A Comprehensive Exploration of Multi-Level Simulation in Hardware and Software

Johannes Gaus

Institute of Information Security and Dependability (KASTEL)
Advisor: M.Sc. Sebastian Weber

This work provides a systematic overview of the concept of multi-level simulation and its applications. We begin by explaining the fundamentals of simulation, illustrating how simulations aim to capture real-world complexities. Going beyond the basics, we look at different definitions of a model. We discuss how to handle different abstraction levels and how to connect them.

The major portion involves introducing several methods for conducting multi-level simulations. We present different techniques, such as using agent-based simulations or machine learning concepts. Multi-level simulations find applications in various fields, from social sciences to engineering, showing their versatility and relevance. We discuss different approaches from different areas and break down different patterns, explaining how they help resolve complexities. Real-world examples are used to illustrate the effectiveness of these design patterns in overcoming obstacles in the simulation processes.

In conclusion, this work not only provides a comprehensive guide to understanding multi-level simulation methodologies and their wide-ranging applications across diverse fields but also addresses a notable gap in the literature by offering a much-needed overview in a field where such reviews are scarce.

1 Introduction

Simulation is an important tool for testing and developing different systems. As technology advances, the need for simulation methods becomes crucial to test every aspect of a proposed change [11]. Simulation involves analyzing real-world systems or environments and then generating useful models which we can run in a simulation [1]. It is widely used across different industries to understand and enhance various systems. There exist many different classifications and approaches on how to build a simulation. One category of simulations is discrete, stochastic process-oriented simulations, which model events occurring at distinct time points. Another category is Monte Carlo-type simulations, commonly used in activities like sampling studies, financial analyses, and risk assessments. Equation-based numerical simulations, including models relying on equations such as differential equation solvers, provide a different perspective. Additionally, there are many more approaches like Agent-Based Simulations and various others, explored in more detail in Chapter 4. [5]

Our focus in this work specifically centers on multi-level simulations. Multi-level simulation allows us to capture the details of complex systems at different abstraction levels. In contrast to a normal simulation, we can use multiple levels of detail in one overall simulation when using multiple levels of abstraction. [3]

Simulation primarily involves the creation of software-based models to replicate system behaviors. While hardware approaches like hardware-in-the-loop (HIL) exist, the predominant method involves coding various elements to model and test system behavior within software environments. In software, simulations are created by writing code to replicate system behaviors. On the hardware side, some physical components are built, but the majority of simulation work is carried out in software, where various elements are coded to model and test the behavior of the entire system. [4]

In Chapter 2, we start by introducing the basics of simulation and how simulation helps test and improve different systems. Chapter 3 further elaborates on multi-level simulation. We look at different levels of detail in systems and how they connect. In Chapter 4, we talk about the different ways to approach multi-level simulation. We introduce different methods and techniques used to simulate at different levels and how to connect these different levels. Finally, in Chapter 5, we introduce ongoing challenges in the field, and in Chapter 6, we give a summary and an outlook.

2 Fundamentals of Multi-Level Simulation

2.1 Introduction to simulations

Overall Simulation is a useful tool if fully understood and used correctly [11]. In essence, a simulation can be conceptualized as a model, and a model serves as a representation of a fraction of reality. An isolated model does not have a temporal dimension; temporal considerations come into play only when the model undergoes simulation. If we only look at a model without the simulation part we can only see static properties. As soon as we run a simulation with the model we get dynamic properties. In this dynamic setup, time

becomes crucial for fully understanding how systems behave. [4]
We can define a simulation as a model or representation of a real-world system with all its limits and boundaries. [1] And when considering multi-level simulations we just add different abstration level and different models which all have to work together.

We can take a look from a software engineering perspective we can define three basic principles when talking about abstraction.[12] The first principle is about dealing with complexity. Abstraction is a key concept in engineering and related areas. Our world is complicated, so we create models or simplified versions to focus on what is most important at a given time. This allows us to ignore less important aspects. The second concept suggests differentiating between two parts. The first part is the parts of a system that is specific to what we need. The second part involves consistent patterns across all versions. The third principle states that the differentiation we talked about in the second idea is not just useful, it is the only one we should show to the people using our system. They do not need to know how it is built, just how to use it. When talking about abstraction, it boils down to how well an idea includes what is important without adding unnecessary parts or information.[12]

In addition to these aspects, we can define two main parts where abstraction improves the overall handling: Portability and Reusability. [12] Abstraction, by its nature, allows for the separation of concerns and the encapsulation of implementation details. This separation simplifies complex systems by breaking them down into manageable components, making them easier to understand, modify, and reuse in different contexts. For portability, abstraction enables the creation of modular designs that are not tightly coupled to specific hardware or software environments. By abstracting away platform-specific details, systems can be more easily adapted to different platforms without extensive modifications. This flexibility facilitates the movement of systems across diverse computing environments, promoting interoperability and reducing dependencies on specific technologies.

Similarly, abstraction enhances reusability by promoting the development of modular and composable components. Abstracting common functionalities into reusable modules allows to leverage existing design patterns, saving time and effort. Moreover, by decoupling components through abstraction, changes made to one part of a system are less likely to ripple through the entire system, minimizing the risk of unintended side effects.

We can also look at the general model theory according to Stachowiak. Stachowiak defines three main features a model has to fulfill: the Representation Feature, the Simplification Feature, and the Pragmatic Feature. The first one states that models are always representations of something and representations of natural or artificial originals, which themselves can be models once again. The second one describes that models capture only those attributes of the originals they represent that seem relevant to the model creators. The last feature states that models are not inherently uniquely assigned to their originals. They fulfill their substitution function for certain model-using subjects within a specific time interval and come with restrictions on certain mental or actual operations. [16] [21]
If all these features are present, we can identify it as a model. In the case of a simulation, we can have multiple models. These can represent different levels of abstraction or overall different sections of a bigger system.

The definition of various abstraction levels depends on the specific use case and the intended application of the simulation. As discussed earlier, Stachowiak's model theory identifies three key features. The pragmatic feature emphasizes that a model serves its substitution function for certain subjects within a defined time interval and with some limitations. Consequently, our model is tailored to fulfill a specific function based on the use case, where each abstraction level may represent a distinct aspect of the original system or highlight different aspects. Abstraction levels function as hierarchical layers that conceal details. Considering the representation feature we can ensure that each abstraction level consistently represents a part of the real system. Moreover, we can leverage the simplification feature to represent only the relevant aspects of the system in the model, adjusting the level of detail as needed.

However, it is essential to recognize that altering the abstraction level involves a trade-off between detail and effort. Increasing or decreasing abstraction entails a balance between the desired level of detail and the associated workload.

2.2 Hardware

In hardware, it allows for modeling different levels of circuitry, from high-level architectural designs to low-level transistor behavior. The Register-Transfer level acts as a middle layer, representing digital circuits by focusing on data movement between registers and logical operations. At the gate level, the focus is on arranging and connecting logic gates. The electronic level is where the physical implementation of digital circuits happens using transistors. [23]

Hardware-in-the-Loop (HIL) simulation is a testing and validation technique that involves connecting physical hardware components to a simulation environment. In this setup, the actual hardware interacts with a simulated environment to assess its performance under various conditions. The hardware component refers to the physical equipment or system being tested and can function as a separate abstraction level. We cannot achieve more detail than incorporating the real world into our simulation. It could be part of a larger system, such as an electronic control unit (ECU) in a vehicle, a sensor, or any other hardware component. The term "loop" signifies the closed-loop system where the hardware interacts with the simulated environment. The simulation generates inputs that the hardware reacts to, and the hardware's responses are fed back into the simulation. This loop allows for assessing the hardware's behavior in a controlled and repeatable manner. [14]

In this structure, we can use different levels of abstraction for various parts of the simulation. For example, in automotive applications, a car's electronic control unit could be connected to a simulation mimicking different driving conditions. The ECU responds to the simulated inputs as if it were operating in a real vehicle, enabling engineers to test and validate its performance in a virtual environment before actual deployment [14].

2.3 Software

Software abstraction levels function as a hierarchy that simplifies the different layers of computer programs. As discussed earlier in 2, most simulations are built as software tools. In software, we can distinguish between various abstraction levels, ranging from low levels, such as machine code or low-level programming languages, to high levels, encompassing high-level programming languages and different architectural designs of software. Moving into the implementation phase, we can think about high-level programming languages that provide a user-friendly interface for developers. These languages can be utilized to implement higher-level strategies, abstracting away the details of machine code and offering a more intuitive way to express algorithms and logic. The assembly level provides a bridge between high-level languages and machine code. It allows for more direct control over different actions, making it a crucial layer for optimizing the development of models. At the lowest level of software abstraction, machine code communicates directly with the hardware. [23]

2.4 How to work with abstraction levels

The lower the abstraction level, with the lowest level representing the real-world system, the greater the resources needed to achieve a result. The trade-offs between accuracy and speed can vary based on the specific characteristics and requirements of the build model and the system being simulated in a simulation. [27] The choice of the amount of detail in our model mostly involves a lot of trial and error and strongly depends on the overall goal we want to achieve. We do not exactly know what changes lead to what result. And coming back to the Pragmatic Feature from Stachowiak our model always has a specific task or goal which we want to reach. However, reaching that goal is not always simple and obvious. [3]

Benjamin et al. [3] describes four different problems when working with models on different levels of abstraction. First of all, when thinking about the level of abstraction and designing a new multi-level simulation, a good first step is to select the minimal information needed to achieve our modeling goal. If we choose too little, we have too few information, and if we choose too much, our simulation is unnecessarily complex. The other three problems primarily focus on existing multi-level simulations and their potential rework and modification.

The second problem is called decomposition and involves the process of taking a model at a certain level of abstraction and breaking it down into a set of modeling artifacts or concepts that provide more detailed information about the model. The key idea is that when you decompose a model, you end up with a new model that contains a greater amount of information than the original one. This process is applicable to various types of model artifacts and concepts, such as modeling goals or performance metrics.

Furthermore, we have aggregation. It involves taking a set of modeling artifacts or concepts at a certain level of abstraction and creating a set of "higher-level" modeling artifacts or concepts. The goal is that these should be more useful for decision-making. In other words, it is a process of summarizing or condensing information from a lower level of abstraction to a higher level. When we make an aggregation, the resulting aggregated model artifacts contain a smaller quantity of information compared to the lower-level artifacts. These aggregated artifacts often serve as a summary of the information present in the more detailed, lower-level abstraction.

At last we have integration. Here we try to make necessary adjustments to ensure that two or more different models or model fragments can effectively work together. This challenge becomes particularly crucial when attempting to reuse and integrate existing models, especially legacy models, into a new simulation model.

3 Coupling Abstraction Levels

In multi-level simulations we have different abstraction levels. We can choose the appropriate level of detail based on the simulation goals. Depending on the detail and abstraction level of the individual simulations, these different instances have to communicate with each other. [3]

Models with a more detailed approach, also called high-resolution models typically depict objects in greater detail, while low-resolution models, models with a less detailed approach, may involve grouping several of these objects into a single entity. For instance, in a lower level, a fuel tank might be represented as a single object, whereas an higher level model would likely break down the fuel tank into its individual components. It is essential to note that although the higher model offers more accuracy regarding the fuel tank's performance and properties, the lower-level model becomes more practical for overall decision-making. [10] When we have these different levels of abstraction we need them to communicate in order to build a big simulation. In the following the coupling of these different levels is discussed and different approaches presented.

We can differentiate between offline coupling, model integration, and co-simulation. [24] The paradigm of offline model coupling describes the independent execution of models, with subsequent exchange of results following each simulation iteration. Model integration, on the other hand, involves the combination of diverse models within a unified software environment. Within this framework, models interchange data at specified time intervals or designated events during their execution. In the co-simulation context, models with specialized simulation environments establish connections with one another. Diverging from an integrated approach, the synchronization of data exchange becomes imperative, given potential disparities in the simulation times of individual models. [24]

In scenarios where interactions exclusively involve a pair of models in co-simulation, direct coupling is achievable through a predefined interface, with one model assuming responsibility for synchronization. When synchronization is neccesary for the data exchange among multiple models, the implementation can be realized through the adoption of a coordinating middleware. This middleware organises the orchestration of data exchange across the diverse models involved. [24]

We can also further separate different approaches into serial vs. parallel coupling and loose vs. strong coupling. [22] In a serial coupling approach, simulation levels or components are executed one after the other, in a sequential manner. Each level completes its simulation independently before passing its results to the next level. Contrastingly, in a parallel coupling approach, multiple simulation levels or components run concurrently. This enables simultaneous execution of different parts of the simulation, potentially speeding up the overall process.

In a loose coupling scenario, simulation levels operate somewhat independently. Each level maintains its own time-stepping mechanism, and interactions between levels occur at predetermined synchronization points or events. This approach allows for flexibility and is often more scalable. Strong coupling involves tighter integration between simulation levels. In this approach, interactions between levels are more continuous, with data exchanges occurring at each time step. While providing a more detailed representation of interactions, strong coupling may require more computational resources.

Overall, coupling different abstraction levels in various environments and software systems is a big challenge. Developing large software systems takes a long time, and user needs and technology specifications often change. The paper from Ravichandar and Arthur [17] suggests building systems based on "Capabilities" that can easily accommodate functional changes without causing disruptions. Capabilities are functional abstractions designed to have high cohesion, low coupling, and balanced abstraction levels, creating a framework that can tolerate changes. They use two algorithms, Synthesis and Decomposition, to measure these properties and measure metrics like cohesion, coupling, and abstraction level to objectively evaluate system properties. This could also be a possible useful approach for the coupling of multi-level simulation.

When thinking about coupling existing models in different abstraction levels Graul et al. [10] presents four steps to identify an abstraction level mismatch between different models. Firstly, the Abstraction Parameter is crucial to identify, serving as a quantifiable measure. It is essential to verify the comparability of the abstraction parameter concerning different meassurements like the mean across different levels of abstraction. Secondly, Object Definition is a potential source of inconsistency between models depicting the same system at various levels of resolution.

Three main problems of this inconsistency require explicit analysis and resolution before engaging in any communication activity between the models. They defined Omission, Aggregation and Substitution. Omission refers to differences in object specifications, such as objects in the detailed model not defined in the high-level model. Aggregation involves representing a group of objects in the detailed model as a single aggregated object in the high-level model. Substitution occurs when an object in the detailed model may exist in the high-level model but is known by a different name or is a generalization. Lastly, the Variable Unit is crucial for models to be interchangeable. If the same variables are used in models at different resolutions, they should be defined in the same units. For instance, detailed models may use a finer unit of time, like hours or minutes, instead of days or weeks. Ensuring consistency in variable units allows for the interchangeability of models, with attention to details like time units.

3.1 Direct coupling and feedback loops

In a mixed level environment, it becomes essential for models operating at different abtraction levels to communicate effectively. Achieving this requires consistent methods for the flow of information between varying abtraction levels. For instance, while a simulation model may run at a highly detailed level, a higher-level planning simulation might prefer a summarized view of some data. Developing methods and mapping schemes to facilitate both aggregation and disaggregation of information becomes mandatory. These mapping methods are crucial for enabling interactions between models at different abstraction levels. Additionally, they prove valuable in presenting simulation results to diverse users at different levels of detail based on their specific requirements. [10]

When two abstraction levels communicate, there is often a need for both combining and breaking down information. Different approaches should enable effective communication across various levels of abstraction. Additionally, it may be necessary at times to construct or execute a simulation model at a specific level of abstraction while displaying the results at a higher level. This situation can arise when users occupy different levels within an hierarchy. Instead of running simulation models at multiple levels, it becomes cost-effective to operate them at the lowest abstraction level, consolidate or aggregate the information, and then present the aggregated data at higher levels. [10]

We can establish both qualitative and quantitative mechanisms for data aggregation. In quantitative roll-ups, the simulation model is designed/executed at a lower level of abstraction, and heuristics are formulated to aggregate and present information at higher levels of abstraction. These data aggregation heuristics may be based on averages. [10]

Direct coupling and feedback loops are integral concepts in the coupling of different levels. It involves the immediate interconnection of various components or subsystems within a system, without intermediary elements. The effectiveness of these simulations relies on capturing the detailed relationships between interconnected elements, ensuring that the simulation mirrors its real-world counterpart. If we have different abstraction levels with different time steps, we have to synchronize these so that communication and the exchange of information still work. [6]

3.2 Model-Driven Engineering approaches

Model-Driven Engineering (MDE) is an approach in system development, placing emphasis on the pervasive use of models throughout the development and simulation processes. The paradigm emphasizes the abstraction of system components into models, providing a visual representation that involves analysis, design, and simulation. [28]

Model-Driven Multi-Level Simulation extends the principles of MDE to the area of simulation. This involves the integration of models across different abstraction levels, allowing for the creation of comprehensive system views. A Model Driven approach has several advantages. It offers an organized depiction of the different elements in simulation experiments through meta models. When using a standardized version of these meta models you can automatically generate experiment specifications in a preferred language. This approach enhances the efficiency and accuracy of complex simulation experiments. The automatic transformation of code between different specification languages, facilitated by the use of meta models, promotes the reusability of simulation experiments. [28]

As an example we can consider a scenario in developing a smart city transportation system simulation. At a high level, a model is created to abstract the entire transportation network, incorporating major roads, intersections, and overall traffic patterns. Simultaneously, detailed models are crafted for individual vehicles, capturing specific behaviors such as acceleration, deceleration, and responses to traffic signals. The high-level transportation network model integrates seamlessly with the detailed vehicle models, allowing for a comprehensive simulation that considers both overall traffic patterns and specific interactions between individual vehicles. The Model-Driven approach in this context offers several advantages. Meta models are defined to organize the depiction of different simulation elements, providing a standardized representation. Utilizing a standardized version of these meta models facilitates the automatic generation of experiment specifications. For example, one could specify a simulation scenario where traffic increases during rush hours, and the system would automatically generate the corresponding simulation code in the preferred language. Furthermore, the automatic transformation of code between different specification languages, facilitated by the use of meta models, promotes the reusability of simulation experiments. In this way, if a similar transportation system simulation is required for another city, the high-level transportation network model can be reused, and modifications can be made by plugging in the specifics of the new city.

3.3 Muli-level optimization

When dealing with complex applications, it is important to create simulations to gain more insights. One effective way to do this is by clearly distinguishing between different levels of abstraction. [7] When we already have multiple levels in a multi-level simulation, we can try to optimize for the best level to reach our goal.

To achieve multiple levels of detail, it is crucial to thoroughly understand and know each system. Once this understanding is in place, we can then consider the various levels of abstraction and the specific details. Widely applied in fields such as computer science, engineering, and physics, multi-level simulation plays a crucial role in studying complex systems. [7] This is useful for finding and fixing performance problems, making designs work better, and figuring out how a system will behave in different situations.

The paper "Overview of Methods for Multi-Level and/or Multi-Disciplinary Optimization" of De Wit and Van Keulen [7] comes up with 4 steps to define and elaborate a multi-level simulation and how to optimize the abstraction level. Furthermore Delbrügger et al. [8] also commented on these steps. In the first phase, it is important to identify how the elements within the system interact with each other.

In the second step the optimization of individual solutions can be synchronized to arrive at a comprehensive solution for the entire system. Then the optimization problem is categorized into separable, inseparable, and inseparable yet coupled decisions. Separable optimization decisions can be tackled by the distinct components of the decomposed system since they remain uninfluenced by other elements. Inseparable decisions are connected and impact each other, requiring a step-by-step approach to resolve. In the case of inseparable and coupled problems, where the outcomes of one element impact others and vice versa, solutions must be derived either iteratively or through concurrent optimization. As a last step the selection of the solution sequence and methodology can be determined based on the preceding steps. The authors concluded that the outlined methodology for optimizing a multi-level problem is most effective when dealing with weak coupling of models and a limited amount of data exchange.

4 Various Approaches and Patterns

In the following chapter we want to discuss and present differet approaches for multi-level simulations from various fields and different strategies. When constructing and employing multi-level simulations, there are two primary approaches: one involves running distinct simulations of varying abstraction levels independently, as exemplified in this paper by D'Angelo, Ferretti, and Ghini [6], where the more detailed simulation can be triggered when needed. Alternatively, simulations can be executed simultaneously, as demonstrated by Gai, Somenzi, and Ulrich [9]. In simultaneous operation, the simulation algorithm remains independent of element evaluation procedures and logic values but becomes more complex.

4.1 Different approaches

We can use an approach called Cross-Resolution Modeling (CRM) [18]. The paper of Reynolds, Natrajan, and Srinivasan describes a technique where we can define different Multi-Resolution Entities as a basis. One frequently used strategy is to adjust the level of detail in a simulation, either increasing it from a Low Resolution Entity (LRE) to a High Resolution Entity (HRE) or decreasing it from an HRE to an LRE, so that it aligns with the resolution of other entities encountered in the simulation. This adjustment process is termed "aggregation" when moving from HRE to LRE and "disaggregation" when moving from LRE to HRE. Reynolds, Natrajan, and Srinivasan try to solve this problem but could only narrow it down. The main unresolved concern still revolves around keeping consistency intact when dealing with multiple levels of detail within the same simulation.

Agent-Based Modeling is an approach where we simulate the interactions of individual agents within a system. This methodology captures emergent behaviors and interactions across various levels of abstraction, providing a nuanced understanding of system dynamics. Unlike traditional modeling approaches that rely on aggregate system-level variables, ABM enables the representation of discrete entities with unique attributes and behaviors, allowing for a more granular examination of complex systems. One key benefit of multi-agent simulation is its ability to use separate entities to represent every important concept identified by the user. This approach excels in modeling complex systems where entities interact based on defined rules and behaviors, showcasing its ability to demonstrate difficult system dynamics. These agents can also operate at various levels of abstraction within the system, with higher-level agents representing aggregate entities or subsystems, and lower-level agents representing individual components or entities within those subsystems. Higher-level agents can oversee and coordinate the actions of lower-level agents, while lower-level agents contribute to the overall behavior and dynamics of the system. This hierarchical structure enables the simulation to capture interactions and dependencies between entities at different levels of abstraction. [25]

Hybrid Simulation is another approach combining different simulation techniques such as discrete-event and continuous simulation. The central concept involves integrating two or more diverse simulation paradigms into one big simulation. Specifically in multi-level simulation we can use different approaches to build different levels and then couple them. They support various formalisms, and they are structured in levels that cover multiple areas within various systems. Different approaches exist depending on the available data or the relevance to the different aspects of the modeling process. [2] Although the resulting system or simulation may become more complex in terms of components, relationships, types of connections, and the techniques needed to integrate them, the attractiveness lies in the potential for a representation that is closer to reality. [15]

Machine learning and simulation share the common goal of predicting system behavior through data analysis and mathematical modeling. Simulation, rooted in natural sciences and engineering, recreates causal relationships, for example in computational fluid dynamics. The paper from Von Rueden et al. [26] presents two types of simulations. Simulation-Assisted Machine Learning and Machine-Learning Assisted Simulation. The first approach uses simulations to generate training data for the machine learning model. The second approach uses machine learning surrogate models when the original precise model is too resource-intensive. These models, developed using machine-learning techniques with real-world or high-fidelity simulation data, offer insights into system behavior. Established methods like model order reduction and data assimilation benefit from machine learning for calibration and state estimation. Additionally, machine learning aids in detecting unknown patterns in simulation data.

When we narrow it down from overall simulations to multi-level simulations we can see some approaches from Wittek and Rausch [29] where they handle the complexity inherent in engineering cyber-physical systems. The authors introduce the concept of multi-level simulation, which involves obtaining a holistic perspective on a more general level linked with multiple detailed models of smaller system sections. The paper specifically dives into the progress made in using machine learning techniques for regression to learn state mappings between different simulation levels.

Another approach for building multi-level simulations is the Parallel and Distributed Simulation (PADS) paradigm. PADS operates by simultaneously running simulations on multiple execution units, which are commonly distributed across different machines or processing nodes. The primary objective of PADS is to minimize the overall execution time, by harnessing the collective power of distributed computational resources. By distributing the computational workload across multiple processors or machines, PADS enables simulations to be executed in parallel, thereby accelerating the simulation process. However, it's essential to acknowledge that the adoption of PADS can also introduce certain complexities to the simulation environment. Managing the coordination and synchronization of parallel simulation instances, ensuring data consistency across distributed systems, and addressing potential communication overheads are some of the challenges associated with PADS implementation. Despite this challenge, the benefits of PADS, including enhanced scalability, improved performance, and efficient resource utilization, make it a valuable approach in multi-level modeling scenarios. [19]

4.2 Multi-Level Modeling Challenges in Human Mobility and Social Sciences

We chose to further explore the challenges of human mobility and behavior and how to make a multi-level simulation in this specialized domain. Given the wealth of literature and a personal interest in this topic, we found the work of Serena et al. [19] particularly compelling and relevant. It also presents some insights into different areas where multi-level modeling is used. In this particular field multi-level modeling mainly involves representing a system at various levels of detail within a single model. This approach enables the coexistence of different modeling paradigms, offering flexibility in choosing the most suitable paradigm for each level. The term "multi-level" in this context goes beyond the conventional layering of simulations and refers to the integration of multiple sub-models within the same simulation. The models may adopt hierarchical, flat, or hybrid structures, and the coupling of these different levels must also be considered as it presents a significant challenge. The paper's focus on human mobility and behavior underscores the broad range of applications. Given the inconsistency in concepts and terminology surrounding multi-level modeling in the literature, this paper aims to provide a systematic overview by analyzing and classifying relevant scientific literature.

The authors emphasizing deterministic and stochastic model approaches and presenting various simulation paradigms, including:

1. Discrete-Event Simulation (DES)
2. Agent-Based Model (ABM)
3. Cellular Automata (CA)
4. Continuous Simulation
5. System Dynamics (SD)
6. Monte-Carlo Methods

Discrete-Event Simulation (DES) operates with state changes occurring at discrete points in simulated time. DES can employ a time-stepped approach, where fixed time steps represent units of time, or an event-driven approach, where timestamped events unfold chronologically. As mentioned before Agent-Based Models (ABM) present systems as collections of autonomous, interacting entities, each defined by specific features and behaviors. The interconnection of agents with the simulated environment, often referred to as situated agents, emphasizes their local relationship with the environment. Cellular Automata (CA) position agents within a group of cells, updating their state based on neighbors' current states. CA, with continuous or discrete state space, serve investigations into complex systems, emphasizing interactions among agents. Continuous Simulation describes systems using sets of differential equations, providing a precise simulation of timings and ensuring time consistency with other models. System Dynamics (SD), an incarnation of continuous simulation, deploys differential equations and feedback loops to model interactions. Monte-Carlo Methods compute outcome probabilities through iterative simulations involving stochastic decisions for variables with inherent uncertainty.

The same paper [19] also discusses different approaches in the area of Epidemic modeling. Particularly in the context of the recent COVID-19 outbreak. Epidemic models are designed to predict the progression of outbreaks over time and assess the effectiveness of containment measures such as vaccination campaigns and lockdowns. Given that infectious diseases spread through proximity or direct contact with infected individuals, human mobility plays a crucial role in epidemic diffusion. The paper emphasizes various scales, including local, regional, and global levels. These levels serve as a good example for a multi-level simulation. We can look at different levels of detail depending on the specific use case. If we go back to Stachowiak and the pragmatic feature we can clearly distinguish different levels of abstraction and different goals. For instance, if we aim to simulate the epidemic development in one room versus the global development.

In the paper a common framework is discussed that couples within-host and between-host models. Within-host models are bound to pathogen-host interactions and immune responses, while between-host models focus on pathogen diffusion between individuals, utilizing within-host simulations to determine transmissibility and recovery rate. Both within-host and between-host models rely on differential equations as their foundation. Within-host models, as mentioned before, designed to represent individual infection dynamics, commonly consider parameters such as the count of infected and healthy cells, death rate, shedding rate, and transmission rate. Tailoring within-host models to specific diseases is crucial. On the other hand, between-host models adopt a compartmental approach, categorizing the population into mutually exclusive epidemiological statuses. Such models, often employed with sufficiently large populations, approximate epidemic trends, neglecting transmission dynamics related to specific individuals or groups. And again we have a good example for different abstraction levels. The paper also discusses the Susceptible-Infected-Recovered model (SIR), which is a prevalent choice for between-host analyses, categorizing individuals as susceptible, infected, or recovered, with transition rates dependent on factors like pathogen diffusion rate and mobility patterns.

Variants of the SIR model, such as the Susceptible-Exposed-Infected-Recovered (SEIR) model and the Susceptible-Infected-Susceptible (SIS) model, offer nuanced representations by considering exposure, reinfection, and hospitalization. The integration of social factors, human mobility, and epidemic dynamics forms a common theme in multi-level modeling approaches for epidemic simulations. Frequently, between-host compartmental models like SIR and SEIR are coupled with mobility models to represent both local virus spread and global pathogen diffusion. Human mobility is part of the model through diverse structures, including graph-based approaches, agent-based models and equations-based methods. And again as mentioned multiple times before trade-offs between accuracy and computational efficiency are addressed in various models. Adaptive multi-resolution frameworks and multiscale approaches aim to balance the benefits of detailed agent-based models with the computational efficiency of equations-based simulations. Some models explore the connection between epidemic spread and social factors, awareness, and opinions.

In summary, the reviewed literature of this review showcases diverse approaches to multilevel epidemic modeling, differing in terms of layer semantics and modeling paradigms. Compartmental models based on Ordinary Differential Equations (ODEs) are common for computational efficiency, while agent based models (ABMs) are preferred for accuracy. Despite challenges, ABMs remain valuable for 'what-if' analyses and inference in situations where detailed information is limited.

We can also take a look at the use of multi-level simulations in social sciences [19]. A prevalent approach in social sciences involves the use of multilayer graphs for modeling social relationships. This entails nodes representing individuals, edges denoting connections between them, and layers representing different environments where these connections occur. This methodology proves effective for investigating the diffusion of trends, news, and ideas across multiple social networks, as individuals often belong to various interconnected layers. Another aspect covered in the paper is the representation of online and real-life connections in different network layers. Information propagation is detailed through a compartmental model, where individuals may be ignorant, spreaders, variations, oysters, or in recovery stages. The paper also delves into the field of population dynamics in the social sciences, addressing issues such as disease predisposition, homophily, migration, and gentrification. Gentrification, the influx of middle-class or wealthy individuals into popular neighborhoods, is particularly explored. Accurate modeling of such scenarios requires reliable data analyzed at both individual and aggregate levels.

Overall Serena et al. [19] makes a distinction between Multilayer vs. Multilevel vs. Multiscale Modeling:

- Multilayer models are suitable for different types of relationships
- Multilevel models integrate multiple existing models in a single simulator
- Multiscale models find a trade-off between accuracy and efficiency

4.3 Different patterns for multi-level simulations

In the paper [20] by Serena et al. they present the benefits of multi-level Modeling and Simulation. It highlights the advantages, such as efficient resource utilization and faster development, while acknowledging challenges in complex development processes. The authors propose a set of design patterns to address issues related to interoperability, execution orchestration, and state variable updates in multi-level models. The paper draws inspiration from general software engineering literature and tailors certain patterns specifically for multi-level applications.

The paper mainly differnetiates between 3 approaches. Equation-Based Models (EBMs) which utilize continuous-space, continuous-time frameworks based on differential equations, making them efficient for describing aggregate parameters over large populations. Agent-Based Models (ABMs) operate in continuous-space, discrete-time, employing autonomous, interacting computational objects for systems with crucial agent interactions. And Cellular Automata which are characterized by discrete-space and time-stepped models, employing a grid of cells with simple rules for state updates, often computationally efficient through parallel computations. Each approach tackles specific modeling needs based on system characteristics and computational considerations. In the following part we want to introduce some of these pattern.

Overall the paper introduces five classes of patterns:

1. Orchestration Patterns

   - Model Controller Pattern

   - Director-Worker Pattern

   - Concurrent Modularity Pattern

2. Structural Patterns

   - The Composite Pattern

   - Bridge Pattern

   - Adapter Pattern

3. Execution Policy Patterns
   - Sequential Execution Pattern
   - Patallel Execution Pattern
4. Information Exchange Patterns
   - Return Value Pattern
   - Pipe through Temporary Files Pattern
   - Shared Memory Pattern
5. Multiscale patterns
   - Aggregation/Disaggregation Pattern
   - Adaptive Resolution pattern

The first patterns manage the execution flow of sub-models, addressing execution sequencing. Structural patterns detail the aggregation of sub-components into complex models, drawing from general Object-Oriented programming principles. Execution policy patterns determine the mapping between components and execution units. Information exchange patterns specify data transfer between models of various types, such as continuous and discrete-space models. At last Multiscale patterns facilitate the integration of models with varying levels of detail, providing a comprehensive framework for multi-level modeling.

In the Orchestration patterns we manage the execution flow of diverse components, which may include different types of models or models with distinct parameters. These patterns dictate how control is transferred from one component to another. The Models' Controller pattern involves an external entity overseeing sub-model execution, scheduling, state management, and information exchange. It centralizes scheduling logic but may become complex. The Director-Worker pattern employs a hierarchical structure, where control passes from a director to a worker. These patterns can coexist, enhancing flexibility. The Concurrent Modularity pattern allows components to interact without a strict hierarchical structure, promoting peer-to-peer interaction. Realizations include Director on Hold, where the director suspends until workers finish, and Worker on Demand, which pre-allocates a worker pool to avoid dynamic creation overhead. The Concurrent Modularity pattern facilitates concurrent execution of semantically different models, addressing challenges like time granularity and synchronization through methods like time translation, checkpointing, or rollback mechanisms.

The structural patterns focus on composing software elements for enhanced flexibility and maintainability. These patterns originated in the field of software engineering. The Composite pattern facilitates hierarchical object composition, treating atomic and composite objects uniformly. The Bridge pattern separates abstraction from implementation, featuring Abstraction (high-level interface), Implementor (technical functionalities), and Concrete Implementor (concrete implementation). Applied to agent based modeling, it can separate agent definitions from behavior code. The Adapter pattern enables collaboration between components with incompatible interfaces, using Client (object using the Adapter), Adaptee (object needing adaptation), Target (object Client wants to use), and Adapter (intermediary translating interfaces). In multi-level modeling, the Adapter pattern is crucial when integrating sub-models initially not intended to work together. In a multiscale traffic model, it can adapt micro and macro zones, allowing seamless collaboration.

The Execution policies patterns dictate the rules and guidelines for mapping the model onto the underlying execution units. The selection of an execution policy is uncertain because of several factors, such as the model's size, complexity, available computational resources, and the desired level of performance. In the sequential execution pattern, the model operates on a single execution unit, typically a solitary processor or processor core. This approach is commonly employed when models inherently follow a sequential structure or when existing implementations are designed in a sequential manner. Sequential execution is deemed suitable when the time required for model analysis is not a limiting factor. On the other hand, parallel execution involves the division of the model into smaller components that can run concurrently on multiple execution units, potentially spanning interconnected machines. Traditionally applied to enhance the performance of monolithic models, parallel execution is valuable for concurrently processing events in discrete-event simulations or utilizing parallel solvers for large sets of differential equations. In the context of multi-level simulation, parallelization proves beneficial when employing domain partitioning. This technique divides the simulation space into distinct partitions, each modeled at different granularity levels or using diverse types of models.

Next we have the Information Exchange patterns. They play a crucial role in defining how data is communicated between sub-models, with factors such as information type (discrete vs. continuous) and component relationships (hierarchical vs. peer-to-peer) influencing the transfer mechanisms. The Return Value pattern, within a Director-Worker scenario, involves the Director calling the Worker, which subsequently returns a result. While straightforward, challenges arise when the Director operates concurrently with Workers. The futures pattern, inspired by concurrent programming, can address this issue by having Workers return an object representing a "promise" to compute a result, with the Director blocking until the result is ready. An alternative approach is the Pipe through Temporary Files pattern, where information exchange occurs through temporary files. This method offers flexibility, versatility, and allows developers to choose suitable data representations. However, it introduces overhead due to file operations, necessitates user responsibility for file cleanup, and requires additional mechanisms for notifying consumers of new data. In the Shared Memory pattern, data is stored in a memory region accessible by all sub-models, offering more efficiency than temporary files but requiring emulation on distributed-memory architectures. Careful consideration is needed to avoid read-write conflicts in two-way communication scenarios. Rounding Strategies address the challenge of exchanging data between sub-models with continuous and discrete state representations in multi-level modeling. For instance, in a multi-level epidemic model combining Ordinary Differential Equations-based predictions with more detailed agent based modeling, rounding strategies are vital to ensure global consistency properties, such as conservation of model-specific entities. Various strategies exist, including tracking rounding errors and adjusting the agent based model's state variables when necessary to maintain overall consistency.

At last we have the Multiscale patterns. They tackle the challenges associated with representing a sub-model at various levels of detail, aiming to find an optimal balance between execution time and result precision. A common strategy involves Spatial Aggregation-

Disaggregation, where a microscopic level provides detailed entity behavior, and a macroscopic level deals with aggregate metrics. These levels may use different models or the same model with varying parameters. The Aggregation/Disaggregation design pattern guides the transition between levels, involving the consolidation of micro-level entities into a macro-level entity (aggregation) or the creation of micro-level entities to represent a macro-level entity (disaggregation). This pattern is commonly applied in agent based models and extends to other model types. Various manifestations of this pattern include the Zoom pattern, where micro entities are eliminated during the transition into macro zones, the Puppeteer pattern, where micro entities are frozen and controlled by macro agents, the View pattern, where micro entities' state mirrors macro entities' state, and the Cohabitation pattern, featuring bidirectional interactions between micro and macro entities. The Adaptive Resolution pattern is relevant in multiscale frameworks where a sub-model's level of detail is determined by spatial or time resolution. Conditions triggering a change in resolution are model-specific. For example, a multiscale infection propagation model may initiate with an agent-based approach for detailed dynamics and switch to an equation-based methodology for computational efficiency after reaching a specific threshold of infected individuals. The presented patterns offer solutions to some challenges in the domain of multi-level simulations, yet numerous open challenges persist, demanding further exploration and resolution.

4.4 Different patterns for multi-level agent-based simulations

Now after we discussed some more broader approaches we want to showcase different approaches and design patterns specifically for agent-based multi-level simulations presented in [13] by Mathieu, Morvan, and Picault. The paper outlines four design patterns focused on simplifying the modeling and implementation of multi-level agent-based simulations. These simulations are designed to handle entities that belong to different but interconnected abstractions or organizational levels. The proposed patterns are derived from typical situations found in the literature. For each pattern, the paper presents use cases, associated data structures, and algorithms.

We introduce the following four patterns:

1. Zoom Pattern
2. Puppeteer Pattern
3. View Agents Pattern
4. Cohabitation Pattern

The Zoom pattern characterizes scenarios in Multi-Level Agent-Based Simulations involving destructive aggregation or disaggregation. This pattern entails creating agents at the target level while simultaneously destroying the original agents in the source level. In the case of aggregation, micro agents "dissolve" into the macro agent, either abruptly or gradually. For disaggregation, the macro agent is eventually destroyed after creating micro agents, whether progressively or in a single step. For example in traffic simulations with three observation levels: "micro" (vehicles), "meso" (groups of vehicles), and "macro" (vehicle flow) this can be used. Efficiently simulating large, heterogeneous networks often requires coupling micro (agent-based) and macro (equation-based) models. The coupling, whether for aggregation or disaggregation, is generally destructive. During aggregation, micro agents representing vehicles dissolve into a macro agent, modifying its internal state. Conversely, dynamic hybrid simulations allow the disaggregation of a macro area into micro agents (vehicles), resulting in the destruction of the macro agent.

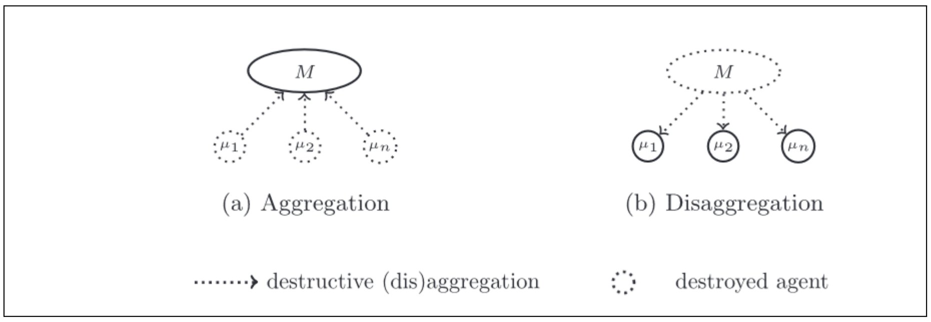

Figure 1: The Zoom Pattern [13]

The next pattern is called the Puppeteer pattern. This is characterized by a scenario where agents from the source level are not destroyed but instead "frozen." In this pattern, these source-level agents, termed "puppets," lose the capability to independently execute their decided entry point, rendering them inactive. Despite this, their individual state continues to evolve, and the update entry point can still be executed by the scheduler. The distinctive feature is that the behavior of these "puppets" is delegated to the agents at the target level. The Puppeteer pattern is used for example in the field of biology. For modeling the impact of PAI-1 molecules on tumor metastasis where fighter agents aggregate into squad agents, showcasing the delegation of behavior while retaining the evolving state of the source-level agents. The Puppeteer pattern provides a mean to handle complex interactions and aggregations across different abstraction levels.

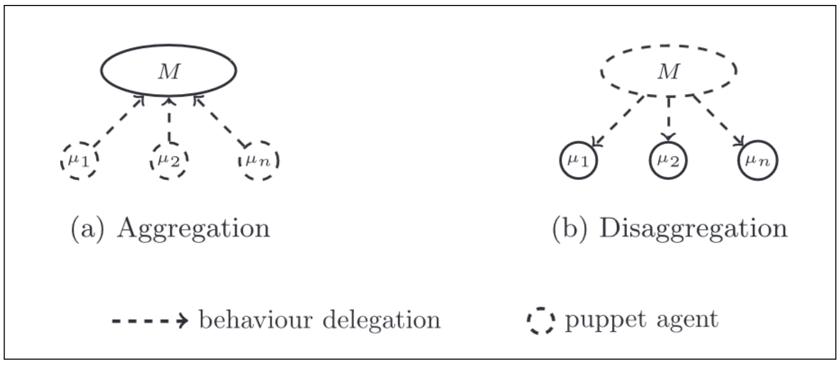

Figure 2: The Puppeteer Pattern [13]

The next pattern, The View Agents pattern introduces "View" agents in multi-agent simulations, aiming to concretely represent agents from the source level on the target level. These "View" agents consistently mirror the state of the source-level agents. Their update entry point involves composing or decomposing the states of the source-level agents. While equipped to perform decide actions, this is limited on their actions not influencing the represented agents, often involving data processing for user display. Information exchange follows a unidirectional path, moving from visualized agents (source level) to "View" agents (target level). This pattern is characterized by a form of state delegation. In multi-agent simulations, effective observation tools are crucial for visualizing emergent behaviour. The View Agents pattern addresses this by introducing "View" agents to represent agents involved in collective appearances. For instance, in fluid flow simulations, a "View" agent can autonomously embody and represent emergent appearances, offering feedback capacity. Similarly, interactive traffic flow simulations enable users to magnify a macroscopic area, creating a corresponding microscopic representation of interacting vehicles through "View" agents. The View Agents pattern enriches the representation of appearances in simulations, introducing an additional level of abstraction.

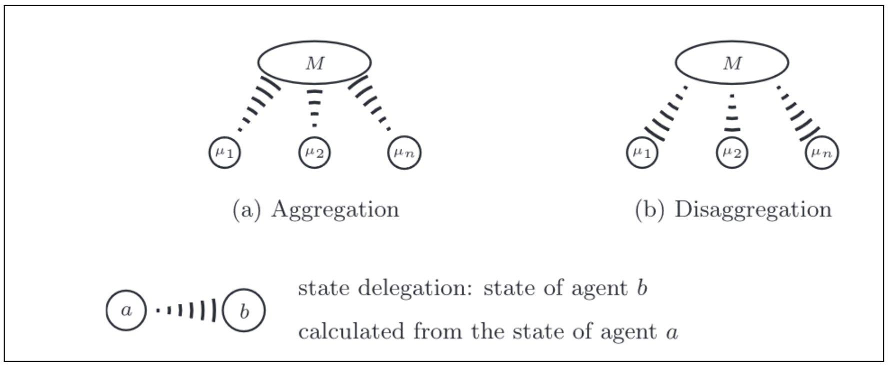
M
μ1
μ2
μn
(a) Aggregation
(b) Disaggregation
a
b
state delegation: state of agent b
calculated from the state of agent a

Figure 3: The View Agent Pattern [13]

The last presented pattern is called the Cohabitation pattern. Here, agents from both the source and target levels actively engage in update and decide actions. Notably complex, the interactions between micro and macro levels are bidirectional, mutually influencing each other. This pattern finds prominent application in simulations of intricate systems, particularly in the realm of biological systems, such as the multi-level modeling of tumor development. Similarly, in hybrid traffic flow simulations, Cohabitation involves bidirectional information exchange. Micro agents are generated based on macro agent characteristics and micro area conditions, impacting both levels. For instance, congestion affects the initial velocity of generated vehicles, showcasing the intricate interplay between micro and macro levels in complex systems.

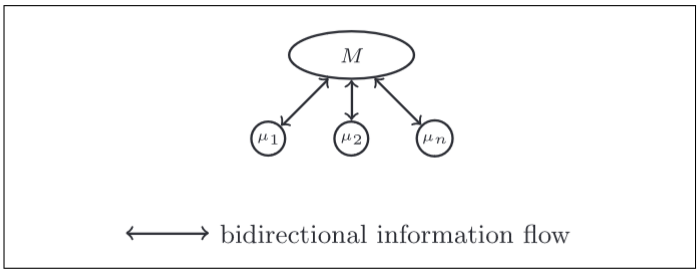

Figure 4: The Cohabitation Pattern [13]

Overall these patterns present some approaches for building multi-level agent based simulations using an agent-based approach.[13]

5 Discussion of challenges and open questions

Overall, the computational intensity of simulations poses challenges in terms of resource demands and simulation duration. This includes not only the need for powerful hardware but also considerations of time efficiency. The use of different levels of abstraction for a simulation, as described in Section 3, can help manage the resource problem. If we do not need as much information, we can simply change the abstraction level and go higher. However, with advancing developments in various fields of simulation, not only is the need for more computational resources more present, but also the need for other approaches to gain the necessary computational power. Researchers may need to explore parallel computing, optimization algorithms, or hardware advancements to mitigate these challenges.

To ensure seamless integration across various abstraction levels, addressing data consistency, model synchronization, and the prevention of errors during the integration process, strategies such as advanced data management techniques, robust synchronization protocols, and automated error-checking mechanisms may be crucial in overcoming these challenges. As described in Section 3, we already have some coupling approaches to ensure everything works. However, with larger simulations and an increasing number of coupling approaches, the overall strategies to achieve error-free communication become more challenging. Validating and verifying models in multi-level simulation is complicated due to the complexity of representing real-world systems accurately. Research efforts might focus on developing advanced validation techniques, possibly incorporating real-time data feedback and comparing simulation outcomes with empirical data.

The absence of standardized methodologies poses challenges in terms of consistency and comparability. Many papers present different approaches for multi-level simulations (see chapter 4), each utilizing distinct naming conventions and explanations. Given the wide-ranging nature of the field, spanning multiple diverse areas, researchers may need to engage in collaborative efforts to establish common frameworks or interoperability standards, fostering a more unified multi-level simulation community. Currently, the landscape of multi-level simulation lacks comprehensive reviews and overviews, indicating a gap in structured assessments of this area. It is essential to address this for a more systematic exploration of multi-level simulation methodologies. There is a need for in-depth analyses and summaries that can provide a structured understanding of the diverse approaches within multi-level simulation. A good example is [19]. Encouraging a shift towards more organized reviews and overviews is crucial. By advocating for more thorough assessments and comprehensive summaries, we can enhance the accessibility and usability of multi-level simulation methodologies. Collaborations may involve establishing common platforms and creating frameworks for sharing simulation models and data.

6 Conclusion

As mentioned in the introduction, multi-level simulation allows us to capture the details of complex systems at different abstraction levels. In this work, we provide an introduction to the fundamentals of simulations, clarifying what a simulation is and explaining the differences between a simulation and a multi-level simulation.

After introducing the concept of different abstraction levels and models highlighting the challenge of coupling different abstraction levels, we present various approaches to handling abstraction levels and overcoming associated challenges. We discuss distinctions between abstraction levels in hardware and software, addressing their utilization in simulations. Furthermore, we explore working with different abstraction levels and modifying existing simulations. Following this, we introduce various methods to couple different levels, starting with the simpler direct coupling and progressing to approaches like model-driven engineering. We also provide insights into optimizing abstraction levels in simulations and selecting the appropriate level of detail.

The most extensive section covers diverse approaches to multi-level simulation. We provide a focused overview, emphasizing specific key points without aiming to encompass all available information on the subject. We discuss different strategies, such as various Machine Learning approaches or Agent-Based Simulation. Additionally, we offer insights into traffic simulations, epidemic modeling simulations, and social science simulations, illustrating how these specific areas utilize multi-level simulation. Following that, we present a set of different design patterns to address some challenges in working with multi-level simulation. Moreover, we introduce four design patterns specifically tailored for agent-based simulations. In the discussion chapter, we explore open topics and challenges, summarizing unresolved questions. This work aims to provide a comprehensive overview of various approaches to simulations, particularly focusing on multi-level simulations. Our goal is to present an overall view of how multi-level simulations are constructed across various domains, showcasing their diverse applications in different fields.

I declare that I have developed and written the enclosed thesis completely by myself, and have not used sources or means without declaration in the text.

Karlsruhe, 01.03.2024

. . . .

(Johannes Gaus)

References

[1] J. Banks. “Introduction to simulation”. In: *2000 Winter Simulation Conference Proceedings (Cat. No.00CH37165)*. Vol. 1. Orlando, FL, USA: IEEE, 2000, pp. 9–16. ISBN: 9780780365797. DOI: 10.1109/WSC.2000.899690. URL: http://ieeexplore.ieee.org/document/899690/ (visited on 12/21/2023).

[2] R. Bardini et al. “Multi-level and hybrid modelling approaches for systems biology”. en. In: *Computational and Structural Biotechnology Journal* 15 (2017), pp. 396–402. ISSN: 20010370. DOI: 10.1016/j.csbj.2017.07.005. URL: https://linkinghub.elsevier.com/retrieve/pii/S2001037017300314 (visited on 12/29/2023).

[3] P. Benjamin et al. “Simulation modeling at multiple levels of abstraction”. In: *1998 Winter Simulation Conference. Proceedings (Cat. No.98CH36274)*. Vol. 1. Washington, DC, USA: IEEE, 1998, pp. 391–398. ISBN: 9780780351332. DOI: 10.1109/WSC.1998.745013. URL: http://ieeexplore.ieee.org/document/745013/ (visited on 11/12/2023).

[4] Hans-Joachim Bungartz et al. *Modeling and Simulation: An Application-Oriented Introduction*. en. Springer Undergraduate Texts in Mathematics and Technology. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014. ISBN: 9783642395239 9783642395246. DOI: 10.1007/978-3-642-39524-6. URL: https://link.springer.com/10.1007/978-3-642-39524-6 (visited on 12/21/2023).

[5] J.S. Carson. “Introduction to modeling and simulation”. In: *Proceedings of the 2004 Winter Simulation Conference, 2004.* Vol. 1. Dec. 2004, p. 16. DOI: 10.1109/WSC.2004.1371297. URL: https://ieeexplore.ieee.org/document/1371297 (visited on 01/06/2024).

[6] Gabriele D’Angelo, Stefano Ferretti, and Vittorio Ghini. “Multi-level simulation of Internet of Things on smart territories”. en. In: *Simulation Modelling Practice and Theory* 73 (Apr. 2017), pp. 3–21. ISSN: 1569190X. DOI: 10.1016/j.simpat.2016.10.008. URL: https://linkinghub.elsevier.com/retrieve/pii/S1569190X16302507 (visited on 11/12/2023).

[7] Albert De Wit and Fred Van Keulen. “Overview of Methods for Multi-Level and/or Multi-Disciplinary Optimization”. en. In: *51st AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference&lt;BR&gt; 18th AIAA/ASME/AHS Adaptive Structures Conference&lt;BR&gt; 12th.* Orlando, Florida: American Institute of Aeronautics and Astronautics, Apr. 2010. ISBN: 9781600869617. DOI: 10.2514/6.2010-2914. URL: http://arc.aiaa.org/doi/abs/10.2514/6.2010-2914 (visited on 12/09/2023).

[8] Tim Delbrügger et al. “Multi-level simulation concept for multidisciplinary analysis and optimization of production systems”. en. In: *The International Journal of Advanced Manufacturing Technology* 103.9-12 (Aug. 2019), pp. 3993–4012. ISSN: 0268-3768, 1433-3015. DOI: 10.1007/s00170-019-03722-1. URL: http://link.springer.com/10.1007/s00170-019-03722-1 (visited on 11/12/2023).

[9] S. Gai, F. Somenzi, and E. Ulrich. “Advances in Concurrent Multilevel Simulation”. en. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 6.6 (Nov. 1987), pp. 1006–1012. ISSN: 0278-0070. DOI: 10.1109/TCAD.1987.1270341. URL: http://ieeexplore.ieee.org/document/1270341/ (visited on 01/07/2024).

[10] Michael Graul et al. “A framework for modeling and simulation at multiple levels of abstraction”. In: ed. by Dawn A. Trevisani. Orlando (Kissimmee), FL, May 2006, p. 622706. DOI: 10.1117/12.668496. URL: http://proceedings.spiedigitallibrary.org/proceeding.aspx?doi=10.1117/12.668496 (visited on 01/07/2024).

[11] Ricki G. Ingalls. “Introduction to simulation”. In: *Proceedings of the 2011 Winter Simulation Conference (WSC)*. ISSN: 1558-4305. Dec. 2011, pp. 1374–1388. DOI: 10.1109/WSC.2011.6147858. URL: https://ieeexplore.ieee.org/abstract/document/6147858 (visited on 01/06/2024).

[12] G. Kiczales. “Towards a new model of abstraction in software engineering”. In: *Proceedings 1991 International Workshop on Object Orientation in Operating Systems*. Oct. 1991, pp. 127–128. DOI: 10.1109/IWOOOS.1991.183036. URL: https://ieeexplore.ieee.org/document/183036 (visited on 01/07/2024).

[13] Philippe Mathieu, Gildas Morvan, and Sebastien Picault. “Multi-level agent-based simulations: Four design patterns”. en. In: *Simulation Modelling Practice and Theory* 83 (Apr. 2018), pp. 51–64. ISSN: 1569190X. DOI: 10.1016/j.simpat.2017.12.015. URL: https://linkinghub.elsevier.com/retrieve/pii/S1569190X17301946 (visited on 11/15/2023).

[14] Aras Mirfendreski et al. “Finding Coupling Strategies of a Real-Time Capable Fourier-Transformation-Based Engine Model on a HIL-Simulator”. en. In: *Simulation and Testing for Vehicle Technology*. Ed. by Clemens Gühmann, Jens Riese, and Klaus Von Rüden. Cham: Springer International Publishing, 2016, pp. 43–65. ISBN: 9783319323442 9783319323459. DOI: 10.1007/978-3-319-32345-9_5. URL: http://link.springer.com/10.1007/978-3-319-32345-9_5 (visited on 01/06/2024).

[15] Navonil Mustafee et al. “Hybrid simulation studies and Hybrid Simulation systems: Definitions, challenges, and benefits”. In: *2015 Winter Simulation Conference (WSC)*. Huntington Beach, CA, USA: IEEE, Dec. 2015, pp. 1678–1692. ISBN: 9781467397438. DOI: 10.1109/WSC.2015.7408287. URL: http://ieeexplore.ieee.org/document/7408287/ (visited on 01/07/2024).

[16] Karlis Podnieks. “Philosophy of Modeling: Neglected Pages of History”. In: *Baltic Journal of Modern Computing* 6.3 (2018). ISSN: 22558942, 22558950. DOI: 10.22364/bjmc.2018.6.3.05. URL: http://www.bjmc.lu.lv/fileadmin/user_upload/lu_portal/projekti/bjmc/Contents/6_3_05_Podnieks1.pdf (visited on 02/27/2024).

[17] Ramya Ravichandar and James D. Arthur. “Cohesion, Coupling and Abstraction Level: Criteria for Capability Identification”. In: *Journal of Software* 3.1 (Jan. 2008), pp. 1–8. ISSN: 1796-217X. DOI: 10.4304/jsw.3.1.1-8. URL: http://www.academypublisher.com/ojs/index.php/jsw/article/view/1916 (visited on 01/07/2024).

[18] Paul F. Reynolds, Anand Natrajan, and Sudhir Srinivasan. “Consistency maintenance in multiresolution simulation”. en. In: *ACM Transactions on Modeling and Computer Simulation* 7.3 (July 1997), pp. 368–392. ISSN: 1049-3301, 1558-1195. DOI: 10.1145/259207.259235. URL: https://dl.acm.org/doi/10.1145/259207.259235 (visited on 12/29/2023).

[19] Luca Serena et al. “A review of multilevel modeling and simulation for human mobility and behavior”. en. In: *Simulation Modelling Practice and Theory* 127 (Sept. 2023), p. 102780. ISSN: 1569190X. DOI: 10.1016/j.simpat.2023.102780. URL: https://linkinghub.elsevier.com/retrieve/pii/S1569190X23000576 (visited on 01/07/2024).

[20] Luca Serena et al. “Design Patterns for Multilevel Modeling and Simulation”. In: *2023 IEEE/ACM 27th International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*. Singapore, Singapore: IEEE, Oct. 2023, pp. 48–55. ISBN: 9798350337846. DOI: 10.1109/DS-RT58998.2023.00015. URL: https://ieeexplore.ieee.org/document/10305763/ (visited on 01/07/2024).

[21] Herbert Stachowiak. *Allgemeine Modelltheorie*. German. Wien New York: Springer, 1973. ISBN: 9783211811061.

[22] Trevor Sweafford and Hwan-Sik Yoon. “Co-simulation of dynamic systems in parallel and serial model configurations”. en. In: *Journal of Mechanical Science and Technology* 27.12 (Dec. 2013), pp. 3579–3587. ISSN: 1738-494X, 1976-3824. DOI: 10.1007/s12206-013-0909-x. URL: http://link.springer.com/10.1007/s12206-013-0909-x (visited on 11/15/2023).

[23] Andrew S. Tanenbaum and Todd Austin. *Structured computer organization*. 6th ed. Boston: Pearson, 2013. ISBN: 9780132916523.

[24] Sebastian Thiede et al. “Multi-level simulation in manufacturing companies: The water-energy nexus case”. en. In: *Journal of Cleaner Production* 139 (Dec. 2016), pp. 1118–1127. ISSN: 09596526. DOI: 10.1016/j.jclepro.2016.08.144. URL: https://linkinghub.elsevier.com/retrieve/pii/S0959652616313087 (visited on 11/12/2023).

[25] Thomas Huraux, Nicolas Sabouret, and Yvon Haradji. “A Multi-level Model for Multi-agent based Simulation.” In: 2014. URL: https://www.researchgate.net/publication/317185201_A_Multi-level_Model_for_Multi-agent_based_Simulation.

[26] Laura Von Rueden et al. “Combining Machine Learning and Simulation to a Hybrid Modelling Approach: Current and Future Directions”. en. In: *Advances in Intelligent Data Analysis XVIII*. Ed. by Michael R. Berthold, Ad Feelders, and Georg Krempl. Vol. 12080. Cham: Springer International Publishing, 2020, pp. 548–560. ISBN: 9783030445836 9783030445843. DOI: 10.1007/978-3-030-44584-3_43. URL: http://link.springer.com/10.1007/978-3-030-44584-3_43 (visited on 01/07/2024).

[27] Steven Walczak and Paul Fishwick. "A centralized methodology for multi-level abstraction in simulation". en. In: *ACM SIGSIM Simulation Digest* 19.4 (Dec. 1988), pp. 25–31. ISSN: 0163-6103. DOI: 10.1145/65774.65775. URL: https://dl.acm.org/doi/10.1145/65774.65775 (visited on 11/12/2023).

[28] Pia Wilsdorf et al. "A Model-Driven Approach for Conducting Simulation Experiments". en. In: *Applied Sciences* 12.16 (Aug. 2022), p. 7977. ISSN: 2076-3417. DOI: 10.3390/app12167977. URL: https://www.mdpi.com/2076-3417/12/16/7977 (visited on 12/29/2023).

[29] Stefan Wittek and Andreas Rausch. "Learning State Mappings in Multi-Level-Simulation". In: *Simulation Science*. Ed. by Marcus Baum et al. Vol. 889. Cham: Springer International Publishing, 2018, pp. 208–218. DOI: 10.1007/978-3-319-96271-9_13. URL: http://link.springer.com/10.1007/978-3-319-96271-9_13 (visited on 01/07/2024).