into account when deciding to make a lane change. An example scenario is depicted in Figure 5.4. The center car on the right lane can for example decide to change to the left lane. Within the MOBIL model first a safety criterion gets evaluated, this criterion calculates whether the accelerations of the car and its new follower f' does not surpass a safe deceleration limit. Furthermore, from the perspective of the car will be calculated whether its acceleration on the new lane compared to its current acceleration surpasses a predefined margin. When both of the criteria succeed in this test, a lane change event is executed. We added the ability to dynamically add bias to the incentive criterion, which allows cars to change lanes in order to take the correct turn at an intersection.

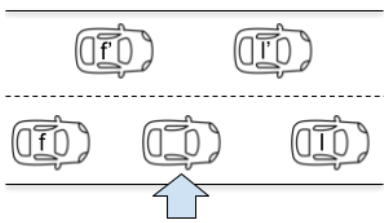

Figure 5.4: Intelligent Driver Model: acceleration of the car is influenced by the acceleration of its surrounding agents

Junction behavior

We expand a bit more into the junction behavior of a car. Because much of the dynamic stop-and-go behavior is caused by junction conflicts. We define a conflict as the situation where two cars approach a junction simultaneously. We modeled the conflict context into the traffic environment by adding to each lane whether there is an upcoming priority road. A car driving on this lane verifies whether an active conflict is nearby. In the case of a conflict, it will gradually decrease acceleration depending on the distance to the upcoming junction. If necessary the car stops right before the junction.
The simulation also supports traffic lights. Similar to the conflict behavior, a car agent will verify whether there is a red or yellow traffic light located on the junction. Consequently, it will stop in front of the junction and wait until the traffic light switches to green.

Furthermore, a model was added to prevent deadlocks. Deadlocks at intersections can occur when there is a large amount of traffic on a priority lane without traffic lights. By default, the cars located on non-priority lanes will endlessly wait. We added a mechanism that, similar to real-life scenarios, cars located at the priority lane will stop, after a while, for "non-priority cars" that are waiting.
An additional technique that could help to prevent deadlocks is Dynamic Traffic Assignment (DTA) [92]. DTA allows car agents to keep track of traffic density on their route. When the traffic density is too high they can decide to pick an alternative route. But we didn't implement it yet in our simulator.

5.3.5 Meso simulator

The meso simulator leverages the same environment model depicted in Figure 5.3. It was also implemented using an agent-based paradigm, but in this scenario, the primitive agent is a road segment, instead of a car. The road segment is the puppeteer that takes control of the incoming car agents. Time is modeled discrete and it progresses simultaneously with the micro simulator. The meso model behavior is based on a density and velocity relationship.

Macro model

Most state-of-the-art macro traffic simulation models are inspired by the LWR (Lighthill-Whitham-Richards) model [93], where traffic behavior is represented by a continuous equation that models the relationship between traffic density and traffic flow [94]. Derived from the LWR theory is the fundamental diagram. This diagram represents a curve relating observed densities to observed flows [95] as shown in Figure 5.5. Traffic density defines the number of vehicles per unit distance (vehicles/km). Traffic flow defines the number of vehicles that pass a discrete location per unit of time (vehicles/h). Most macro models construct such a fundamental diagram based on observations. These observations can directly originate from real-life road measurements such as traffic loops. For each of the road segments used in the simulator, a fundamental diagram can be created and leveraged during the simulation. Figure 5.5 shows the three major phases of a fundamental diagram. The colored dots represent raw measurements, the lines represent a linear equation that fits the part of the raw measurements.

1. Free flow phase: During this phase traffic density is building up, and as a result traffic flow increases linearly.

2. Critical density phase: the maximum density for a given road is reached. This is when congestion starts.

3. Congestion phase: As a result of the congested state of the road, the traffic flow decreases steadily when the density increases.

During calibration, each of these phases needs to be identified and the parameters of the equations representing the behavior of each of these phases need to be found based on the raw measurements. Various techniques can be used to perform this calibration [95], but this is beyond the scope of this work.

We initially decided to also leverage this fundamental diagram as basis to describe the behavior of our meso model. But as part of our observation, we saw that it was

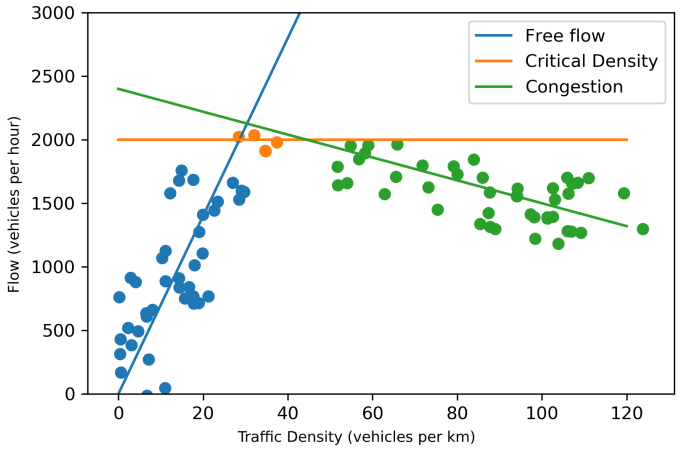

Figure 5.5: Fundamental Diagram of traffic behavior: relationship between Traffic Density and Velocity. (dummy data)

hard to accurately identify the three phases in the raw density and flow measurements of an urban traffic environment. We noticed that the fundamental diagram is mainly used to model highway traffic and that might not be ideal to model urban traffic. Therefore, we decide to use a simplified model, which directly models the relationship between traffic density and traffic velocity. In our observations, we saw that, in an urban context, there tend to be a clear inverse relationship between density and velocity. Typically, the velocity tended to decrease with increased density. In this model, we can use a single equation to model this relationship. The parameters of the equations tended to vary a lot depending on the road type and on the location of the road within the environment. For example at busy intersections, an increase in density resulted in a stronger decrease in velocity.

Model calibration

The model that we use in this simulation is based on a density - velocity relationship. Each individual road segment in our simulation has its dedicated model. The model gets calibrated based on raw density and velocity measurements. We use a polynomial regression technique to fit the model onto the raw measurements. An example is demonstrated in Figures 5.6, 5.7 and 5.8. In order to prevent overfitting, the degree of the polynomial is changed depending on the standard deviation. As can be seen in Figures 5.6, 5.7 and 5.8 the model can be either a zeroth-, first- or third-degree polynomial. Fitting to orders higher than the third degree did not add additional accuracy based on our observations. Furthermore, we added constraints to the model. We only use the model between the range of observed densities. For densities below or above the observed density, the model will return a fixed value, which is the corresponding velocity value for the respective lowest or highest observed density. Finally, we define a maximum and minimum velocity for each model.

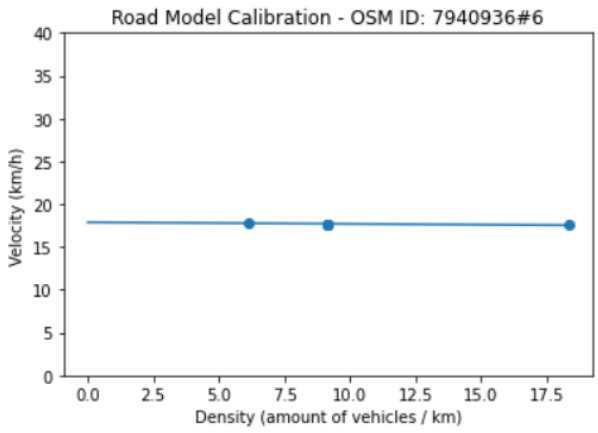

Figure 5.6: Zeroth degree polynomial

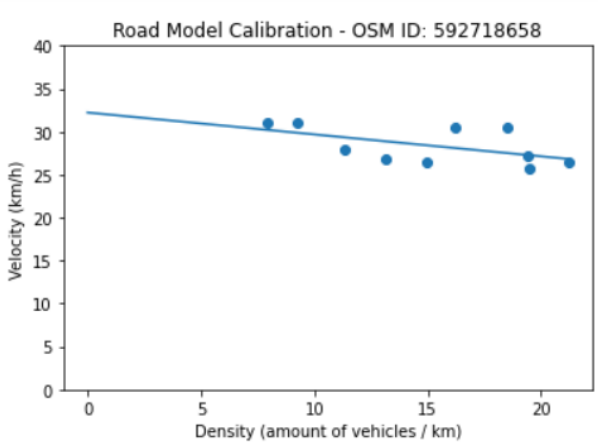

Figure 5.7: First degree polynomial

Meso model

The macro model from the previous section forms the core of our meso model. We extend the meso model with additional functionality that allows the model to keep track of individual cars. That is why we refer to the model as a meso-level model instead of a macro-level model. It functions according to the puppeteer pattern explained in section 5.1.1.

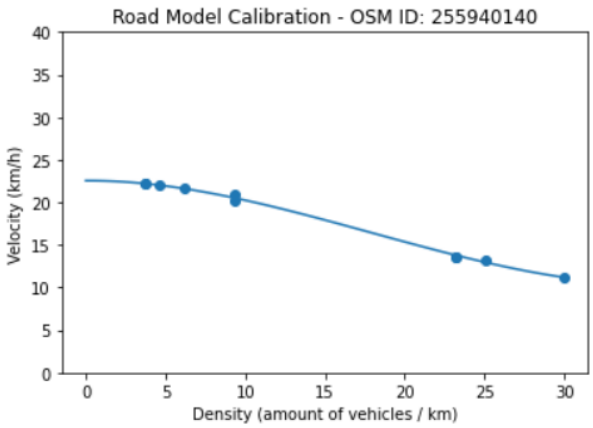

Figure 5.8: Third degree polynomial

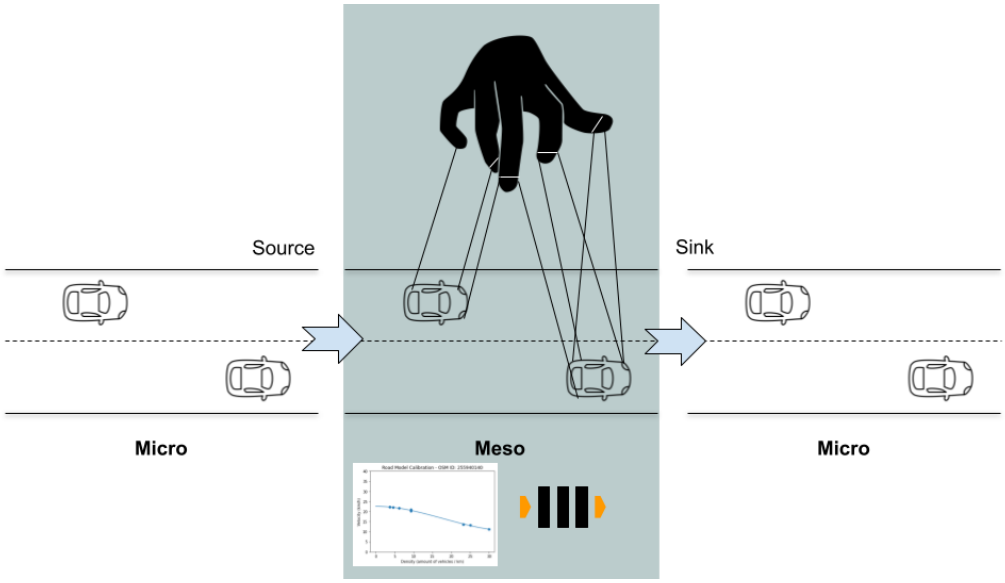

Figure 5.9: Meso Model: According to puppeteer pattern - micro agents enter the meso model and are lose the control over their individual state to the overarching meso-level agent

The structure of the model is depicted in Figure 5.9. First, all roads in the environment are split into discrete parts of a predefined maximum distance. We then define sources and sinks where cars can enter and leave. The sinks and sources also allow for direct integration with the micro level simulation. The model consists of an internal priority queue data structure that keeps track of when a car should leave the road segment. This travel duration gets calculated using the calibrated macro-model for the specific road segment. When a car enters, the density of the road segment is increased, and based on this density the average velocity and the car's travel duration are determined. Based on the travel duration, the car's exit time is calculated and is then added to the

priority queue. When the simulation time exceeds the car's exit time, the car is removed from the priority queue and then added to the queue of the next road segment or added to the sink.
The route of the car is taken into account, and at intersections, different cars can take different turns. Compared to the micro-level simulation, conflicts situations and traffic lights are not included in the meso-level simulation. As a result, the meso-level is not an appropriate abstraction level to simulate behavior at busy intersections. Instead, the model can best be used to simulate busy roads without too many conflict situations.

5.4 Dynamic switching between levels

In the previous sections, we showed the implementation of a micro and a meso-level traffic simulator. We also added the functionality to integrate both abstraction levels in a multi-level simulation. Each part of the road environment will be handled by a single dedicated simulator entity. The abstraction of the simulation entity (meso or micro) can vary as depicted in Figure 5.9.

Each car agent in the simulation is aware of which simulator it is operating. When a car enters a road that is not part of its current simulator, the simulator creates a migration request. This migration request is handled by the simulation coordinator, which will, at the end of a time step migrate the agents between simulators. We will discuss below how migration between the various abstraction levels occurs:

5.4.1 Transitioning between levels

Micro to meso: To switch from micro to meso-level we use the puppeteer pattern presented in section 5.1.1. We keep track of individual entities and their state, but the road segment serves as a master entity that controls the behavior of the individual entities. When a car enters a meso zone, the micro-level state remains in memory but is not used in meso-level simulation. Instead, a reference to the vehicle agent state is created in the meso-simulator while the micro-level reference is deleted from the micro simulator. Also, a reference to the routing information of the vehicle is copied to the meso-simulator. The current velocity and acceleration of the car are not used in the meso-level simulation. The velocity is fully determined by the density - velocity function which is part of the meso-model, as explained in section 5.3.5.

Meso to micro: Similar to the micro-to-meso transition, a migration request is created when a car leaves a meso-zone. The meso-simulator has kept a reference to the internal state of the original micro agent and this information is recovered in the micro simulator.

5.4.2 Incentive for dynamic switching of abstraction levels

The simulator coordinator supports the functionality to dynamically switch the simulator that is responsible for a region of the environment. This allows us to dynamically detect when and where a region should be either simulated at the micro-level or at the meso-level. And based on this we can balance between computational cost and validity. We assume in the use case that the micro-level is the 'reference' or baseline level, and that the meso level is less accurate. The decision to switch is implemented by the use

of experimental frames. The experimental frame defines the operating context of our meso-level model. When we observe that the current context does not comply with what is defined in the experimental frame we will trigger a switch of abstraction levels, more specifically from meso to micro.
The meso-model is not able to simulate conflicts at intersections. As a result, the validity at busy intersections is not guaranteed. As a result the experimental frame of the meso-model is limited to roads with a limited amount of conflicts at intersections. In our simulator, we keep track of whether the context of the meso-simulation complies to the experimental frame as defined above. We do this as part of the MAPE-K loop, which was discussed in section 5.3.3. During the measure phase, we specifically keep track of the number of conflicts that occur in a sixty-second period. During the analysis phase, we evaluate whether a threshold is exceeded. When this is the case a plan is formulated to move the region from meso to micro. Consequently, this gets executed by the simulation coordinator between simulation time steps. As a result, all agent entities managed by the simulator are migrated to the new micro simulator and are re-instantiated at their corresponding locations. The accelerations and velocities are directly calculated again by the micro simulator based on the IDM model.

5.5 Results

We test the dynamic micro to meso switching scenario in a small contained zone in experiment 1. In experiment 2 we present a larger experiment, with multiple abstraction zones. In each of the experiments, we keep track of the computational cost, which is measured in ms per time-step. And we keep also track of the validity error. This is measured by comparing the RMSE (Root Mean Square Error) between the velocity observed at the street level in the meso simuation, compared to the baseline simulated at the micro-level. The experiments were executed on a desktop computer with an Intel i7-6700K - 4Ghz CPU and 16GB of memory.

5.5.1 Experiment 1: Dynamic switching between micro and meso

In this experiment, we set up a scenario of a single main road and traffic coming from north, south, east, and west. We initiate the simulation in a hybrid setup, where the main road is at meso level and the incoming roads at micro level. We measure the validity of the meso level using the Root Mean Square Error (RMSE) of the velocity, where we compare the observed velocity to the baseline velocity. The baseline is a fully micro-level simulation. The meso model is calibrated on the baseline data. We start the first 300 seconds of the simulation with a steady inflow of cars. After 300 seconds we double the inflow. As a result, there will be a large increase of conflicts at intersections on the main road.
Figure 5.10 shows the computational cost of our experiment and Figure 5.11 shows the validity or the RMSE error. We measure the computational cost based on the duration of a single simulation step. We refer to this as the step duration, measured in milliseconds. The baseline scenario is a fully micro-level simulation. The static meso and micro, is a classic hybrid simulation scenario where the abstraction level of each region remains constant. Finally, the dynamic meso and micro scenario allows the simulation coordinator

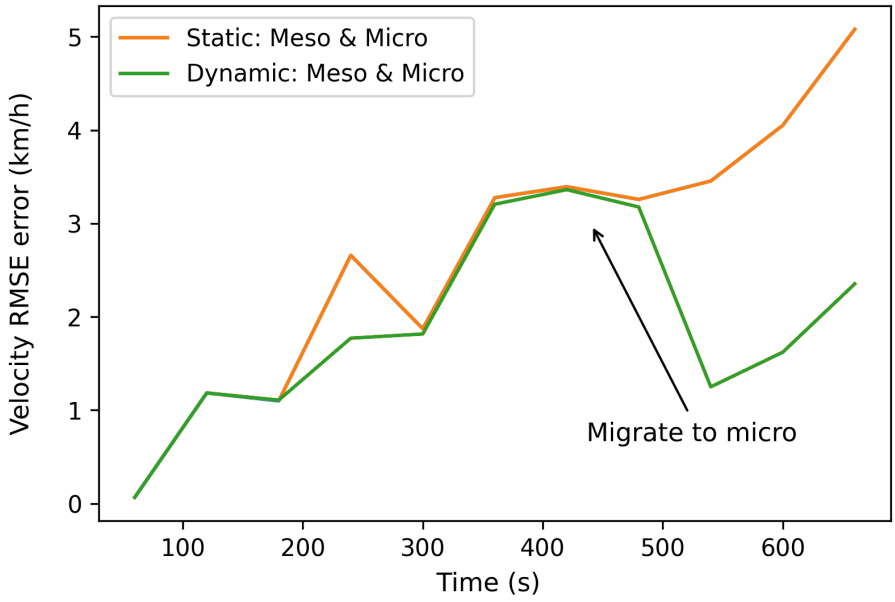

Figure 5.10: Validity Error (RMSE) over time

to dynamically switch abstraction levels at simulation run-time. We do this by monitoring the experimental frame of the meso simulation region, as discussed in section 5.4. The results depicted in Figure 5.10 and 5.11 show that when dynamic switching is enabled we can effectively detect scenarios that are not supported by the meso level simulation. In this case, after 440 seconds, the simulator detects that the number of conflicts do not comply with the experimental frame of the meso simulation. As a result, the MAPE-K loop will execute a plan that switches the main road from a meso level simulation to a micro level simulation. As a result, we can see a strong decrease in the velocity RMSE, and thus a decrease in the error. At the same time, we observe an increase in the computational cost.

5.5.2 Experiment 2: Full Experiment

In this experiment, we attempt to simulate a full urban environment using our dynamic multi-agent simulation approach. This is one of the major contributions of this work, given that we haven't seen in related literature that a dynamic abstraction methodology has been applied to a realistic, production-ready traffic simulation at this scale. The setup looks as follows: the environment is based on the inner street network of Antwerp. We identify four zones that can switch abstraction levels. The setup is visualized in Figure 5.2.

We initialize the abstraction zone at the micro abstraction level. We then initialize cars driving from fourteen origin zones to fourteen destination zones in the inner city. During the first part of the simulation, the traffic volume is low but over time we increase

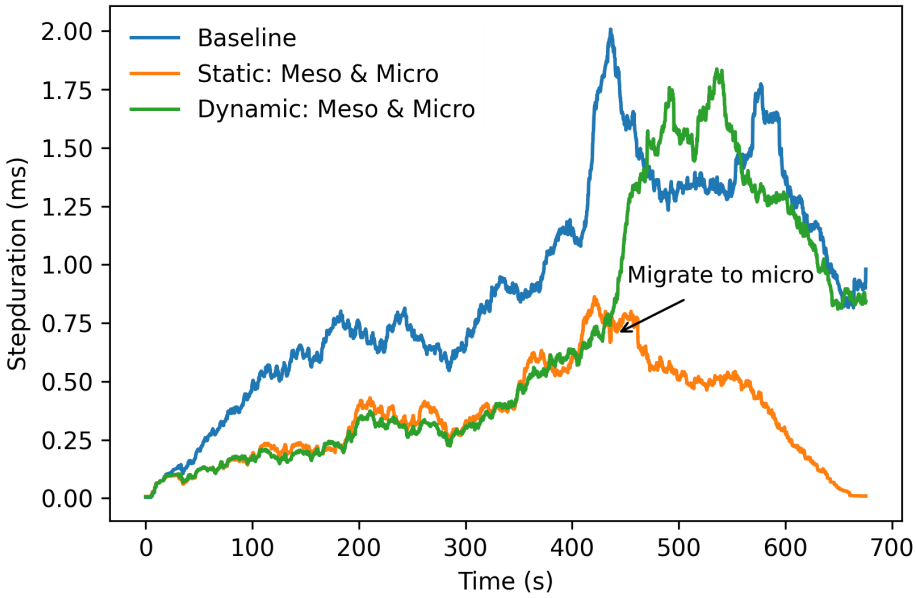

Figure 5.11: Computational cost expressed as step duration over time

the volume by increasing the rate of cars being added to the origin zones. We then run three variations of this experiment:

- Setup 1: Static Micro only: In this experiment the MAPE-K module is disabled. Furthermore, the meso level is not initalized. All cars in the simulation are simulated at the micro level.
- Setup 2 - Static Meso and Micro: In this experiment the MAPE-K module is disabled. Switching between levels is not possible, and all four zones are running at the meso level, the regions outside these zone are still simulated at the micro level.
- Setup 3 - Dynamic Meso only: In this experiment we only allow switching from micro to meso level, switching from meso to micro level is not allowed.
- Setup 4 - Dynamic Meso and Micro: In this experiment the MAPE-K module is allowed to switch either of the four zones from micro to meso level and from meso to micro level.

The results of the experiment are displayed in Figure 5.13 and Figure 5.12 below.

The results of this experiment prove that the proposed dynamic switching methodology allows effective switching between abstraction levels to properly balance between computational performance and the validity of the simulation results. The MAPE-K module is keeping track at the micro and meso level of the amount of conflicts that occur at intersections. Initially this amount is very low in zone 2, 3 and 4, as a result these

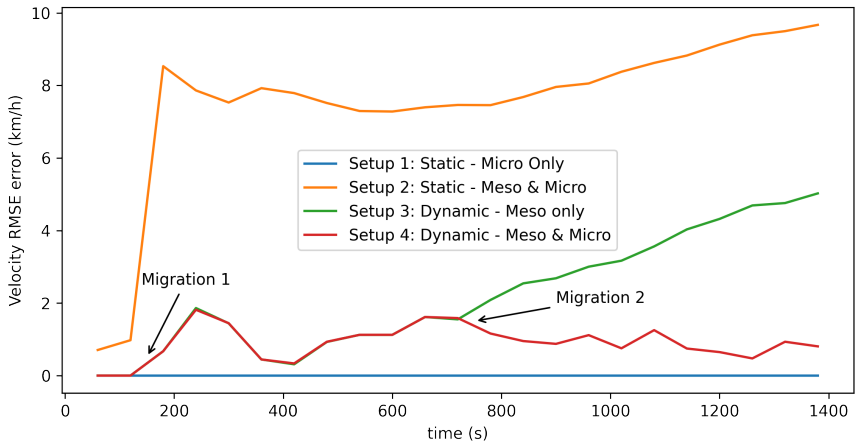

Figure 5.12: Experiment 3 - Validity Error (RMSE) over time

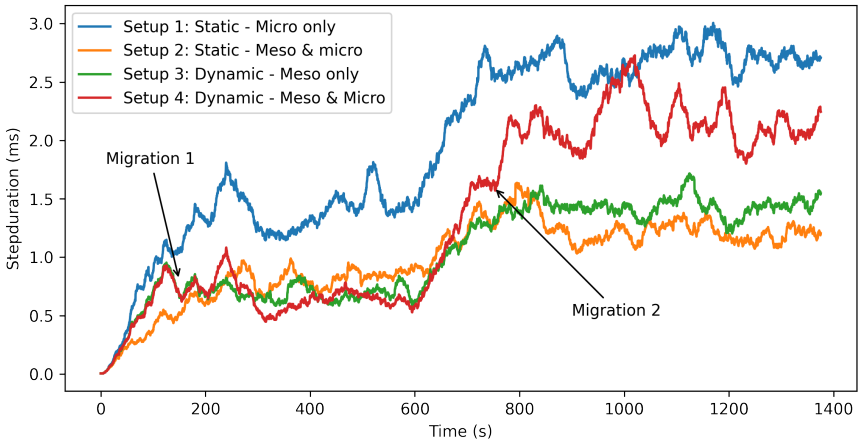

Figure 5.13: Experiment 3 - Computational cost expressed as step duration over time

Title	Time	Description	Zones	Setup 3	Setup 4
Migration 1	150	Migration from micro to meso	2, 3 and 4	yes	yes
Migration 2	700	Migration from meso to micro	2	no	yes

Table 5.2: Migrations occurring during the experiments

zones will be migrated to the meso level after 150 seconds when the MAPE-K module is enabled. Zone 1 experiences a high amount of conflicts from the start, therefore it remains at the micro-level during setup 3 and 4. During setup 3 all zones are initialized at the meso level and remain at this level during the entire simulation. The benefit of the dynamic switching strategy is shown in Figure 5.12, the error of setup 2 is very high from the start. Whereas the average error of setup 3 and 4 remain rather low because in this setup zone 1 remains at the micro-level.
After some time, the traffic density increases. This leads to a significant increase in conflicts in zone 2. As a consequence, we see a significant increase in the average error in setup 3 compared to setup 4. The reason for this, is that only setup 4 allows zone 2 to switch from meso to micro level.

Figure 5.13 we show the computational benefit of dynamic hybrid simulation. Both setup 2, 3 and 4 are significantly less computationally expensive compared to setup 1, which only supports micro level simulation. We show that after the first migration (see table 5.2) the computational cost is significantly lower compared to the static micro level simulation. After 700 seconds we can observe a significant increase in computational complexity after the second migration of setup 2. This is the cost we pay for the migration from meso to micro, which was needed to reduce the average error introduced by the increase of traffic density in zone 2, as can be observed in Figure 5.12. When interpreting the computational results, it is important to understand that the execution time of a micro agent strongly depends on the context of the agent. This is due to optimizations added to the micro agent. For example, when a car agent enters a new road it will need to do additional calculations. Whereas, when a car is driving with one car in front on the same street segment, this car doesn't have to perform the computationally very expensive calculations to determine if there are conflict situations at the upcoming intersection. And, when an agent migrates from the meso zone to the micro zone this also adds additional computational overhead. Furthermore, the meso level execution time is only 0.7 % of the micro level execution time, meaning that the the major share of stepdurations demonstrated in Figure 5.13 is caused by the micro level simulator that remains active in all four setups. This can lead to outliers in the step duration. For example, at time step 1000 the execution time of Setup 4 (dynamic - meso & micro) is higher then the setup 1 (static - micro only). This is because of a strong increase in migrations leading to additional overhead and micro-level calculations. The most important observation is that the general trend of Setup 4 is significantly better compared to Setup 1.

5.6 Conclusion

In this chapter, we presented a method and simulation architecture that is able to dynamically switch between abstraction levels of a traffic simulation use case. The results show that we succeed in balancing computational cost and model accuracy. We show that the concept of experimental frames to determine model validity based on context and the MAPE-K paradigm allow for a generic, reusable way to dynamically optimize simulation scalability. To our knowledge a dynamic multi-level traffic simulation has not been presented before at this scale in related work. We show that our method could be implemented in commercial traffic simulators and can also be leveraged in other agent-based simulation domains. In future work, we will aim to add different macro models. Furthermore, we want to combine the work of dynamic abstraction with dynamic load

balancing. This will allow us to take advantage of large-scale computation infrastructure such as Cloud infrastructure or High Performance Computing infrastructure. We also want to test our framework in an optimization scenario, its dynamic scalability properties could be interesting to be used in the context of traffic light optimization, where a large amount of simulation runs are required and efficiency is key.

Chapter 6

Data-Driven abstraction of traffic simulations

6.1 Introduction

In this chapter, we discuss the work related to the implementation of data-driven traffic models in the context of traffic simulation. As we've seen in previous chapters, creating accurate simulation traffic models is not an easy task. These models are computationally often very inefficient. As a result traffic simulation modelers leverage traffic models at different levels of abstraction. This allows them to better balance computational cost, scale of the experiment and accuracy of the simulation results.
In this chapter, we look into a different technique. Instead of leveraging explicitly designed models we aim to learn the traffic behavior using machine learning models. We will specifically discuss the implementation of a state-of-the-art Graph Neural Network that was used effectively to predict traffic evolution over time. We will adapt this model to make it compatible with our custom simulation framework and integrate it in a hybrid simulation setup.
The integration of such a model leads to a number of challenges that we will discuss in this chapter.

The feature diagram in Figure 6.1 shows that our contributions in this chapter are mainly in the Multi Formalism area, where we focus on entity aggregation by switching multiple agents in the Agent Based Formalism to a machine learning based approximation. Furthermore, we abstract not only in the model state representation space but also in the temporal space. This is achieved by increasing the execution time intervals of the abstract model. Furthermore, we work with a fixed model abstraction strategy where some areas are abstracted from the simulation start.

In the first section, we provide the motivation on why leveraging data-driven abstraction can be useful. In section 6.3, we provide background regarding data driven traffic simulation, the use of deep learning for modeling traffic and the architecture of the Graph Neural Network that we use in this chapter. In section 6.4, we present a framework on how the data-driven model can be integrated in a multi-level simulation setup. Finally,

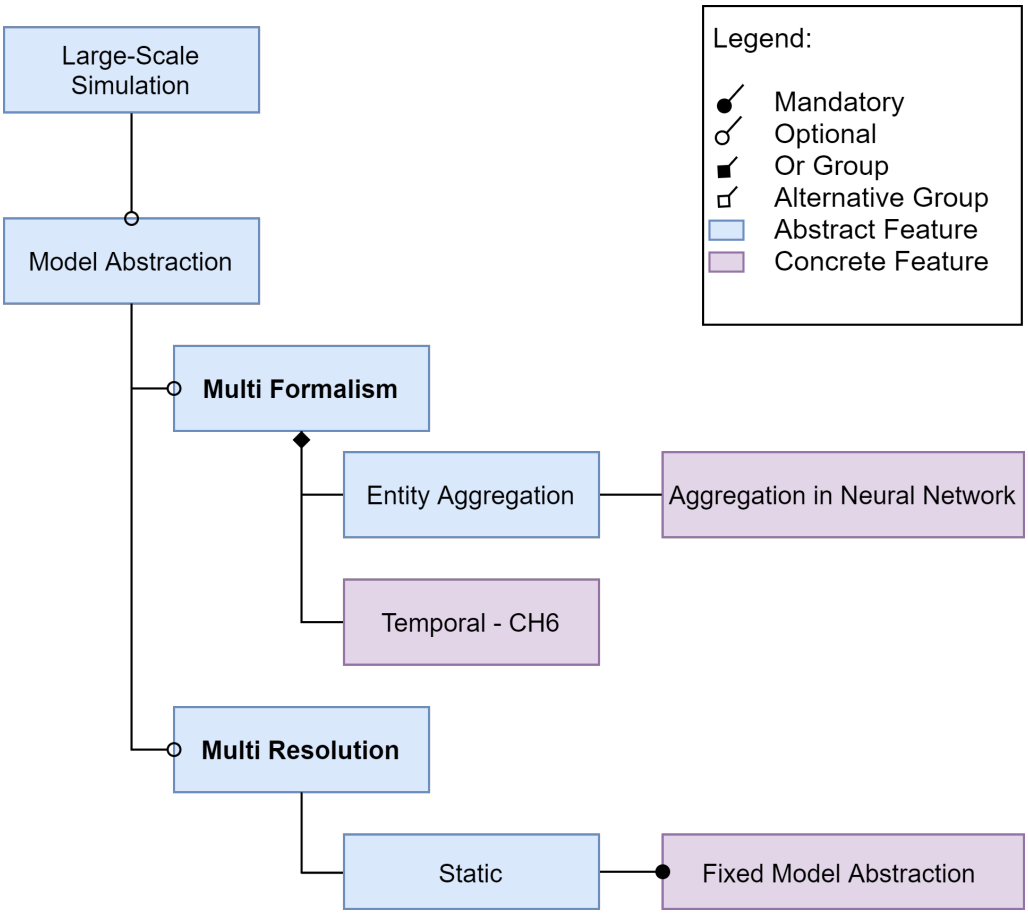

Figure 6.1: Feature Diagram - Contributions Chapter 6

we present initial results of how well the Graph Neural Network is able to predict the behavior observed by the simulator.

6.2 Motivation

In chapter 5, we saw that traffic macro-models were often explicitly modelled. These models typically need a high modelling development and calibration effort. In this chapter, we propose a data-driven method to mitigate the shortcomings of explicitly modelled traffic models. Data-driven models have capability to learn behavior based on raw data. This could make calibration much easier. Furthermore, the model can predict for a long time in future, highly reducing the required amount of model executions. This can significantly reduce the execution time of a data driven macro model opposed to a traditional macro model. This is, because the model we present allows to predict the behavior for a longer time in the future with a single execution step.

One of the other disadvantages of this model is that it assumes the availability of high

quality and large datasets of traffic in certain regions. In practice, such datasets are often not available. But with the increase of Internet of Things deployments and smart city environments we assume that over time this availability will continue to improve.

6.3 Background

In this section, we discuss the relevant work and relate it to the contributions that we make.

6.3.1 Using data-driven models for traffic simulation

The use of data-driven models to work together with traffic simulation is most described in the context of optimization where surrogate models, or simulation approximations, are used to reduce the computational cost of the optimization process. We provide an overview of most relevant related work in this section. Osorio et al. [96] [97] present in their work a manually modelled surrogate model. They leverage two models, a high fidelity model and a low fidelity surrogate model. During the optimization process they switch between the models based on the expected accuracy of the model. As a result they were able to reduce the computational cost by 78% percent.
Gil et al. [98] leverage a fuzzy rule surrogate model to approximate the behavior of a micro traffic simulation. They use this model to reduce the computational cost of traffic light optimization.
El Vlahogianni et al. [99] use an ensemble of surrogate models based on machine learning techniques to predict traffic data based on raw sensor data. They don't combine the surrogate models with a simulator but directly learn from the surrogates.
Finally, Chen et al. [100] present a surrogate model to replace a computationally expensive traffic simulator to optimize highway toll charges.
Most of the related work presented in this section leverages surrogate models to reduce the computational cost of optimization problems. They often propose a surrogate model that replaces the entire simulation. This can lead to a significant reduction in accuracy. We believe that a combination of both a surrogate model or an approximation with actual traffic models will be able to better balance between accuracy and computational efficiency. Especially when the models are used on longer time horizons, because in that case the validity error tends to increase over time.
Furthermore, there are many scenarios where high fidelity information regarding the traffic state is required. For example, in the case of traffic light optimization it makes sense to keep track of individual cars entering intersections and keeping track of their local behavior when crossing the intersection. Most of this information would be lost when approximating the entire simulator by a surrogate model. The same for simulating the impact of local events (e.g. car crashes) holds on the rest of the network. The techniques discussed in this section are mainly implementations of classic machine learning and surrogate modeling techniques, with the rise of deep learning in the last years there might be more effective techniques to model traffic behavior. We'll provide an overview of the latest research in this area in the next section.

6.3.2 Deep learning for short-term traffic prediction

Most related research in the context of modeling traffic environments and their behavior can be found in the context of traffic prediction. The goal of such model is typically to predict the upcoming traffic based on past observations. This research hasn't been published with a goal to create traffic surrogate models but since it is able to properly learn the dynamics of traffic we believe it can be used as a surrogate model as well. While in the earlier days, traffic models were purely based on explicit models like flow models, or differential equations, current state-of-the-art short-term traffic prediction techniques are based on machine learning [101]. Such systems, however, either fail to capture the actual physical characteristics of the road network and/or fail to preserve the spatial and temporal relation of the traffic data, resulting in poor traffic predictions.

Further, training such systems is a tedious and costly process as they require a vast amount of training data. Yu. et al [102], created a spatio-temporal graph convolutional neural network (STGCN) solving the scalability problem and allowing the model to better learn the spatial and temporal relationships. This leads to a significant increase in the performance of the traffic predictions. Zheng [103] used the same idea and implemented the attention mechanism into the STGCN resulting in better performance in accuracy, but a decrease in training speed. The STGCN is also used in other domains: Cases. et al. [104] developed an STGCN architecture for relational between sensor data, while He. et al. [105] implement this architecture for context trajectory predictions with respect to autonomous driving. Yet there are still some gaps within the state-of-the-art such as transferability of knowledge, context-aware predictions and street network adaptivity. In this work, we make use of a novel STGCN architecture that has been designed to cope with the above mentioned drawbacks. We do this by developping an adapted architecture that relies on graph convolution to better take regional information into account and allow it to transfer this knowledge to other regions.

Traffic networks are commonly represented using graph structures. In our work, we represent individual road segments as nodes and connections between these segments as edges. A road segment is defined as a part of a road where cars that enter the segment also leave this segment. For example, a road will be split in multiple segments if there is an adjacent street. A road graph does not change frequently over time and thus can be presented as a static graph. Each node in the graph has features describing its state such as velocity, density and flow. The state of the nodes changes over time, in our case the average velocity aggregated over one minute intervals. The aim of our STGCN neural network is to use past traffic states to predict future states. Graph convolution [106] is the core operation used in our model to achieve predictive power. Similar to a regular convolution, spatial information will be aggregated. Each node will be aggregated with its direct neighbours, and this is done in a node per node fashion resulting in an identical graph containing aggregated node states. This operation is repeated multiple times to include the influence of indirect neighbours with a residual connection to achieve more numerically stable gradients. The graph convolution is applied over all time steps, sharing weights, resulting in a series of traffic states. These states are evaluated over time to create prediction for future traffic states. Figure 6.2 gives an abstract view of the spatio-temporal graph convolution neural network illustrating the difference between historical input states and future output states. The model first applies the convolution

operation to the input states, this results in an abstract embedding state of the input. This embedding is then mapped using a Recurrent Neural Network architecture, which allows to model time-based relationships, to the output states that represent the future timesteps of the traffic network. In the remainder of this chapter, we discuss the implementation of the STGCN network in a simulation context where we attempt to predict future traffic states of parts of the simulated traffic network.

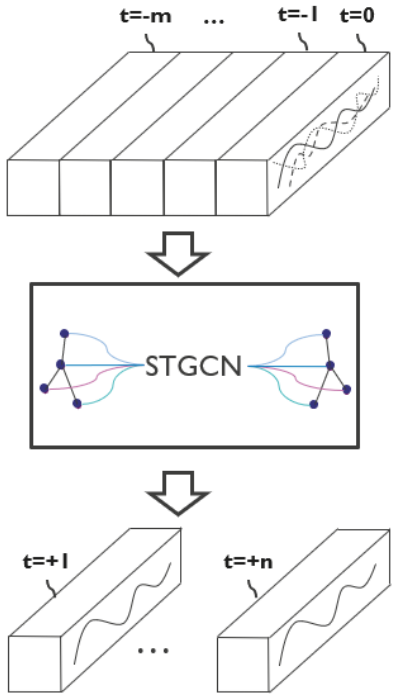

Figure 6.2: Abstract implementation of the spatio-termporal graph convolution neural network. Past traffic states are used to predict future traffic states across the road network

6.4 Implementation

In this section, we discuss how we can implement the Spatio Temporal Graph ConvolutionalNeural Network (STGCN) model in a multi-level or single-level simulation environment to reduce the computational cost and increase accuracy of the macro-level traffic models. One of the main advantages of using the STGCN is that it can accurately predict over longer time horizons compared to the meso and macro models often used in the state-of-the-art. Furthermore, it is able to learn and represent complex traffic behavior directly observed by the data.
This makes it a perfect candidate for a simulation model at a higher abstraction level. As explained in chapter 5, we assume the micro-level model to be the most accurate representation of traffic behavior, because it contains the highest level of granularity. But, it is computationally too complex, and we don't need this level of accuracy in all

regions of the simulation. Therefore, a multi-level simulation is often proposed. In such a simulation we combine multiple traffic models at various levels of abstraction.

In this thesis we want to leverage the STGCN model as a macro-level model. We use a similar setup as the one presented in chapter 5 but replace the meso-model with the proposed STGCN model. It will be implemented in a multi-level simulation setup where the simulation environment is partitioned in multiple zones. Each zone can be simulated either using a micro-model or a STGCN macro model. In the scope of this chapter, we use a static configuration of the abstraction levels of each zone.

The model will be fed with traffic observations of the last five minutes and it will predict the relevant traffic parameters over the next thirty minutes. These parameters are the traffic density, traffic velocity and flow at each street of the STGCN zone. The fact that we can predict for up to thirty minutes leads to a temporal abstraction, which we discussed in the introduction of this thesis. The meso model and micro model used in chapter 5, were executed 4 times per second. Whereas, in the model that we propose a time horizon of thirty minutes is used. This leads to an enormous reduction in computational cost.
One of the main consequences of abstracting over this time axis is that the model can't react to sudden changes of behavior caused within this 30 minute time frame. As a consequence, the predicted traffic behavior can be invalid for maximum the duration of the specified time horizon. After this time the model will be executed again, now taking the changes in the previous time interval into account. This is also one of the main challenges when we integrate models with different time resolutions in the same simulation, as is the case in our proposed setup.
We aim to solve this using the MAPE-K framework presented in chapter 5. At the edges of the model we can use this framework to detect such events. When such an event occurs we can automatically trigger an additional execution of the model, taking the unexpected behavior into account, and limiting the invalid behavior resulting from the STGCN model.

Another challenge is the integration of the micro-level model and the STGCN model. The STGCN zone is characterized by a number of streets, we can differentiate between regular streets, sinks and sources. The sources are the streets where cars enter the STGCN zone from the micro-level simulation. At the sinks the car leaves the STGCN zone and enter the micro-level model. These sources and sinks are where the major integration challenges occur. Because when a micro-level car enters a STGCN zone we can't keep track of this individual model during its propagation in the zone. Furthermore, we can keep track of the traffic density at the sinks, but we won't be able to identify which car is leaving there. We also won't know at which exact timestep the car will leave the sink because of the mismatch in time resolution between the models.
This means that we can't prevent information loss when integrating these models. But we can limit this information loss by storing the information of the cars entering the STGCN zone. Each car stores it routes, so we can use this information to find the sink it will leave the zone from. Instead of fully removing the car from the simulation, we can thus store its state. At the sinks we can store a FIFO queue of all cars that will leave from a given sink. At each sink, the STGCN model will have information about

the traffic density and average velocity of the cars. Based on this density we can sample cars from the queues during this 30-minute interval and restore the state of the car in the micro model. Furthermore, we can adapt the current velocity to the velocity observed at the edges.

6.5 Experiment

In this section, we evaluate the effectiveness of the STGCN when trained on simulation data. This gives an insight in whether the STGCN is a viable candidate for replacing explicitly modelled macro traffic models. We will evaluate the model based on the accuracy of the predicted velocity. To do this, we simulated a couple of thousand cars driving through the city center in the micro simulator and log the average velocity and density observed at the street level for two hours and a half. Based on the data of the first two hours we both trained the explicit macro level model presented in chapter 5 and the STGCN. We then evaluated the model on the last half hour. This should give a representative indication on how well the STGCN model is able to learn the traffic dynamics of our simulation. Furthermore, it allows us to directly compare the STGCN with the explicitly modelled macro model presented in chapter 5.
The results are presented in Figure 6.4. We gathered data of 960 streets and use both the macro model and the STGCN to predict the traffic velocity in the upcoming 30 minutes based on the previous observations. We evaluate the accuracy of the models by calculating the Mean Squared Error (MSE) between the actual observed values in the micro simulator with the predicted values. We did this for the entire 960 streets and for a zone of a limited number of streets shown in Figure 6.3, which is referred to as zone 1. The reason for this separate evaluation on zone 1 is that zone 1 contains a high number of traffic lights on the main street. This leads to deviating velocity patterns in the observed traffic behavior compared to the average behavior observed for the entire street network of 960 streets.

Figure 6.3: Experiment Evaluation Zone showing traffic lights and the streets of interest

Figure 6.4: MSE Error on traffic simulation test set

The results in Figure 6.4 show that the error of the STGCN model on the entire traffic network is low. The average observed error, an MSE of 3.173 on the test data. Which is squared, so when we take the square root it can be interpreted as an average error of 1.78 km/h in predicted velocity which is a very promising result, close to the local performance of the macro model. It shows that the model is able to learn the overall traffic behavior. The error of the STGCN model is higher because of the impact of the traffic lights on the observed behavior. The error of the macro model starts low but increases over time. This resulted in an average observed error at street-levels that is near zero at the beginning of the test-set, but the error increase to an average plateau near an MSE value of 5. The reason for this is that the STGCN predicts the network state for the following 30 minutes within a single execution, whereas the macro model gradually alters its state during the many executions of the model in this 30 minute time period. The STGCN error is higher from the start, but remains stable over time. The STGCN currently doesn't take input data at the sources of its zone into account, which refrains the model from adapting to newly incoming observations. To solve this, a different approach would be to predict only for a single timestep and use the output of the predicted timestep, combined with new observations again as input to the STGCN model. This is an autoregressive approach, but it has as disadvantage that the error at a later timestep might increase due to the vanishing gradient problem.
When we compare the macro and STGCN results after 10 minutes, we see that the Macro model is less stable. This is also because the model is not adapted to taking traffic light behavior into account as was discussed in chapter 5, leading to sudden increases in errors depending on the traffic light state. The STGCN is more stable because it is better able

to anticipate to these spikes in velocity changes. In future work we will continue to adapt the STGCN to better take the impact of the traffic lights into account in order to reduce the gap in MSE between the predictions in zone 1 and the predictions of the entire traffic network.

6.6 Conclusion

In this chapter we presented the work we are currently doing in the context of integrating a data driven model in a state-of-the-art traffic simulation framework. Our main focus was on training the STGCN on data observed by the traffic simulator to show that the STGCN could operate as macro-model in a traffic simulator. We showed that the STGCN was able to effectively do this, but needed to be better tuned to cope with the impact of traffic lights on the observed traffic behavior. Furthermore, we proposed an implementation architecture of how we can integrate the STGCN in a multi-level traffic simulation setup. Because of the abstraction in time resolution that can be obtained by the limited amount of predictions the model needs to execute we expect significant reductions in the execution time of the simulation.

Chapter 7

Conclusions

In this dissertation we looked at how we could test large-scale Smart city or Internet of Things environments using simulation. We saw that there is limited related work available which allows for simulating large-scale smart environments. We did an in-depth analysis of the possible solutions to improve simulation scalability. And we then focused on developing a methodology for large-scale agent-based simulation in the context of smart traffic environments. We put special emphasis on the fact that the proposed methodology is able to dynamically optimize the computational cost based on the context of the simulation.

The methodology presented in this work is based on two major pillars inspired by our initial analysis. In the first we looked at improving partitioning capabilities, we presented a generic, decentralized method to dynamically reduce overall computational cost, taking both communication and model computation cost into account (chapter 3). We demonstrated its efficiency using a custom developed distributed traffic simulating.

In the second pillar we looked at switching model formalisms and abstraction levels to balance between model accuracy and computational cost. We presented a toy method where we used information theory metrics as a generic approach to detect easy to approximate areas (chapter 4). When such an area was detected we switched model formalisms from a detailed agent-based model to less complex discrete event based approximation. We expanded on this work by implementing a traffic simulator that switches between a state-of-the-art agent-based traffic micro-model and a meso-model (chapter 5). We leveraged the MAPE-K framework into the simulator architecture to allow generic implementation of context-based dynamic abstraction. Furthermore, we showed to we are able to simulate large-scale realistic scenarios and both maintain simulation accuracy while reducing the computation cost. Finally, in chapter 6 we discussed the implementation of a state-of-the-art machine learning model in a traffic traffic simulator. We showed that the model is able to learn relevant traffic behavior patterns.

Chapter 8

Future Work

In future work we would like to continue to validate our method on various scenarios that could take advantage of large-scale simulation. We developed the work presented in this thesis with a smart city environment in mind. We would like to look at integrations of our method and traffic simulation framework as part of a traffic management decision support application. To do this, we will need to combine our parallel and distributed traffic simulator with our multi-model traffic simulator. During this thesis we always took the assumption that our micro-models, which are state-of-the-art, could be used as a baseline for real-world behavior, which was sufficient to prove the effectiveness of our methods. But to make the simulation behavior representative for the real world, we will need to calibrate these models on actual real-world data. Especially in the case of the micro models, the complexity of this work cannot be underestimated.

In a related scenario, we aim to leverage our framework in the context of traffic light optimization. Modern optimization techniques, such as deep reinforcement learning, heavily rely on proper simulation models. This simulation often forms a bottleneck when executed at scale. We believe the methods presented in this work can partly mitigate this bottleneck. We envision a method where we could dynamically switch abstraction levels during the learning process. In theory, the control of abstraction levels could become part of the learning process.

We have shown that Graph Convolutional Neural networks are able to effectively represent, and thus simulate traffic dynamics. We will continue to fine-tune the model and work on the implementation of the model within the micro traffic simulation architecture. Furthermore, to reduce the error in the early timesteps we will look at implementing an autoregressive approach to the STGCN model and also take input data at the STGCN zone sources into account. This will lead to additional challenges related to the vanishing gradient problem that needs to be addressed.

We have tested our multi-abstraction method with a fixed amount of regions that could switch between abstraction levels. It would be interesting to dynamically highlight regions, which were not predefined, that are allowed to switch abstraction levels. Generic metrics, such as entropy, as presented in chapter 5 can be used to do this.

One of our main contributions with regards to dynamic partitioning was the generic MAPE-K method. We would like to implement load-balancing methods described in the state-of-the-art as part of our method. This would allow us to compare and validate multiple load balancing algorithms. The MAPE-K method allows these same algorithms to be used in very different simulation applications. Furthermore, We want to explore the benefits of a hybrid decentralized and centralized adaptive load balancing approach in micro traffic simulation. In this hybrid scenario we envision two MAPE-K loops: 1) a decentralized heuristic, as proposed in this chapter, and 2) a centralized load balancing algorithm that is able to find global optimum.

Appendix A

Publications

A.1 First Author

1. Conference Papers (P1)
    a) Adaptivity in Distributed Agent-Based Simulation: A Generic Load-Balancing Approach - Stig Bosmans, Toon Bogaerts, Wim Casteels, Siegfried Mercelis, Joachim Denil, Peter Hellinckx - International Workshop on Multi-Agent Systems and Agent-Based Simulation - Springer - 2020
    b) Reducing computational cost of large-scale simulations using opportunistic model approximation - Stig Bosmans, Siefried Mercelis, Peter Hellinckx, Joachim Denil - Spring Simulation Conference (SpringSim) - 2019
    c) Towards evaluating emergent behavior of the internet of things using large scale simulation techniques (wip) - Stig Bosmans, Siegfried Mercelis, Peter Hellinckx, Joachim Denil - Proceedings of the 4th ACM International Conference of Computing for Engineering and Sciences - 2018
    d) Challenges of modeling and simulating Internet of Things systems - Stig Bosmans, Siegfried Mercelis, Joachim Denil, Peter Hellinckx - International Conference on P2P, Parallel, Grid, Cloud and Internet Computing - 2018
    e) Acsim: Towards Hyper-scalable Internet of Things Simulation - Stig Bosmans, Siegfried Mercelis, Marc Ceulemans, Joachim Denil, Peter Hellinckx - International Conference on P2P, Parallel, Grid, Cloud and Internet Computing - 2017
    f) Predictive Algorithms for Scheduling Parameter Sweep Calculations in a Cloud Environment - Stig Bosmans, Glenn Maricaux, Filip Van der Schueren - International Conference on P2P, Parallel, Grid, Cloud and Internet Computing - Springer - 2016
2. Journal Papers (A1)
    a) Adaptivity in multi-level traffic simulation - Stig Bosmans, Toon Bogaerts, Wim Casteels, Siegfried Mercelis, Joachim Denil, Peter Hellinckx - Simulation Modelling Practice and Theory - Elsevier - 2021 (Submitted but not yet accepted)

b) Testing IoT systems using a hybrid simulation based testing approach - S Bosmans, S Mercelis, J Denil, P Hellinckx - Computing - Springer - 2019

c) Cost-aware hybrid cloud scheduling of parameter sweep calculations using predictive algorithms - Stig Bosmans, Glenn Maricaux, Filip Van Der Schueren, Peter Hellinckx - International Journal of Grid and Utility Computing - 2019

Appendix B

Acsim Architecture

B.1 Introduction

This appendix summarizes the early state of the Acsim architecture. It was published as a workshop paper to the 2017 edition of 3PGCIC. This was an initial version, and throughout the dissertation we discussed various iterations of the Acsim simulator that expanded on the work presented in this appendix.

The Internet of Things (IoT) paradigm has gained a lot of attention in the last years. Both in an academic context as in the industry. This has lead to many innovative solutions improving the lives of citizens and workers for the better. Examples of such solutions can be in found in areas such as smart health-care where sensors can be used to measure certain health parameters of a user or in areas such as smart grids where the power consumption of consumers are monitored in order to better match the supply of energy. However, we are still at the start of the revolution that IoT might bring. At the moment, many solutions rely on centralized processing of sensor data in order to perform some actions in a reactive manner. This form of centralized decision making can lead to performance bottle necks when applied to ultra large scale IoT environments such as smart cities. Instead, a decentralized decision making strategy will be much more powerful as it could lead to a dynamic, adaptive emergent behavior. This type of behavior is characterized by the fact that it emerges from interaction of individual (IoT) entities that interact with each other and with a changing environment, leading to a preferably optimized global behavior. Actually, this type of IoT systems can be seen as Complex Adaptive Systems (CAS) , which is defined as a system characterized by apparently complex, adaptive behaviors that results of often nonlinear spatio-temporal interactions among a large number of component systems at different levels of organization [107]. This field of study has mostly been applied to studying the emergent behavior of biological or economic systems such as the immune system or the stock market, but can also be applied to studying the emergent behavior of a decentralized large-scale IoT system **birdsey2017large** [108].
In order to study the emergent behavior of such complex adaptive IoT systems, simulation is key. The required cost and effort to deploy the vast amounts of IoT entities in the real world would otherwise be too high. Simulation techniques can be used to validate

and verify if the demonstrated emergent behavior is preferable [4]. In this paper we will often refer to a Python IoT simulation framework that can be used to test both virtual and real-life, large-scale, complex adaptive IoT systems and allows for integration [109] with a real-life IoT environment. We pay special attention to study the domain-specific characteristics of IoT systems and try to leverage those as much as possible in the simulation framework in order to reduce the required modeling efforts. In section two, we look at the characteristics of modeling IoT entities and the IoT environment. Section three zooms in on various modeling strategies that can be applied and Section four discusses the high level architecture of the simulation framework.

B.2 Characteristics of modeling the Internet of Things

The contribution of the simulation framework that we present in this paper is the fact that we add domain knowledge into the framework. This has two major advantages: 1) Reducing the modeling effort: This is possible because we can include domain specific features in the framework. 2) Improve the opportunity to scale: when, in a later phase, the simulation architecture moves from a monolithic architecture to a parallel and distributed (PADS [110]) architecture, we can leverage IoT domain specific assumptions in our favor to include prebuilt simulation partitioning and scaling strategies. In this section we look into more detail what the high-level characteristics of IoT systems are.
As mentioned in the introduction, from a behavioral perspective, we consider a large-scale IoT system as a complex adaptive system. This is because on an abstract basis an IoT system consists of many individual, heterogeneous, autonomous components that have the ability to interact with each-other and with the environment. Furthermore, the behavior of these components is preferably adaptive, given that the environment in which they operate is chaotic and quickly changing. Examples of such systems are smart cities, smart grids, smart buildings etc. Therefore, more reactive, low-scale IoT systems such as body sensor networks are not taken into consideration in the scope of this work. From a modeling perspective, we can look at IoT systems, and also CAS systems in general, as multi-agent systems (MAS) [111]. MAS systems are defined from the bottom-up, whereby the individual entities and the environment define the dynamics of the entire systems. It is therefore vital to better understand the characteristics of the different types of IoT entities and environments.

B.2.1 IoT Device Characteristics

The key entities of IoT systems are of-course the devices itself, for example, sensors such as GPS sensors, temperature sensors or air quality sensors. But also actuators such as smart traffic lights or autonomous vehicles. Most of these entities interact with their direct environment. The spatio-temporal properties are an important characteristic of these devices. Furthermore, these devices often employ a level of intelligence, apart from standard reactive behavior they can also demonstrate some advanced planning behavior, for example, a smart traffic light could adapt its light toggling behavior when an emergency vehicle is nearby. This requires coordination and integration of the traffic light with a number of sensors and middleware systems.
In many cases IoT devices are limited in terms of the power they can consume. As a result, their processing power and their connectivity properties are limited. Furthermore, IoT

devices are characterized by the heterogeneity of underlying operating systems and their hardware properties. Most of the IoT simulators described in literature focus on testing and modeling these low-level resource-related properties, both on a small scale [112] [113] and on a larger scale [114]. However, in the scope of this work we are less concerned about these aspects, because we don't consider them vital for testing the emergent behavior of the entire system. The spatio-temporal properties and the intelligence employed by the device are of more concern to the emergent behavior, therefore the focus of our simulator will be on modeling and simulating these characteristics. But of-course, the modeler is free to integrate low-level aspects into the simulation models, this will however require additional effort.

B.2.2 IoT Actor Characteristics

Apart from the sensors and actuators, another important, however often ignored, component of an IoT simulator is the human actor. As argued by Nunes et al, humans are an essential part of cyber physical systems (CPS) or IoT systems, but should no longer be considered an external or unpredictable factor [43] . Instead, they should become a key part of the overall system. Especially, when looked at the system for an emergent behavior perspective, we will see that the human actor, and the interaction of the human with the devices leads to emergent behavior. For example, it would be extremely difficult to optimize a distributed traffic light optimization system without taking actual traffic, which is generated by human behavior, into account. Therefore, when modeling a realistic simulation of a large-scale Internet of Things systems, we cannot ignore individual behavior of human actors.

B.2.3 IoT Environment Characteristics

Finally, the dynamics of the environment must be taken into account. The environment will mainly define how IoT entities are related to each other and how they can interact with each other. We take following IoT environments into consideration:
Network environment: A network environment defines the relations between individual entities based on an undirected graph datastructure. For example, a mesh network of sensors can be represented by a network environment. The interactions of individual sensors are based on how the devices are connected to each other in the graph. This type of environment can also be used to represent smart grid systems where different households are connected to various energy providers.
Street network environment: Within a street network the interactions of entities are based on both the direction and the street where an IoT entitity is positioned. A street network is similar to a network environment, in the sense that it is represented by a graph datastructure, however, instead of an undirected graph a directed graph is used. This type of environment is mostly used for smart city or smart traffic use cases.
Continuous space environment: Finally, an IoT environment can also be represented by 2D or 3D continuous space. Here the interaction of IoT entities relies on the spatial relationships between the entities in 2D/3D space. This type of environment can for example be used to represent smart buildings or smart offices.
Combined environment: The environment types described above can also be combined. In many cases this is even preferable. For example in the case of a smart city system, a smart traffic light should be able to interact with other smart traffic lights to

improve overall traffic flow, in this case a network environment seems most appropriate to model the relations of the traffic lights. On the other hand, mobile cars should also interact with the traffic lights, in this case a street network is more appropriate.

B.3 IoT modeling strategies

In this section we look at how the characteristics of IoT devices, which we defined in the previous chapter, can be leveraged in our framework in order to reduce the modeling effort without sacrificing too much computational efficiency. In some cases the ease of modeling and maintaining performance leads to a trade-off. For example, our modeling framework is built in python. This allows for an easy development and modeling environment which leads to faster prototyping. However, given that Python is an untyped interpreted language, a lot of efficiency is lost and as a result performance is drastically lower compared to other high-level programming languages such as Java and C++. We solve this in our framework by leveraging python's capability to closely interact with pre-compiled C libraries by means of the Cython compiler. Parts of the framework that are critical for performance are compiled to C or are implemented by C-based libraries wrapped in Cython. The goal of the framework is to match these two factors as good as possible, this is done by providing the modeler with domain specific functionality which is optimized and easy to implement using a high-level API. The domain specific functionality that we offer is based on the IoT modeling characteristics presented in the previous section. In the remainder of this section we discuss some of the domain-specific modeling techniques that the framework offers.

B.3.1 Agent based modeling

As pointed out by K. Batool et al. [115] there is currently no standard methodology available for modeling complex real-world IoT scenario's. However, when looking at the literature, many practical IoT applications are modeled using a discrete event simulation (DES) approach [116] [37], an agent based simulation (ABS) approach [112] [117] [111], or a combination of both [44] [114]. As mentioned in the previous sections B.2.1 and B.2.2, IoT devices and actors are characterized by their heterogeneity, their individual and adaptive behavior, and consequently their unique interactions with the environment. Based on these characteristics the Agent Based Modeling (ABM) paradigm allows for a very expressive way to model both IoT device and IoT actor behavior. With ABM a bottom-up modeling approach is taken, whereby individual entities are implemented as an individual agent. Each agent has individual properties and has the ability to communicate with other agents or with the environment [118]. As noted by G. Fortino et al. agents represent a very expressive paradigm for modeling dynamic distributed systems. Their primary features (autonomy, social ability, responsiveness, pro-activeness, and mobility) perfectly fit both generic and specific requirements of IoT systems [44]. G. Fortino also notes that the ABM paradigm isn't suitable for dealing with certain low-level network aspects, in their work they propose a combination of ABS and DES, where a DES simulator is responsible, and better suited, to simulate these low-level aspects. Since the scope of the simulation framework that we present in this work is limited to simulating overall behavior, without directly taking low-level aspects into account our framework is limited, and build around the agent based simulation approach.

B.3.2 Domain specific environment capabilities

Based on the IoT environment characteristics described in section B.2.3 we can conclude that a proper representation of a physical environment is important when developing IoT applications. Especially in the context of smart city applications, knowledge of street networks must be taken into account. We offer this functionality in our framework by means of a Geographic Information System (GIS) engine. More specifically, an Open Street Map (OSM) [90] parser was included in the GIS engine of the framework. Open street map is an open source project that collects street and other geographical data of the world. The OSM parser in our framework extracts streets data and loads it into a directed graph so that it can be used and queried efficiently by modelers from an environment object. It offers functionality to calculate routes between locations and it allows to easily determine which entities are located on a street or crossing. Optimization is of course key in the domain specific functionality that the framework offers. Therefore all environments are driven by optimized data structures. Especially, locality information is an important feature in the context IoT modeling, for example a modeler often wants to know what the nearest neighbors of a given entity are, or in other words which entities are in closest proximity of another entity. Proximity in this context is an abstract notion, as the way to find this locality information depends on the type of environment that is used. In a street network or graph network, proximity between entities depends on the distance between graph vertices. This information can quite easily be determined by using a graph data-structure and traversing the graph using breadth- or depth-first-search. While in a 2D or 3D environment locality depends on the distance in continuous space determined by Euclidean distance for example. In the latter case we could naively calculate the distance between all agents, this would however lead to a very inefficient and unscalable solution. Instead, we optimize this by implementing an R*-tree data-structure [119] that allows nearest neighbor or range queries to performed in logarithmic time versus linear time.

B.3.3 Modeling IoT agent behavior

Based on the characteristics of IoT devices we look at three different approaches to model IoT device and actor behavior. All of the approaches have been implemented and tested in the framework that we present. Overall, we assume a discrete time-stepped simulation, whereby behavior is updated at each time-step interval. Therefore each agent implements a step method, which is called by the simulation kernel at a fixed interval. During the execution of this step method agents are able to interact with each other, with the environment and consequently update their internal state.

Reactive Modeling: The most basic behavior to model is that of simple 'if-then' reactive rules [120]. This can either be implemented directly in code as part of aforementioned step method or via an additional domain specific language on top such as state charts. Reactive modeling is thus mostly appropriate for modeling simple, reactive behavior. This type of behavior often occurs in IoT devices, take for example a smart traffic light that toggles lights based on perceived traffic or a predefined schedule. A problem however is that modeling more complex behavior often gets complex and unmanageable. Take for example the modeling of a (smart) vehicle, many different behaviors and states need to be tracked: driving behavior, collision avoidance, adhering to traffic regulations such as speed limits, stopping for a red light etc. In such case it is often preferable to

split up behavior in logic classes to maintain both readability and maintainability. This is called a layered modeling approach. For example the behavior to drive at a certain speed on a given route can easily be isolated. Actually, since modeling driving behavior often occurs in smart city applications the framework offers predefined driving behavior classes that can easily be reused by other applications. This layered approach offers a clean decomposition of overall functionality or behavior, however, it is not always clear how to decompose such behavior of a system, and also it requires interactions between layers [121].

Belief Desire Intent (BDI) Modeling: A major shortcoming of reactive modeling is that more advanced and high-level human-like planning behavior is hard to implement in such formalism. This will however be required when implementing increasingly complex reasoning in either advanced IoT devices or human IoT actors. Since the goal of the framework is to test and evaluate IoT systems as a whole, this type of complex, not always deterministic behavior will need to be taken into account. This is definitely the case when our goal is to evaluate complex adaptive behavior. A technique that allows modeling of such planning behavior, is the belief-desire-intent (BDI) architecture. This model is based on practical reasoning that we do in everyday life. The modeler needs to declare the beliefs, desires and intents of system. A belief, represents the information an agent holds about the environment, these beliefs exist by perception of the environment or by interaction with other agents. The desires represents the goals of an agent. Finally, intents are the actions that can be taken based on the current desires and beliefs. In other words, the intents represent a plan of action that an agent can take in certain scenario's. Note, that often part of the plan has to be implemented in a reactive way, often by more low-level programming languages. Various BDI engines have been implemented in the past, many of them based on AgentSpeak, a programming language that combines the ideas of logic programming and the BDI architecture in order to model abstract reasoning behavior in agents [122].

Lom et al. demonstrate in their work by means of example how BDI and AgentSpeak can be used to model behavior of smart city entities. For example, they model the behavior of a smart street lantern. The beliefs of the lantern are its current states, its energy consumption, its schedule and its maintenance status. It desires are to measure or predict its consumption, send its status to maintenance companies, fulfill actions based on its schedule and send its energy consumption predictions to a smart power grid system. Based on the current belief state of the smart lantern, many plans of actions can be taken to accomplish its desires. These plans of actions are not necessarily deterministic, and its sequence of actions can change based on changes in its beliefs.

BDI is a technique that has been researched in-depth for years in the context of modeling multi-agent systems. It should however be also very useful for describing complex reasoning behavior in Internet of Things agents. Although, the idea of AgentSpeak was to allow for an easy modeling approach that is also accessible for people without a computer science background. We see that in practice, the complexity of the architecture and the programming language is high and as a consequence an in-depth knowledge is required to get started with it. Also, BDI lacks the capability to adapt its behavior over time. This prevents us from fully adopting this technique. However, in literature initiatives are described to make the BDI idea less complex to implement.

Data driven Modeling: Finally, another approach to modeling behavior of IoT entities is using a combination of data mining and machine learning techniques. This approach is especially useful when the behavior of IoT entities needs to resemble that of already

observed data which is stored in a data stream. In consequence it leads to implicitly validated behavior when the trained model's accuracy is appropriately high. Kavak et al.[123] demonstrate in their work how a data-driven modeling approach can be used to create realistic mobility patterns. They use geo-tagged twitter data to predict certain movement patterns and use this to drive high-level decision making of individual agents. Just like BDI, also this data driven approach is ideal for modeling high level behavior and planning but still requires low-level modeling in order to implement reactive behavior. For example, in the example of Kavak et al. it the data driven model can be used to decide when and where an agent will navigate to, but the actual driving behavior still needs to be implemented by another more appropriate formalism.

In this section we presented various modeling approaches that can be used when modeling behavior of IoT entities. This list is driven by the characteristics of Internet of Things devices. When each formalism should be used will depend on the type of behavior that needs to be modeled. The BDI and data driven modeling approach are most appropriate when modeling complex behavior and decision making. Whereas reactive modeling is used to represent more simple behavior patterns.

B.4 High level framework architecture

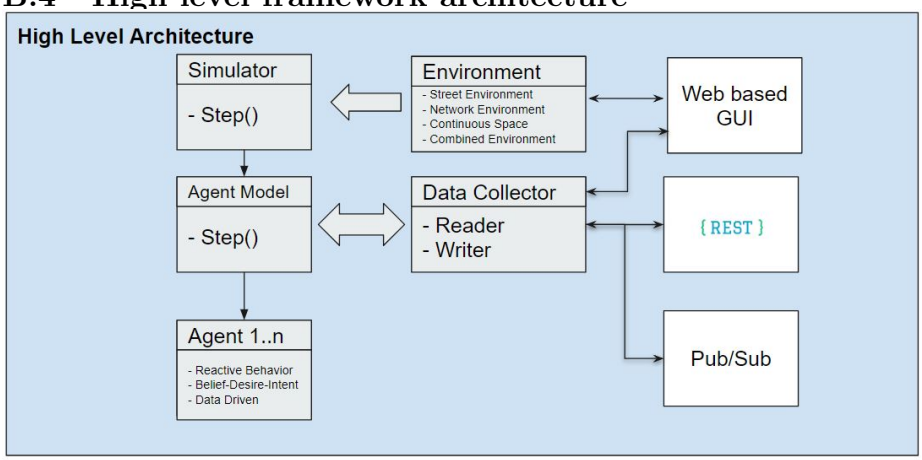

Figure B.1: High level architecture of agent based simulation framework

Figure B.1 shows the architecture of the IoT simulation framework. Most of it is based on the Mesa simulator [59]. Mesa is a generic agent based simulation framework. It aims to enable users to quickly build agent based simulations and can be seen as the Netlogo and Repast alternative for the Python language. We leveraged many of the architectural ideas and extended them with IoT domain-specific models and a more in-depth functionality to integrate a running, real-time simulation with an existing IoT environment. As shown in figure B.1 the simulator component is the main component and is used to configure overall simulation information like the models and the environments that are be used. It implements a step method, which, when triggered, cascades down to trigger the model and agent step methods. The model component is responsible to initializing and managing the individual agents of a certain type, for example the car model will initialize all car agents. Furthermore, it interacts closely with the data collector component to collect and modify state data of one or more agents.The Agent implements the behavior of the individual agent and will update its state when the step

method is triggered. Various behavior modeling strategies can be taken, as explained in section B.3.3. The environment component allows interaction with agents and between agents. Various environments are already implemented based on the overview in section B.2.3. Finally, the Web based GUI component, the Rest interface and Pub/sub interface communicate with the data collector to visualize and collect state information of the agents.

B.5 Conclusion

In this work we presented the characteristics of Internet of Things devices, actors and environments. We looked at how we can optimize simulation related implementations of these entities using an agent based simulation approach. Optimization is possible by including domain knowledge in the simulation engine, by means of optimized data-structures and techniques. Finally, the advantages and disadvantages of a reactive, data-driven and belief-desire-intent modeling approach were discussed. We looked at how each of these modeling approaches can be applied to Internet of Things modeling. In future work, we will pay special attention to the simulation architecture that is used and how we can move from a monolithic architecture to a distributed architecture so that large-scale simulations can be included.

Appendix C

Cost-aware hybrid cloud scheduling of parameter sweep calculations using predictive algorithms

Abstract

This paper investigates various techniques for scheduling parameter sweep calculations cost efficiently in a hybrid cloud environment. The combination of both a private and public cloud environment integrates the advantages of being cost effective and having virtually unlimited scaling capabilities at the same time. To make an accurate estimate for the required resources, multiple prediction techniques are discussed. The estimation can be used to create an efficient scheduler which respects both deadline and cost. These findings have been implemented and tested in a Java based cloud framework which operates on Amazon EC2 and OpenNebula. Also, we present a theoretical model to further optimize the cost by leveraging the Amazon Spot Market.

C.1 Introduction

This appendix presents work not directly related to this dissertation that was part of my master thesis. The core of the work has been executed in 2013 - 2014, an expansion of this work was later submitted for publication in 2017, during the first year of my PhD.

The increasing demand for processing power over the past years forced technology to evolve from supercomputers to other solutions. Grid systems have been a viable solution for a long time, however these resources aren't unlimited. For additional computing power, techniques were developed to address idle resources on desktop computers. Idle desktop computers were used to create a heterogenous grid. [124] [125] [126] More recently, cloud computing is changing information technology. Various companies offer public cloud services, providing virtually unlimited resources on demand, but with a cost. These resources can be complemented with self owned resources such as a private cloud. To keep the cost at a minimum, a scheduler will distribute a minimum of workload to the public cloud, but the workload is still processed before a deadline. Related work

shows how a cost optimization scheduler can reduce the cost to process a workload [127] [128]. However, this research uses heuristics to estimate the workload. In this paper, various (non-heuristic) techniques will be discussed in depth to create a cost efficient scaler which will allocate resources in the public and private cloud.

In our research the workload is defined as a parameter sweep which is an application that executes the same piece of code multiple times with a set of input parameters. For example the surface of an airplane wing or the weight of an airplane can be parameters to determine the fuel consumption of the plane. Each combination of different input parameters is referred to as a job. Our research focuses on how we can cost efficiently schedule a parameter sweep application in a hybrid cloud environment while respecting the deadline. In order to do that prediction techniques are used to estimate the total execution time to construct a reliable model as fast as possible. [129] Different sample selectors schedule the workload of the parameter sweep by increasing priorities, the higher the priority the sooner the job has to be executed. A scaler is used to divide the workload on both the private and the public cloud. Also, the scaler is responsible to start extra instances when the predicted runtime exceeds the deadline. To respect the cost it might be possible that the scaler will stop public instances in order to save money.

We present a novel theoretical approach to further optimize the cost by combining amazon spot market instances with regular reserved cloud instances using predictive scheduling techniques.

This paper is organized as follows: Different sample selectors are discussed in section two. Section three focuses on the implementation of various prediction techniques. Scaling and scheduling are discussed in section four. Section five describes the developed framework. The results of our research are evaluated in section six. In the seventh section we present a theoretical approach for further optimizing the cost of the system. Finally our work is concluded and the future research is outlined in section seven and eight.

C.2 Sample selector

The sample selector assigns priorities to each individual job of a parameter sweep. The higher the priority of the job, the sooner that the job will be executed. Once a job or parameter combination is executed the runtime of that job is logged into a database. These runtime measurements are then used as scattered points in the prediction model (section 3). The order of the execution of the parameter combinations is very important in regard to get a reliable prediction as soon as possible. In order to have a reliable prediction, testing reveals that the whole range of parameters should be covered. The random sample selector (Figure 1b) is clearly the most promising because samples in a wide parameter range are selected at an early stage.

We implemented three different sample selectors in our framework:

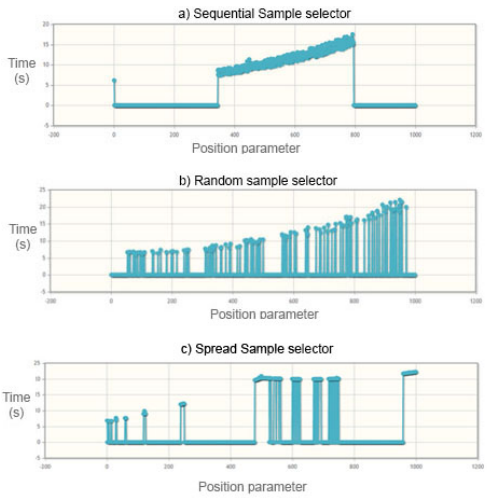

Figure C.1: Sample selector example

C.2.1 Sequential

When using a sequential sample selector, parameter values will be fetched from the database one after another. This will make it very difficult to estimate the runtime at an early stage because only one part of the parameter range is taken into account (Figure 1a).

C.2.2 Random

A random sample selector will select random parameters within the parameter range. The random sample selector is the most effective because each part of the parameter range is taken into account. Experimental results show that a wide area of the parameter sweep is covered at an early stage.

C.2.3 Spread Algorithm

The goal of this algorithm is to maximize the spread of the selected samples. The algorithm will select unprocessed samples so that the distance between processed samples is the greatest possible. The algorithm works as follows:

1. Increase priority of the highest value of the unexecuted parameters;
2. Increase priority of the unexecuted parameter closest to the last increased priority parameter divided by two;
3. Increase priority of the unexecuted parameter closest to the highest unexecuted parameter divided by two + the highest unexecuted parameter divided by two;
4. Repeat step two until parameter zero is reached;
5. Repeat the algorithm from the beginning.

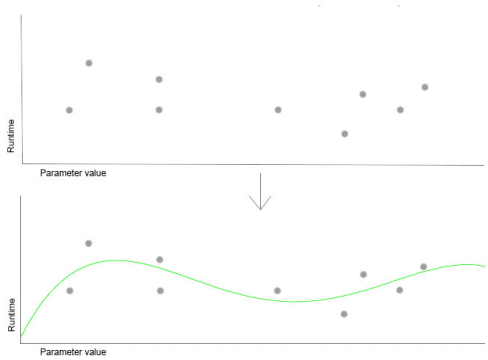

Figure C.2: Scattered points before and after interpolation (y-axis: duration (s), x-axis: job position)

C.3 Prediction techniques

To execute a parameter sweep application as cost efficient as possible and to respect a deadline, it is necessary to estimate how much resources will be required to execute a complete parameter sweep. Regression algorithms are used to create an estimate of the runtime and of the corresponding resources required to construct a reliable model as fast as possible. A significant dataset needs to be fetched to the algorithms in order to obtain a reliable estimation. Depending on this estimation the framework will automatically start or stop cloud instances to optimise the cost. In our research we implemented three different mathematical approaches to predict the duration: Polynomial, Radial basis functions and Kriging [129]. To build the prediction models the Java Jama matrix library is used. [130]

C.3.1 Polynomial

When using the polynomial regression technique, a polynomial will be constructed with runtimes of jobs that are already executed. The constructed polynomial model approximates the known points as good as possible using the least squares method. To limit the processing power required and prevent the polynomial from overfitting, the degree of the polynomial will be set independent of the number of points provided to construct the polynomial. This results in a polynomial which will not intersect with the provided points, but they will be approximated as good as possible using the least squares method (Figure 2).

C.3.2 Radial Basis Functions

The RBF model constructs a function around all the known runtimes. The weights of these functions will be adapted with the least squares method to approximate the model consisting of the existing durations as good as possible. RBF has a faster convergence and is more accurate in many situations. Different kernel functions [129], [131], [132] can be implemented in the radial basis functions. In the framework we used the following functions:

- Multiquadric function
- Inverse multiquadric function
- Epanechnikov kernel function

These enumerated functions are also referred to as kernel functions.

C.3.3 Kriging

Kriging is very similar to radial basis functions, but Kriging evaluates the kernels in every dimension of a sample[129] [132] this results in better results when multiple dimensions are used. The dimensions of the samples are defined by the number of parameters. Just as with radial basis functions the three described kernel functions are implemented in the framework.

C.3.4 Other prediction techniques

In related work[129] [132], the use of neural network were investigated, but results were very poor for this technique. Our developed framework is modularly constructed so that other prediction techniques can be easily incorporated.

C.3.5 Overhead

Each job has to be fetched from the database before it can be executed and after the execution the result has to be written to the database. These operations take about 1,5s on average in the test environment described in section 6. When a single parameter sweep consists of 1000 jobs we will have to take 1500s of overhead into account. The overhead is monitored and logged to the database so that the real total duration of the parameter sweep can be estimated. Equation (C.1) describes how the total duration is determined:

$$Totalruntime = (d + (\overline{o} * jobs))/workers \tag{C.1}$$

d is the predicted runtime
o is the measured overhead
Jobs is the total amount of jobs in the parameter sweep
Workers is the amount of active workers

C.4 Scaling

Scaling keeps the balance between the provided deadline and cost optimization. The scaler will evaluate the predicted remaining durations and consider whether the current deployed cloud resources are feasible to reach the deadline. If the predicted duration exceeds the deadline multiple successive times, extra resources are allocated. Multiple instances will be scaled at once until the predicted duration matches the deadline. The deadline is evaluated multiple times before actually scaling. If there is no hardware available to host a private cloud, it might be beneficial to use a public cloud provider. In this

thesis however, we assume there is enough hardware available to profitably host a private cloud and that this is more profitable than a public cloud. To reduce cost, the scaler will always opt for private instances unless these are already exhausted. The scaler will also monitor if the allocated resources are redundant. If the parameter sweep can finish on time without an instance, this instance will be terminated for further cost reduction.
In this thesis we used OpenNebula as the private Cloud Provider.[133] OpenNebula is an Open Source cloud management platform. OpenNebula has the ability to manage multiple hypervisors from one place.[134] Amazon EC2 was picked as public cloud provider because it is the biggest IAAS provider which provides an extensive Java SDK.**awssdk**

C.4.1 Scaling mode: Private, Public, Hybrid

Scaling modes provide more control over different clouds. When using private or public cloud, only the selected one will be used. In a hybrid environment, both clouds will be used, with the private cloud as the preferred cloud provider. If the private cloud provider is not able to provide the requested resources, public instances will be used instead. When using OpenNebula (ON) as the private cloud provider, this method of "cloud bursting" is also manageable by the private cloud provider[134]. If all resources available in the private ON network are exhausted, public instances can be ran and managed from ON. While these instances are not in deployed to hardware managed by OpenNebula, they can still be used as a native ON instance.

C.4.2 Strict versus Tolerant scaling

When using Amazon EC2 as a public cloud provider usage per hour will be charged **amazonpricing**. If the instance is only used for 5 minutes and terminated afterwards, a full hour will still be charged. To make a cost effective scaler, it is necessary to keep the remaining time of the instances in mind. When set to tolerant, scaling will not occur if the deadline will be missed, but there is enough time left in running instances to finish the parameter sweep. The maximum time passed a deadline with tolerant scaling is 1 hour. If the deadline is very important, the strict scaling method will deploy instances, without considering remaining time for the running instances. Strict and Tolerant scaling has no effect in a private cloud.

C.4.3 Private Greedy

When a private greedy scaler is used, all resources available to the private cloud will be allocated. Using This overallocation of resources will cause a parameter sweep to be processed as fast as possible. If a deadline for a specific parameter sweep requires more resources than the private cloud can provide, the scheduler would use all private resources, whether the private greedy method was used or not. Private greedy is only available when using a private cloud (private/hybrid mode).

C.5 Cloud framework

C.5.1 Overview

The architecture of the framework consists of three major components (Figure 3):

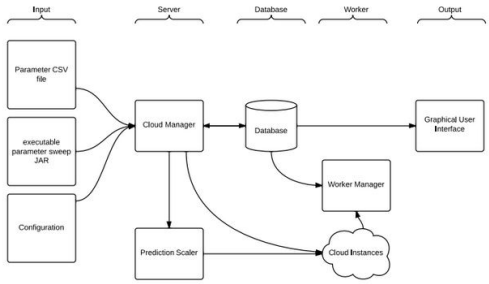

Figure C.3: Overview architecture of the cloud framework

- Input
- **The Cloud Manager**
- **The Database**
- **The Workers**
- Output

The cloud manager is the heart of the framework. It will generate all possible parameter combinations of a parameter sweep and it will write these parameter combinations to the database. At the same time the server installs, configures and launches the workers. The workers are public or private cloud instances operating on a Linux Ubuntu operating system. Once the workers are launched, they will get and execute the next job with the highest priority. The worker will then write the result and the execution time to the database. The server constantly reads the measured runtime of the jobs that are already executed from the database. Using these runtimes it can build a prediction model which estimates the total execution time of the entire parameter sweep. This process will repeat until all the jobs are solved.

C.5.2 Input

To submit a new parameter sweep three components are required. A parameter csv file will provide the parameters, a parameter sweep executable is a piece of code that will run multiple times and finally the configuration file defines how the framework interacts.

Parameter CSV

The CSV file contains a table with all possible input parameters. Each column is a different parameter. The framework will use this table to generate all the possible combinations. These parameter combinations will be written to the database.

Parameter Sweep Executable

The executable is the actual parameter sweep application. It is a Java JAR file that can be executed when all the parameters are passed as arguments. The application will return its result after completion.

Configuration

The whole configuration of the framework can be set using a JSON configuration file **json**. The most important configuration variables are discussed:

- *NumberOfInstances:* Defines the amount of workers to start with.
- *ParameterFile:* The path to the CSV file that contains all the parameters.
- *PredictionModel:* Defines the prediction technique that will be used.
- *PredictionKernel:* Defines the kernel used by the prediction technique.
- *LoggingEnabled:* When logging is enabled all the predictions are logged to the database. They can be reviewed using the web based GUI.
- *ConsoleAWSInstanceFeedback:* When enabled this will show all the installation feedback for each worker.
- *InstanceSize:* Defines the amount of resources necessary for an individual worker
- *Deadline:* Defines the deadline in dd/mm/yyyy hh:mm format.
- *Strict:* Defines the scaler to be strict or tolerant in the deadline (section C.4.2).
- *PrivateGreedy:* Defines whether or not the Private Greedy option will be used in the scaler (section C.4.3).

C.5.3 Cloud Manager

The cloud manager is an application written in Java that runs on the server. It handles and processes the input data and writes all the possible parameter combinations to a database. At the same time the cloud manager communicates with a cloud provider to launch and set up workers. Once the workers are launched and set up, an sftp connection is made to upload the parameter sweep and the workermanager executables. These executables are started using ssh commands.

The cloud manager will also assign priorities to the jobs. A job is referred to as a unique parameter combination, so each row in the database can be seen as a single job. The priority of the job influences the order of execution. The execution order is very important to build a reliable runtime prediction because each job logs its individual runtime which is then used to predict the runtime of the entire parameter sweep. By using priorities, there is no need for communication between the cloud manager and the active workers after these have been setup and the worker manager is started.

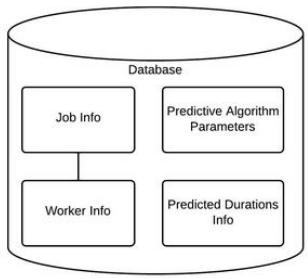

Figure C.4: Database structure

Prediction Scaler

The prediction scaler operates on a single thread which is started by the cloud manager application. The prediction scaler will read the runtimes from the jobs that are already done and fetch these runtimes to incorporate them in a prediction model which estimates the total execution time. The estimated runtime is then used to upscale or downscale by starting or terminating workers in order to meet a given deadline and to stay below the maximum cost. The prediction scaler is also responsible to log the predictions to the database.

C.5.4 Database

The database is used to act as a queue for all parameter combinations and to store additional information for the parameter sweep currently executed. The information is stored in different tables. (Figure 4).

The Job Info table contains all information about a job. A row in this table contains the parameter itself, the instanceID which processed the job, a state which determines if the job is already processed, the result of the processed parameters, a queue position and priority, an estimated and actual runtime and the overhead time.
The Worker Info table contains all information about all the workers for a parameter sweep. In this table, the worker ID, IP address and starttime are stored.
The Predictive Algorithm Parameters table contains all the information about the current predictive algorithm, the used kernel, sample selector and the time the prediction started.
The Predicted Durations Info table contains the total estimated runtimes and the number of points in the model used to obtain this estimation.

The database component of the framework is also interchangeable with other databases by implementing an abstract class. In this framework, a MySQL database with an InnoDB engine was used because of its row-locking features [135] **mysql**. Without these it might be possible for jobs to be fetched and processed multiple times. A MongoDB implementation for example with a shadow table which overcomes this shortcoming was implemented as well. Nevertheless MySQL was better suited to deal with the different synchronisation issues and for this selected as database component in this project.

C.5.5 Workers

Cloud instance

Cloud instances can be deployed to a private or public cloud. In our research OpenNebula was used as a VIM to create our own private cloud provider and Amazon EC2 as the public cloud provider. The greatest advantage to use a private cloud provider is that it runs on your own system, in a controlled environment, however resources are limited to the available hardware. To overcome this, a public cloud provider is also incorporated in the framework. The great advantage of using a public cloud is that there is a virtually unlimited amount of resources available on demand, but at a greater cost than a private cloud provider. Our framework can run in different configurations: public (using the public cloud provider only), private (using the private cloud provider only) and hybrid (using both cloud providers). How the public and private cloud provider complement each other in this hybrid configuration is discussed in section 2.4. Other cloud providers can be easily integrated in our framework by implementing an abstract class.

Worker manager

The worker manager is an application written in Java that will be executed on each worker. The worker manager connects to the database and fetches the next job with the highest priority. It will invoke the executable parameter sweep JAR file and record its runtime. The result, runtime, overhead and information about the instance which processed the job are written back to the database.

Graphical User Interface

The Graphical User Interface (GUI) collects data from the database and shows it using a web based interface. (Figure 5) The GUI is build for testing purposes making it much easier to evaluate different parameter sweep configurations. Three charts are shown using jqplot javascript library [136]. For each parameter sweep it will show the following information:

- General information about the parameter sweep:
  - Prediction model
  - Prediction kernel
  - Sample Selector
  - Starttime
  - Endtime
- Active workers:
  - Starttime
  - Number of jobs executed
  - Public IP address
- A chart that shows the estimated runtime in function of the position of all the jobs that are executed.

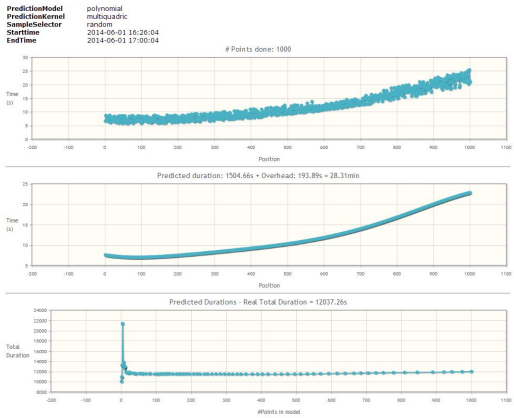

Figure C.5: GUI

- A chart that shows the predicted runtime in function of the position of each job.
- A chart that shows the total predicted execution time of the entire parameter sweep in function of the amount of samples into the prediction model. Each sample can be seen as the runtime of an executed job. The more samples, the more accurate the prediction model should be.

C.6 Test results

C.6.1 Test setup

In this test setup two simulations of a parameter sweep application were build. The simulation uses only one input parameter. We decided to use simulations instead of real parameter sweep applications because this gives us the possibility to fully control the runtime of each job. The input parameter immediately influences the runtime because this parameter is passed into a polynomial and kriging model, the output of these two models results into the final duration.

Each simulation uses a different polynomial and Kriging model and in one of the two simulations pseudo random noise was added. The use of only one parameter enables us to easily plot the data into a two-dimensional graph, which is more convenient to evaluate.

C.6.2 Results

In this section we will evaluate all implemented prediction techniques, scheduling techniques and sample selectors. All tests were executed ten times to be statistically significant.

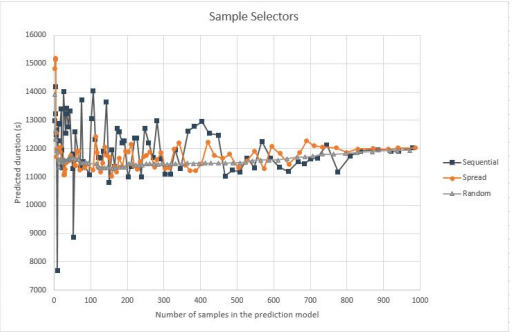

Figure C.6: Three sample selectors combined - Real runtime: 12003s

Sample selectors

Figure 6 shows the predicted total duration in function of the number of samples added into the prediction model. To come to these results the parameter sweep application with noise described in section 3.1 was used together with a radial basis function predictor and a multiquadric kernel. The real total duration was on average 1203 seconds.

The figure clearly illustrates that the sequential sample selector is of no use because it fluctuates too much. The spread sample selector performs better but still isn't perfect because it stabilizes only after 800 input samples. Optimization of the spread algorithm to decrease the gaps between the selected samples (Figure 1c) might improve this. The results of the random sample selector are very stable. Tests with other configurations and parameter sweep applications confirm these results.

Predictions

All three prediction techniques and all kernels discussed in section three were tested multiple times with different applications. The three best techniques are covered: Polynomial, Kriging with an inverse multiquadric kernel and RBF with a multiquadric kernel. The predicted duration in function of the number of samples added into the prediction model are plotted in Figure 7 and 8. Both tests were performed using a random sample selector. Figure 7 uses the parameter sweep application with noise and Figure 8 uses the parameter sweep application without noise.

Both graphs show that it takes a to long until the Kriging technique reaches a trustworthy estimate. Other techniques perform better, the RBF technique and the polynomial technique are well matched. However most of the tests with different parameter sweep applications pointed out that the RBF technique performs slightly better.

Scaling

- Test setup 1: No scaling, 2 instances started
- Test setup 2: Scaling, started with 2 instances (tolerant)
- Test setup 3: Scaling, started with 2 instances (strict)

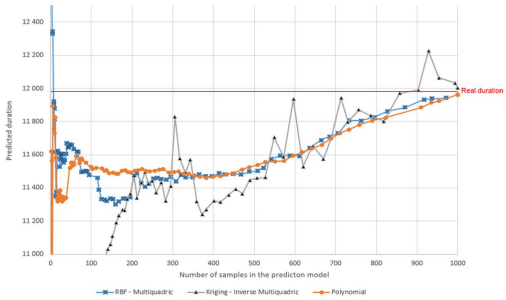

Figure C.7: Prediction techniques compared - Noise added

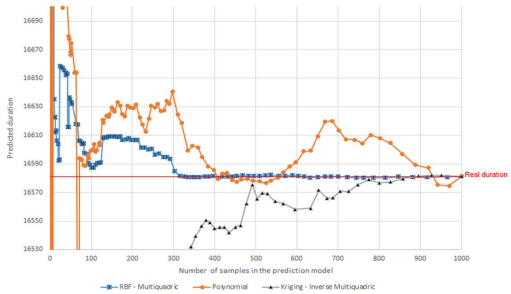

Figure C.8: Prediction techniques compared - Without noise

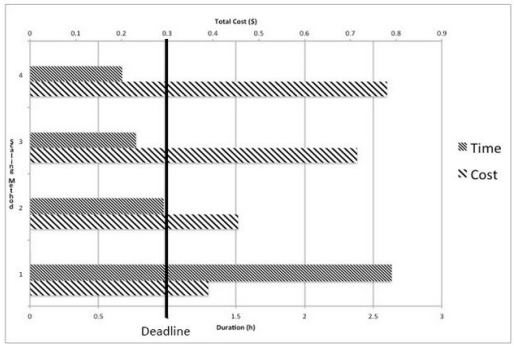

Figure C.9: Cost and duration of a parameter sweep with scaling

- Test setup 4: No scaling, 12 instances started

Tests were performed in Public mode, the deadline was set one hour later, using radial basis functions with a multiquadric kernel and a random sample selector.

As we can see in the results (Figure 9), scaling has a significant impact, even with a very short deadline and relatively simple parameter sweep. We can see that test setup one (without scaling) processed the full parameter sweep the fastest, but it is also the most expensive option.
The other test setup without scaling, setup four is the cheapest setup, but the deadline was abundantly exceeded. In both setups where scaling was used, the deadline was met, but when using the tolerant method, parameter sweep was executed as cost effective as possible.
The reason why the strict method is almost as expensive as the setup without scaling is because of its behaviour to strictly reach the deadline. When the predictor's estimation went up at the end, the scaler deployed extra instances just a few minutes before the deadline, while the tolerant method would consider the remaining time of the running instances.

C.7 Cost effective scheduling

Until now we focused in this paper solely on reserved instances in a private or public cloud environment. The advantage of working with public reserved instances is that the required cost is fixed and that your instance is operational during a predefined or virtually endless period of time. This fixed approach makes it easy for our model to predict runtimes and to predict the cost that will be required to run certain jobs as demonstrated in the earlier sections of this paper.
However, when the main goal is to run these calculations as cost-effective as possible, having a fixed price might not be the best way, especially not when knowing that during certain periods of time the load on the overall public cloud infrastructure is low as a result of a lower demand. And as basic economics prescribes, a low demand should decrease the cost when the supply remains constant.

This is exactly what the Amazon AWS platform tries to achieve with their spot market. It allows users to bid on EC2 instances and when the demand is low, amazon will be able to offer their cloud resources at a much lower cost compared to the reserved resources (Figure 10). But note that this works both ways, when the demand is high, the biddings will increase and as a result the cost of the cloud resources will increase as well. Because of that, the Amazon platform allows you to bid a certain price for using their spot market resources, and as long as the market price is below your bidding amount you will be able to make use of these resources. However, when the market price exceeds your bidding amount the processes running on the spot market resources will be terminated.

C.7.1 Integrating Amazon spot market

Leveraging the Amazon spot market as a tool to lower the cost of running parameter sweep calculations on public cloud instances can be very powerful and can result in major cost savings but will come with extra challenges and a more complex prediction model.

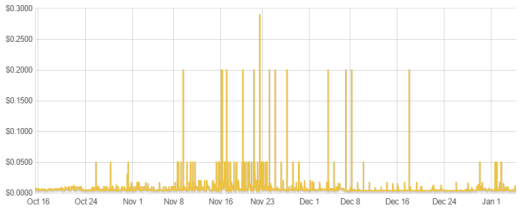

Figure C.10: Spot market price fluctuations for a linux micro instance (Oct 2016-Jan 2017)

In this section we will discuss the challenges necessary to overcome in order to integrate the amazon spot market in our framework.

Reliability

As mentioned in the previous section the reliability of using spot market resources will decrease severely compared to the reserved cloud instances. Using historical pricing data, which is publicly available for the last three months, we can use predictions techniques to predict when the demand will be at its lowest. Using these predictions, we can then automatically schedule certain parameter sweep jobs during low-demand periods. The predictions can also be used to determine the most optimal bidding price that provides some certain level of reliability and remains below the default price for reserved instances at the same time. Using the public spot market API, offered by amazon we can then fully automate the bidding and orchestration process of the spot market instances.

Interruption safe model

Although the prediction algorithms can provide a very powerful way to schedule parameter sweep calculations in the spot market, we still have to prepare for interruptions caused by unexpected rise of demand and price. This can be achieved using a termination message that will be send to each spot instance two minutes prior to termination. This functionality is offered by the Amazon AWS platform by default. During this two minute period the instances will off-load the state of the running calculations to an Amazon S3 cloud storage container. This will then be automatically logged in the database of our system, so that when the prices decrease again below our optimal bidding price we can automatically relaunch the terminated instances and continue from where we left off by fetching the calculation state from the S3 container to the relaunched spot instances.

Hybrid model

The disadvantage of solely relying on spot instances is that it will be much harder to complete the jobs before a certain deadline. This is because the system when the spot market demand is too high the predicted schedules will prescribe to wait a certain period of time. To mitigate this, we propose a hybrid model that combines the use of one or more reserved instance, so that when the spot instances have to be terminated, the reserved instance can add these running calculations to their job-queue. This allows the system to continuously make progress instead of being idle while still combining the benefits of having an acceptable level of stability at a much lower cost.

Although, it must be noted that the tradeoff between cost and deadline will have to be made for each individual use case separately.

C.8 Conclusion

Different techniques are discussed on how a parameter sweep application can be cost efficiently scheduled in a hybrid cloud environment while respecting the deadline. Prediction techniques are used to determine the remaining execution time. The radial basis function with a multiquadric kernel together with the polynomial technique showed the best results depending on the application. In order to schedule the jobs of the parameter sweep a sample selector was used. The sample selector assigns priorities to each job, the higher the priority the sooner it will be executed. The execution order has a great impact on the predicted execution times. Results showed that the random sample selector as the better one. The spread algorithm is also very promising but it needs further optimization. The implementation of various scaling methods makes it possible to launch extra instances in the private or public cloud when the estimated execution time exceeds the deadline. On the other hand the scaler will also downscale to not over allocate resources and thus reduce the cost. Globally we can conclude extensive cost optimisation in deadline dependent parameter sweeps can be reached by introducing runtime prediction in scaling and load balancing algorithms on a public, private or hybrid cloud environment. The theoretical hybrid spot market model we discuss promises great results that can further downsize the costs required to run parameter sweep calculations in the public cloud. However it comes with a reduced reliability which will impact runtimes of the system compared to the original approach.

C.9 Future Work

We want to further reduce the overhead when launching new parameter sweep jobs. Also we want to increase the performance of the predictive models by reducing overfitting issues that exist in the current models. Finally, we believe that integrating the theoretical spot market model might largely reduce costs required to run parameter sweep calculations in a public cloud environment.

Bibliography

[1] M. J. Mataric, 'Designing emergent behaviors: From local interactions to collective intelligence,' in *Proceedings of the 2nd International Conference on Simulation of Adaptive Behavior*, 1993 (cited on pp. 2, 18).

[2] D. Roca, D. Nemirovsky, M. Nemirovsky, R. Milito and M. Valero, 'Emergent behaviors in the internet of things: The ultimate ultra-large-scale system,' *IEEE micro*, vol. 36, no. 6, pp. 36–44, 2016 (cited on pp. 2, 18).

[3] M. D. Hill and M. R. Marty, 'Amdahl's law in the multicore era,' *Computer*, vol. 41, no. 7, pp. 33–38, 2008 (cited on p. 2).

[4] S. Bosmans, S. Mercelis, P. Hellinckx and J. Denil, 'Towards evaluating emergent behavior of the internet of things using large scale simulation techniques (wip),' in *Proceedings of the Theory of Modeling and Simulation Symposium*, Society for Computer Simulation International, 2018, p. 4 (cited on pp. 3, 27, 94).

[5] C. M. Macal and M. J. North, 'Tutorial on agent-based modeling and simulation,' in *Simulation Conference, 2005 Proceedings of the Winter*, IEEE, 2005, 14–pp (cited on pp. 3, 27).

[6] B. P. Zeigler, H. Praehofer and T. G. Kim, *Theory of modeling and simulation: integrating discrete event and continuous complex dynamic systems*. Academic press, 2000 (cited on pp. 3, 49).

[7] G. Wainer, 'Cd++: A toolkit to develop devs models,' *Software: Practice and Experience*, vol. 32, no. 13, pp. 1261–1306, 2002 (cited on p. 3).

[8] R. M. Fujimoto, *Parallel and distributed simulation systems*. Wiley New York, 2000, vol. 300 (cited on p. 4).

[9] F. K. Frantz, 'A taxonomy of model abstraction techniques,' in *Proceedings of the 27th conference on Winter simulation*, IEEE Computer Society, 1995, pp. 1413–1420 (cited on p. 4).

[10] D. Caughlin and A. F. Sisti, 'Summary of model abstraction techniques,' in *Enabling Technology for Simulation Science*, International Society for Optics and Photonics, vol. 3083, 1997, pp. 2–14 (cited on pp. 4, 6).

[11] R. R. Barton, 'Metamodeling: A state of the art review,' in *Simulation Conference Proceedings, 1994. Winter*, IEEE, 1994, pp. 237–244 (cited on p. 6).

[12] P. Symonds, J. Taylor, Z. Chalabi and M. Davies, 'Performance of neural networks vs. rbf when forming a metamodel for residential buildings,' *International Journal Of Civil, Environmental, Structural, Construction And Architectural Engineering*, vol. 9, no. 12, pp. 1446–1450, 2015 (cited on p. 6).

[13] R. Gore, S. Diallo, C. Lynch and J. Padilla, 'Augmenting bottom-up metamodels with predicates,' *Journal of Artificial Societies and Social Simulation*, vol. 20, no. 1, 2017 (cited on p. 6).

[14] J. D. Rodriguez, K. W. Bauer Jr, J. O. Miller and R. E. Neher Jr, 'Building prediction models of large hierarchical simulation models with artificial neural networks and other statistical techniques.,' in *Visual Information Processing*, 2008, p. 69 780 (cited on pp. 6, 39).

[15] P. Bogacki and L. F. Shampine, 'An efficient runge-kutta (4, 5) pair,' *Computers & Mathematics with Applications*, vol. 32, no. 6, pp. 15–28, 1996 (cited on p. 6).

[16] V. Stojkovski, 'Multi-abstraction, multi-formalism modelling and simulation: Integrating forrester system dynamics, cellular automata and agent based modelling,' M.S. thesis, University of Antwerp, Belgium, 2015 (cited on p. 7).

[17] L. Bononi, M. Bracuto, G. D'Angelo and L. Donatiello, 'Proximity detection in distributed simulation of wireless mobile systems,' in *Proceedings of the 9th ACM international symposium on Modeling analysis and simulation of wireless and mobile systems*, ACM, 2006, pp. 44–51 (cited on p. 8).

[18] Y. Van Tendeloo and H. Vangheluwe, 'Activity in pythonpdevs,' in *ITM Web of Conferences*, EDP Sciences, vol. 3, 2014, p. 01 002 (cited on pp. 8, 9, 28).

[19] G. D'Angelo, 'Parallel and distributed simulation from many cores to the public cloud,' in *High Performance Computing and Simulation (HPCS), 2011 International Conference on*, IEEE, 2011 (cited on p. 8).

[20] G. D'Angelo, 'The simulation model partitioning problem: An adaptive solution based on self-clustering,' *Simulation Modelling Practice and Theory*, vol. 70, pp. 1–20, 2017 (cited on pp. 9, 29, 32).

[21] A. Muzy, L. Touraille, H. Vangheluwe, O. Michel, M. K. Traoré and D. R. Hill, 'Activity regions for the specification of discrete event systems,' in *Proceedings of the 2010 Spring Simulation Multiconference*, Society for Computer Simulation International, 2010, p. 136 (cited on pp. 9, 28).

[22] L. Lamport, 'Time, clocks, and the ordering of events in a distributed system,' *Communications of the ACM*, vol. 21, no. 7, pp. 558–565, 1978 (cited on p. 10).

[23] D. M. Nicol, 'Principles of conservative parallel simulation,' in *Proceedings of the 28th conference on Winter simulation*, IEEE Computer Society, 1996, pp. 128–135 (cited on p. 10).

[24] J. Misra, 'Distributed discrete-event simulation,' *ACM Computing Surveys (CSUR)*, vol. 18, no. 1, pp. 39–65, 1986 (cited on p. 10).

[25] G. D'Angelo and M. Marzolla, 'New trends in parallel and distributed simulation: From many-cores to cloud computing,' *Simulation Modelling Practice and Theory*, vol. 49, pp. 320–335, 2014 (cited on p. 10).

[26] D. R. Jefferson, 'Virtual time,' *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 7, no. 3, pp. 404–425, 1985 (cited on p. 10).

[27] S. Bosmans, S. Mercelis, J. Denil and P. Hellinckx, 'Testing iot systems using a hybrid simulation based testing approach,' *Computing*, vol. 101, no. 7, pp. 857–872, 2019 (cited on pp. 13, 15).

[28] S. Bosmans, T. Bogaerts, W. Casteels, S. Mercelis, J. Denil and P. Hellinckx, 'Adaptivity in distributed agent-based simulation: A generic load-balancing approach,' in *International Workshop on Multi-Agent Systems and Agent-Based Simulation*, Springer, 2020, pp. 1–12 (cited on p. 14).

[29] S. Bosmans, S. Mercelis, P. Hellinckx and J. Denil, 'Reducing computational cost of large-scale simulations using opportunistic model approximation,' in *SpringSim 19* (cited on pp. 14, 29, 58).

[30] S. Bosmans, S. Mercelis, M. Ceulemans, J. Denil and P. Hellinckx, 'Acsim: Towards hyper-scalable internet of things simulation,' in *International Conference on P2P, Parallel, Grid, Cloud and Internet Computing*, Springer, 2017, pp. 743–750 (cited on p. 14).

[31] S. Bosmans, G. Maricaux, F. Van Der Schueren and P. Hellinckx, 'Cost-aware hybrid cloud scheduling of parameter sweep calculations using predictive algorithms,' *International Journal of Grid and Utility Computing*, vol. 10, no. 1, pp. 63–75, 2019 (cited on p. 14).

[32] A. Bertolino, 'Software testing research: Achievements, challenges, dreams,' in *2007 Future of Software Engineering*, IEEE Computer Society, 2007, pp. 85–103 (cited on p. 17).

[33] S. Nidhra and J. Dondeti, 'Black box and white box testing techniques-a literature review,' *International Journal of Embedded Systems and Applications (IJESA)*, vol. 2, no. 2, pp. 29–50, 2012 (cited on p. 17).

[34] B. Beizer, *Black-box testing: techniques for functional testing of software and systems*. John Wiley & Sons, Inc., 1995 (cited on p. 17).

[35] S. Latre, P. Leroux, T. Coenen, B. Braem, P. Ballon and P. Demeester, 'City of things: An integrated and multi-technology testbed for iot smart city experiments,' in *Smart Cities Conference (ISC2), 2016 IEEE International*, IEEE, 2016, pp. 1–8 (cited on p. 17).

[36] L. Sanchez, L. Muñoz, J. A. Galache, P. Sotres, J. R. Santana, V. Gutierrez, R. Ramdhany, A. Gluhak, S. Krco, E. Theodoridis *et al.*, 'Smartsantander: Iot experimentation over a smart city testbed,' *Computer Networks*, vol. 61, pp. 217–238, 2014 (cited on p. 17).

[37] G. Carneiro, 'Ns-3: Network simulator 3,' in *UTM Lab Meeting April*, vol. 20, 2010 (cited on pp. 17, 96).

[38] O. Developer, *Omnet++ network simulation frame-work* (cited on p. 17).

[39] A. Dunkels, B. Gronvall and T. Voigt, 'Contiki-a lightweight and flexible operating system for tiny networked sensors,' in *Local Computer Networks, 2004. 29th Annual IEEE International Conference on*, IEEE, 2004, pp. 455–462 (cited on p. 17).

[40] P. Levis, N. Lee, M. Welsh and D. Culler, 'Tossim: Accurate and scalable simulation of entire tinyos applications,' in *Proceedings of the 1st international conference on Embedded networked sensor systems*, ACM, 2003, pp. 126–137 (cited on p. 17).

[41] H. Gupta, A. Vahid Dastjerdi, S. K. Ghosh and R. Buyya, 'Ifogsim: A toolkit for modeling and simulation of resource management techniques in the internet of things, edge and fog computing environments,' *Software: Practice and Experience*, vol. 47, no. 9, pp. 1275–1296, 2017 (cited on p. 17).

[42] G. D'Angelo, S. Ferretti and V. Ghini, 'Simulation of the internet of things,' in *High Performance Computing & Simulation (HPCS), 2016 International Conference on*, IEEE, 2016, pp. 1–8 (cited on p. 17).

[43] D. S. Nunes, P. Zhang and J. S. Silva, 'A survey on human-in-the-loop applications towards an internet of all,' *IEEE Communications Surveys & Tutorials*, vol. 17, no. 2, pp. 944–965, 2015 (cited on pp. 19, 95).

[44] G. Fortino, R. Gravina, W. Russo and C. Savaglio, 'Modeling and simulating internet-of-things systems: A hybrid agent-oriented approach,' *Computing in Science & Engineering*, vol. 19, no. 5, pp. 68–76, 2017 (cited on pp. 19, 96).

[45] M. Marjanović, A. Antonić and I. P. Žarko, 'Edge computing architecture for mobile crowdsensing,' *IEEE Access*, vol. 6, pp. 10 662–10 674, 2018 (cited on p. 20).

[46] D. A. Crisan, I. E. Radoi and D. Arvind, 'Coap-mediated hybrid simulation and visualisation environment for specknets,' in *Proceedings of the 1st ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, ACM, 2013, pp. 285–294 (cited on p. 20).

[47] J. A. Murray, M. Sasani and X. Shao, 'Hybrid simulation for system-level structural response,' *Engineering Structures*, vol. 103, pp. 228–238, 2015 (cited on p. 20).

[48] A. Arora, E. Ertin, R. Ramnath, M. Nesterenko and W. Leal, 'Kansei: A high-fidelity sensing testbed,' *IEEE Internet Computing*, vol. 10, no. 2, pp. 35–47, 2006 (cited on p. 20).

[49] C. Bormann, A. P. Castellani and Z. Shelby, 'Coap: An application protocol for billions of tiny internet nodes,' *IEEE Internet Computing*, vol. 16, no. 2, pp. 62–67, 2012 (cited on p. 21).

[50] U. Hunkeler, H. L. Truong and A. Stanford-Clark, 'Mqtt-s—a publish/subscribe protocol for wireless sensor networks,' in *Communication systems software and middleware and workshops, 2008. comsware 2008. 3rd international conference on*, IEEE, 2008, pp. 791–798 (cited on p. 21).

[51] S. Baradaran, N. Maleknasr, S. Setayeshi and M. E. Akbari, 'Prediction of lung cells oncogenic transformation for induced radon progeny alpha particles using sugarscape cellular automata,' *Iranian journal of cancer prevention*, vol. 7, no. 1, p. 40, 2014 (cited on pp. 27, 36).

[52] M. Raberto, S. Cincotti, S. M. Focardi and M. Marchesi, 'Agent-based simulation of a financial market,' *Statistical Mechanics and its Applications*, vol. 299, no. 1-2, 2001 (cited on p. 27).

[53] M. Balmer, N. Cetin, K. Nagel and B. Raney, 'Towards truly agent-based traffic and mobility simulations,' in *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems, 2004. AAMAS 2004.*, IEEE, 2004 (cited on p. 27).

[54] W. K. V. Chan, Y.-J. Son and C. M. Macal, 'Agent-based simulation tutorial-simulation of emergent behavior and differences between agent-based simulation and discrete-event simulation,' in *Proceedings of the 2010 winter simulation conference*, IEEE, 2010, pp. 135–150 (cited on p. 27).

[55] Q. Long, J. Lin and Z. Sun, 'Agent scheduling model for adaptive dynamic load balancing in agent-based distributed simulations,' *Simulation Modelling Practice and Theory*, vol. 19, no. 4, pp. 1021–1034, 2011 (cited on pp. 28, 29).

[56] Y. Xu, W. Cai, H. Aydt and M. Lees, 'Efficient graph-based dynamic load-balancing for parallel large-scale agent-based traffic sim,' in *WinterSim*, IEEE, 2014 (cited on pp. 29, 33).

[57] A. Boukerche, 'An adaptive partitioning algorithm for distributed discrete event simulation systems,' *Journal of Parallel and Distributed Computing*, vol. 62, no. 9, 2002 (cited on p. 29).

[58] R. Franceschini, M. Challenger, A. Cicchetti, J. Denil and H. Vangheluwe, 'Challenges for automation in adaptive abstraction,' in *2019 ACM/IEEE 22nd int. Conference on MDE Languages and Systems Companion (MODELS-C)*, IEEE, 2019 (cited on p. 29).

[59] D. Masad and J. Kazil, 'Mesa: An agent-based modeling framework,' in *14th PYTHON in Science Conference*, 2015, pp. 53–60 (cited on pp. 30, 63, 99).

[60] IBM, 'An architectural blueprint for autonomic computing.,' 2006 (cited on p. 31).

[61] D. G. D. L. Iglesia and D. Weyns, 'Mape-k formal templates to rigorously design behaviors for self-adaptive systems,' *ACM TAAS*, vol. 10, no. 3, pp. 1–31, 2015 (cited on p. 31).

[62] M. Treiber and A. Kesting, 'Traffic flow dynamics,' *Traffic Flow Dynamics: Data, Models and Simulation, Springer-Verlag Berlin Heidelberg*, 2013 (cited on pp. 33, 57, 63).

[63] A. Kesting, M. Treiber and D. Helbing, 'General lane-changing model mobil for car-following models,' *Transportation Research Record*, vol. 1999, no. 1, pp. 86–94, 2007 (cited on pp. 33, 57, 64).

[64] G. Karypis and V. Kumar, 'Multilevel algorithms for multi-constraint graph partitioning,' in *SC'98: Proceedings of the 1998 ACM/IEEE Conference on Supercomputing*, IEEE, 1998, pp. 28–28 (cited on p. 33).

[65] T. Potuzak, 'Distributed traffic simulation and the reduction of inter-process communication using traffic flow characteristics transfer,' in *Tenth International Conference on Computer Modeling and Simulation*, IEEE, 2008, pp. 525–530 (cited on p. 33).

[66] K. Ramamohanarao and et. al., 'Smarts: Scalable microscopic adaptive road traffic simulator,' *ACM TIST*, vol. 8, no. 2, pp. 1–22, 2016 (cited on p. 33).

[67] G. Cordasco, V. Scarano and C. Spagnuolo, 'Distributed mason: A scalable distributed multi-agent simulation environment,' *Simulation Modelling Practice and Theory*, vol. 89, pp. 15–34, 2018 (cited on p. 33).

[68] J. M. Epstein and R. Axtell, *Growing artificial societies: social science from the bottom up.* Brookings Institution Press, 1996 (cited on p. 36).

[69] S. Bosmans, S. Mercelis, J. Denil and P. Hellinckx, 'Challenges of modeling and simulating internet of things systems,' in *International Conference on P2P, Parallel, Grid, Cloud and Internet Computing*, Springer, 2018, pp. 457–466 (cited on p. 42).

[70] C. E. Shannon, 'Prediction and entropy of printed english,' *Bell system tech journal*, vol. 30, 1951 (cited on p. 45).

[71] R. Lamarche-Perrin, Y. Demazeau and J.-M. Vincent, 'How to build the best macroscopic description of your multi-agent system?' In *International Conference on Practical Applications of Agents and Multi-Agent Systems*, Springer, 2013, pp. 157–169 (cited on p. 46).

[72] M. K. Traoré and A. Muzy, 'Capturing the dual relationship between simulation models and their context,' *Simulation Modelling Practice and Theory*, vol. 14, no. 2, pp. 126–142, 2006 (cited on p. 49).

[73] J. Denil, S. Klikovits, P. J. Mosterman, A. Vallecillo and H. Vangheluwe, 'The experiment model and validity frame in m&s,' in *Proceedings of the Symposium on Theory of Modeling & Simulation*, Society for Computer Simulation International, 2017, p. 10 (cited on p. 49).

[74] B. Barroca, T. Kühne and H. Vangheluwe, 'Integrating language and ontology engineering.,' in *MPM@ MoDELS*, Citeseer, 2014, pp. 77–86 (cited on p. 49).

[75] J. Casas, J. L. Ferrer, D. Garcia, J. Perarnau and A. Torday, 'Traffic simulation with aimsun,' in *Fundamentals of traffic simulation*, Springer, 2010, pp. 173–232 (cited on pp. 53, 61).

[76] W. Burghout, H. N. Koutsopoulos and I. Andreasson, 'Hybrid mesoscopic–microscopic traffic simulation,' *Transportation Research Record*, vol. 1934, no. 1, pp. 218–225, 2005 (cited on pp. 53, 57).

[77] B. P. Zeigler, A. Muzy and E. Kofman, *Theory of modeling and simulation: discrete event & iterative system computational foundations*. Academic press, 2018 (cited on pp. 54, 56).

[78] Y. Iwasaki, 'Reasoning with multiple abstraction models,' *Recent advances in qualitative physics*, pp. 67–82, 1992 (cited on p. 55).

[79] G. Morvan, 'Multi-level agent-based modeling-a literature survey,' *arXiv preprint arXiv:1205.0561*, 2012 (cited on p. 55).

[80] P. Mathieu, G. Morvan and S. Picault, 'Multi-level agent-based simulations: Four design patterns,' *Simulation Modelling Practice and Theory*, vol. 83, pp. 51–64, 2018 (cited on p. 56).

[81] R. Franceschini, S. Van Mierlo and H. Vangheluwe, 'Towards adaptive abstraction in agent based simulation,' in *2019 Winter Simulation Conference (WSC)*, IEEE, 2019, pp. 2725–2736 (cited on p. 58).

[82] N. Bouha, G. Morvan, H. Abouaissa and Y. Kubera, 'A first step towards dynamic hybrid traffic modeling,' *arXiv preprint arXiv:1505.07257*, 2015 (cited on p. 58).

[83] A. Messner and M. Papageorgiou, 'Metanet: A macroscopic simulation program for motorway networks,' *Traffic engineering & control*, vol. 31, no. 8-9, pp. 466–470, 1990 (cited on p. 58).

[84] I. T. Haman, V. C. Kamla, S. Galland and J. C. Kamgang, 'Towards an multilevel agent-based model for traffic simulation,' *Procedia Computer Science*, vol. 109, pp. 887–892, 2017 (cited on p. 58).

[85] P. Mella, *The Holonic Revolution Holons, Holarchies and Holonic Networks. The Ghost in the Production Machine*. Jan. 2009, ISBN: 978-88-96764-00-8. DOI: 10.13140/2.1.1954.5922 (cited on p. 58).

[86] A. Koestler, 'The ghost in the machine.,' 1968 (cited on p. 58).

[87] N. Gaud, S. Galland, F. Gechter, V. Hilaire and A. Koukam, 'Holonic multilevel simulation of complex systems: Application to real-time pedestrians simulation in virtual urban environment,' *Simulation Modelling Practice and Theory*, vol. 16, no. 10, pp. 1659–1676, 2008 (cited on p. 58).

[88] M. Fellendorf and P. Vortisch, 'Microscopic traffic flow simulator vissim,' in *Fundamentals of traffic simulation*, Springer, 2010, pp. 63–93 (cited on p. 61).

[89] M. Behrisch, L. Bieker, J. Erdmann and D. Krajzewicz, 'Sumo–simulation of urban mobility: An overview,' in *Proceedings of SIMUL 2011, The Third International Conference on Advances in System Simulation*, ThinkMind, 2011 (cited on p. 61).

[90] M. Haklay and P. Weber, 'Openstreetmap: User-generated street maps,' *Ieee Pervas Comput*, vol. 7, no. 4, pp. 12–18, 2008 (cited on pp. 61, 97).

[91] P. Arcaini, E. Riccobene and P. Scandurra, 'Modeling and analyzing mape-k feedback loops for self-adaptation,' in *2015 IEEE/ACM 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, IEEE, 2015, pp. 13–23 (cited on p. 62).

[92] S. Peeta and A. K. Ziliaskopoulos, 'Foundations of dynamic traffic assignment: The past, the present and the future,' *Networks and spatial economics*, vol. 1, no. 3-4, pp. 233–265, 2001 (cited on p. 65).

[93] M. J. Lighthill and G. B. Whitham, 'On kinematic waves ii. a theory of traffic flow on long crowded roads,' *Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences*, vol. 229, no. 1178, pp. 317–345, 1955 (cited on p. 65).

[94] D. Vikram, P. Chakroborty and S. Mittal, 'Exploring the behavior of lwr continuum models of traffic flow in presence of shock waves,' *Procedia-Social and Behavioral Sciences*, vol. 104, pp. 412–421, 2013 (cited on p. 65).

[95] G. Dervisoglu, G. Gomes, J. Kwon, R. Horowitz and P. Varaiya, 'Automatic calibration of the fundamental diagram and empirical observations on capacity,' in *Transportation Research Board 88th Annual Meeting*, vol. 15, 2009 (cited on p. 65).

[96] C. Osorio and K. K. Selvam, 'Simulation-based optimization: Achieving computational efficiency through the use of multiple simulators,' *Transportation Science*, vol. 51, no. 2, pp. 395–411, 2017 (cited on p. 79).

[97] C. Osorio and M. Bierlaire, 'A surrogate model for traffic optimization of congested networks: An analytic queueing network approach,' Tech. Rep., 2009 (cited on p. 79).

[98] R. A. Gil, Z. C. Johanyák, T. Kovács *et al.*, 'Surrogate model based optimization of traffic lights cycles and green period ratios using microscopic simulation and fuzzy rule interpolation,' *Int. J. Artif. Intell*, vol. 16, no. 1, pp. 20–40, 2018 (cited on p. 79).

[99] E. I. Vlahogianni, 'Optimization of traffic forecasting: Intelligent surrogate modeling,' *Transportation Research Part C: Emerging Technologies*, vol. 55, pp. 14–23, 2015 (cited on p. 79).

[100] X. Chen, L. Zhang, X. He, C. Xiong and Z. Li, 'Surrogate-based optimization of expensive-to-evaluate objective for optimal highway toll charges in transportation network,' *Computer-Aided Civil and Infrastructure Engineering*, vol. 29, no. 5, pp. 359–381, 2014 (cited on p. 79).

[101] E. I. Vlahogianni, M. G. Karlaftis and J. C. Golias, 'Short-term traffic forecasting: Where we are and where we're going,' *Transportation Research Part C: Emerging Technologies*, vol. 43, pp. 3–19, 2014, ISSN: 0968090X (cited on p. 80).

[102] B. Yu, H. Yin and Z. Zhu, 'Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting,' *arXiv preprint arXiv:1709.04875*, 2017 (cited on p. 80).

[103] C. Zheng, X. Fan, C. Wang and J. Qi, 'Gman: A graph multi-attention network for traffic prediction,' in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, 2020, pp. 1234–1241 (cited on p. 80).

[104] S. Casas, C. Gulino, R. Liao and R. Urtasun, 'Spatially-aware graph neural networks for relational behavior forecasting from sensor data,' *arXiv preprint arXiv:1910.08233*, 2019 (cited on p. 80).

[105] H. He, H. Dai and N. Wang, 'Ust: Unifying spatio-temporal context for trajectory prediction in autonomous driving,' *arXiv preprint arXiv:2005.02790*, 2020 (cited on p. 80).

[106] J. Zhou, G. Cui, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li and M. Sun, 'Graph neural networks: A review of methods and applications,' *arXiv preprint arXiv:1812.08434*, 2018 (cited on p. 80).

[107] S. Chan, 'Complex adaptive systems,' in *ESD. 83 research seminar in engineering systems*, vol. 31, 2001, p. 1 (cited on p. 93).

[108] S. Laghari and M. A. Niazi, 'Modeling the internet of things, self-organizing and other complex adaptive communication networks: A cognitive agent-based computing approach,' *PloS one*, vol. 11, no. 1, e0146760, 2016 (cited on p. 93).

[109] S. Bosmans, S. Mercelis, J. Denil and P. Hellinckx, 'Testing iot systems using a hybrid simulation based testing approach,' *Computing*, pp. 1–16, 2018 (cited on p. 94).

[110] R. M. Fujimoto, 'Parallel simulation: Distributed simulation systems,' in *35th proceedings on Winter sim conf.*, Winter Simulation Conference, 2003, pp. 124–134 (cited on p. 94).

[111] H. Yu, Z. Shen and C. Leung, 'From internet of things to internet of agents,' in *Green Computing and Communications (GreenCom), 2013 IEEE and Internet of Things (iThings/CPSCom), IEEE International Conference on and IEEE Cyber, Physical and Social Computing*, IEEE, 2013, pp. 1054–1057 (cited on pp. 94, 96).

[112] K. Mehdi, M. Lounis, A. Bounceur and T. Kechadi, 'Cupcarbon: A multi-agent and discrete event wireless sensor network design and simulation tool,' in *Proceedings of the 7th International ICST Conference on Simulation Tools and Techniques*, 2014, pp. 126–131 (cited on pp. 95, 96).

[113] A. Varga and R. Hornig, 'An overview of the omnet++ simulation environment,' in *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*, ICST, 2008, p. 60 (cited on p. 95).

[114] G. D'Angelo, S. Ferretti and V. Ghini, 'Multi-level simulation of internet of things on smart territories,' *Simulation Modelling Practice and Theory*, vol. 73, pp. 3–21, 2017 (cited on pp. 95, 96).

[115] K. Batool and M. A. Niazi, 'Modeling the internet of things: A hybrid modeling approach using complex networks and agent-based models,' *CASM*, vol. 5, no. 1, p. 4, 2017 (cited on p. 96).

[116] P. Wehner and D. Göhringer, 'Internet of things simulation using omnet++ and hardware in the loop,' in *Components and Services for IoT Platforms*, Springer, 2017, pp. 77–87 (cited on p. 96).

[117] S. Karnouskos and T. N. De Holanda, 'Simulation of a smart grid city with software agents.,' *EMS*, vol. 9, pp. 424–429, 2009 (cited on p. 96).

[118] C. M. Macal and M. J. North, 'Tutorial on agent-based modelling and simulation,' *Journal of simulation*, vol. 4, no. 3, pp. 151–162, 2010 (cited on p. 96).

[119] N. Beckmann, H.-P. Kriegel, R. Schneider and B. Seeger, 'The r*-tree: An efficient and robust access method for points and rectangles,' in *Acm Sigmod Record*, Acm, vol. 19, 1990, pp. 322–331 (cited on p. 97).

[120] S. Abar, G. K. Theodoropoulos, P. Lemarinier and G. M. O'Hare, 'Agent based modelling and simulation tools: A review of the state-of-art software,' *Computer Science Review*, vol. 24, pp. 13–33, 2017 (cited on p. 97).

[121] M. Lom and O. Přibyl, 'Modeling of smart city building blocks using multi-agent systems,' *Neural Network World*, vol. 27, no. 4, p. 317, 2017 (cited on p. 98).

[122] A. S. Rao, 'Agentspeak (l): Bdi agents speak out in a logical computable language,' in *European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, Springer, 1996, pp. 42–55 (cited on p. 98).

[123] H. Kavak, J. J. Padilla, C. J. Lynch and S. Y. Diallo, 'Big data, agents, and machine learning: Towards a data-driven agent-based modeling approach,' in *Proceedings of the Annual Simulation Symposium*, Society for Computer Simulation International, 2018, p. 12 (cited on p. 99).

[124] U. of California, *Open-source software for volunteer computing and grid computing*, 2001. [Online]. Available: `http://boinc.berkeley.edu/` (visited on 16/05/2014) (cited on p. 101).

[125] P. Hellinckx, F. Arickx, J. Broeckhove and G. Stuer, 'The cobra grid: A highly configurable lightweight grid,' *International journal of web and grid services*, vol. 3, no. 3, pp. 267–286, 2007 (cited on p. 101).

[126] G. M. Stig Bosmans and F. V. der Schueren, 'Predictive algorithms for scheduling parameter sweep calculations in a cloud environment,' English, *International Conference on P2P, Parallel, Grid, Cloud and Internet Computing*, vol. 1, no. 3, pp. 789–798, 2016, ISSN: 978-3-319-49108-0. [Online]. Available: `http://link.springer.com/chapter/10.1007/978-3-319-49109-7_76` (cited on p. 101).

[127] L. F. Bittencourt and E. R. M. Madeira, 'Hcoc: A cost optimization algorithm for workflow scheduling in hybrid clouds,' *Journal of Internet Services and Applications*, vol. 2, no. 3, pp. 207–227, 2011 (cited on p. 102).

[128] H. Casanova, A. Legrand, D. Zagorodnov and F. Berman, 'Heuristics for scheduling parameter sweep applications in grid environments,' in *Heterogeneous Computing Workshop, 2000. (HCW 2000) Proceedings. 9th*, 2000, pp. 349–363. DOI: `10.1109/HCW.2000.843757` (cited on p. 102).

[129] P. Hellinckx, 'De evolutie naar desktop grids, problemen en oplossingen,' Ph.D. dissertation, Universiteit Antwerpen, 2008 (cited on pp. 102, 104, 105).

[130] J. Hicklin, C. Moler, P. Webb, R. F. Boisvert, B. Miller, R. Pozo and K. Remington, 'Jama: A java matrix package,' *URL: http://math. nist. gov/javanumerics/jama*, 2000 (cited on p. 104).

[131] G. B. Wright, 'Radial basis function interpolation: Numerical and analytical developments,' Ph.D. dissertation, University of Colorado, 2003. [Online]. Available: `http://amath.colorado.edu/faculty/fornberg/Docs/GradyWrightThesis.pdf` (cited on p. 104).

[132] S. Verboven, P. Hellinckx, F. Arickx and J. Broeckhove, 'Runtime prediction based grid scheduling of parameter sweep jobs,' M.S. thesis, Department of Mathematics and Computer Sciences, University of Antwerp, 2008 (cited on pp. 104, 105).

[133] O. Project. (). 'Opennebula, the open source toolkit for cloud computing,' [Online]. Available: `http://opennebula.org/` (cited on p. 106).

[134] C. Duez, 'Evaluation of opennebula a private cloud platform,' M.S. thesis, University of Antwerp, 2014 (cited on p. 106).

[135] N. Leavitt, 'Will nosql databases live up to their promise,' *Computer*, vol. 43, no. 2, 2010 (cited on p. 109).

[136] C. Leonello, 'Jqplot charts and graphs for jquery,' *Internet*, 2012 (cited on p. 110).