Challenges for Automation in Adaptive Abstraction

Romain Franceschini*†, Moharram Challenger†‡, Antonio Cicchetti§, Joachim Denil†‡, Hans Vangheluwe†‡
*University of Corsica, France; †University of Antwerp, Belgium; ‡Flanders Make, Belgium; §Mälardalen University, Sweden
r.franceschini@univ-corse.fr, moharram.challenger@uantwerpen.be, antonio.cicchetti@mdh.se,
{joachim.denil, hans.vangheluwe}@uantwerpen.be

***Abstract*—Models are well-defined abstractions that provide cost-effective representations of the real-world for a precise purpose. When dealing with complex problems, there usually exist multiple abstractions, typically describing partially overlapping details of the system under study, and resulting in a hierarchy of abstractions. Adaptive abstraction leverages these levels with the aim of dynamically adapting the abstractions used during system execution. In this paper, we describe such process in terms of a MAPE-K (Monitor-Analyze-Plan-Execute over a shared Knowledge) control loop to discuss the challenges towards adaptive abstraction automation. In particular, we elaborate on adaptively selecting a candidate over multiple abstractions, an unaddressed issue in the literature. The discussion is supported by a running example in an agent-based simulation scenario.**

***Index Terms*—Adaptive System, Multi-Abstraction, Multi Paradigm, System Modelling, Agent based Simulation, Traffic Simulation.**

I. Introduction

Models are abstractions that capture specific properties of a system under study. They represent the system in a reduced fashion for a specific purpose [1]. The amount of information in a model determines its abstraction level: the less information it contains, the more abstract it is [2]. Usually, systems with complex dynamics are abstracted in many ways with respect to a set of properties. As such, multiple abstractions are eventually defined, describing partially overlapping properties of the system under study. We intuitively tend to organize these into a hierarchy of models, from the most detailed to the coarser ones.

Before the formulation of a model, an important task is to determine the most appropriate level of abstraction [3]. Together with the properties of interest, the viewpoint on the system is what drives our choices regarding a particular abstraction. The knowledge over the system, the dimensions considered (e.g. space, time, ...), the scales at play, the paradigm, the modeling view as well as the formalism chosen to describe the model shapes the resulting abstraction level.

When reasoning about system properties, humans intuitively switch back and forth between different levels of abstraction depending on the problem to solve. With the idea that such a mental process can be automated, adaptive abstraction takes advantage of multi-level models to dynamically adapt the type of abstractions used during execution or simulation. Among its benefits, this technique improves insight and explainability through the detection of particular phenomena [4] as well as simulation performance [5] by reducing computational needs when coarser models can be used. With respect to a set of properties of interest that should be preserved at all times, adaptive abstraction leverages detected phenomena to automatically switch to a more appropriate abstraction level.

This paper provides a framework to reason on the adaptive abstraction of simulation models. We explicitly reason over the involved properties of models together with an operationalisation using the MAPE-K (Monitor-Analyze-Plan-Execute over a shared Knowledge) framework. Since the early 2000s, a growing interest in adaptive abstraction has been observed in the literature [6], particularly with the agent-based modeling (ABM) paradigm which often requires high computational needs. To the best of our knowledge, the contributions that have already been proposed were developed in an ad-hoc way, providing tailored solutions to the specific problems of each application. Due to the complexity of the technique, a generic framework towards automated support of adaptive abstraction in terms of a MAPE-K control loop is needed to generalize the application of adaptive abstraction.

This paper is organized as follows: In the next section, we provide background about the notion of abstraction as well as the challenges to support adaptive abstraction. Section III presents our motivating example based on a traffic network case study, which we use to illustrate adaptive abstraction as well as our approach in the remainder of the paper. We then develop the MAPE-K approach to present how each step fits the adaptive abstraction requirements in Section IV and conclude the paper in Section V.

II. Background and related works

This section gives the required background information related to abstraction and adaptive abstraction and describes related works in both areas.

A. Abstraction

A foundation of the notion of abstraction has been proposed during a workshop on Multi-Paradigm Modeling [7]. The authors establish the notion of abstraction relative to the information contained in a model M, which represents the questions that can be asked regarding the model: the set of properties $P = I(M)$. Those properties serve to define relations between abstractions. Thus, a relation between two models M_1 and M_2 can have different characters, namely abstraction, refinement, or equivalence with respect to a set of properties:

- Equivalence: $\forall p \in P$: $M_1 \models p \iff M_2 \models p$
- Abstraction: M_1 is an abstraction of M_2 w.r.t P if $\forall p \in P$: $M_1 \models p \implies M_2 \models p$.
- Refinement: M_1 is a refinement of M_2 iff M_2 is an abstraction of M_1.

From a language engineering perspective, the relation between two models w.r.t properties can also be seen as a model transformation. Among transformation intents classified in [8], three are related to abstraction, namely the *refinement*, the *abstraction* and the *approximation* intents. The latter specifies a model m_1 approximates m_2 when m_1 is equivalent to m_2 up to a certain error margin, m_1 preserving more properties than m_2 as the error decreases.

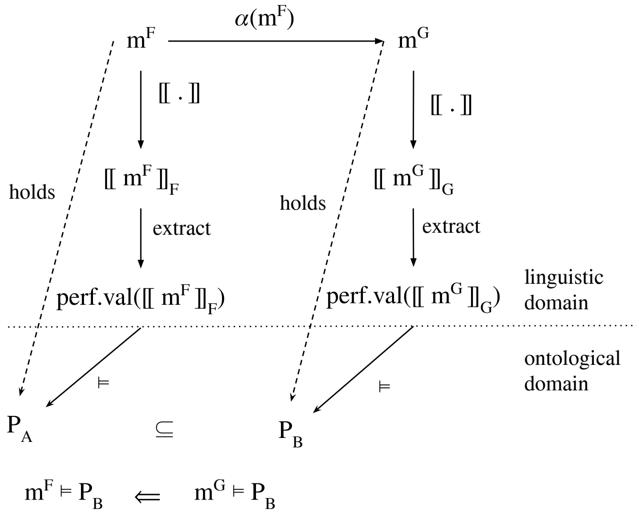

Fig. 1: Abstraction relation between two models w.r.t. properties.

Figure 1 illustrate the process of checking a property, and is inspired by the work of Barroca et al. [9] on relating ontological and linguistic domains. Checking whether a model satisfies a property may require to go over the semantic domain ($[\![.]\!]$) [10]. Let m^F and m^G define two models described in formalism F and G, respectively. m^G is an abstraction of m^F which results from the abstraction function α, as defined in [11]. Such relation can be defined with respect to the set of properties each model satisfies, namely P_A and P_B, with $P_A \subseteq P_B$. A property is verified by extracting a performance value from the semantic domain of a model. This performance value is then evaluated to produce a boolean value that determines whether the associated model satisfies the property. Depending on the tolerance over the performance value during the evaluation, an abstraction may satisfy the same property as the refined model. The abstraction is then an approximation relative to its refinement.

Formulating an abstraction often involves aggregating information across one or several dimensions, either by relaxing them (e.g. lumping, coarser scale) or by removing them altogether (e.g. remove the spatial dimension). Conversely, refinement is about disaggregation and information re-construction. Many dimensions can be involved [3], namely the *temporal* dimension, the *spatial* dimension, the *functional* dimension and orthogonally, all other dimensions found in the states of the models. As those dimensions evolve from one abstraction to another, the most appropriate formalism to represent them evolves accordingly.

B. Adaptive abstraction

Given multiple abstractions describing partially overlapping properties of a system under study and a set of properties of interest, the process of adaptive abstraction analyzes the running system to find opportunities to switch from an abstraction to another dynamically. Such opportunities can arise from simple properties like a component reaching a steady state, invariants known in the domain, or from complex situations involving interactions between several entities leading to emergent phenomena, arising from interactions. We call these the *enabling properties*. As switching itself has a cost, enabling properties are usually analyzed over time to maximize the probability for the newly instantiated abstraction to be relevant for a longer period of time. According to the relations established between abstractions and the properties of interest, the adaptive abstraction process has to determine if a switch can actually be performed. Hence, adaptive abstraction involves distinct steps, namely:

1) sample periodically the running system;
2) detect over time the properties allowing a switch;
3) select, if any, the most appropriate abstraction known to preserve the set of properties of interest;
4) perform the abstraction(s) and/or refinement(s).

In the literature, the first two steps are performed through clustering methods [12], [13], usually over the spatial dimension. As for the analysis of emergent properties over time, weighted graphs are the most often used technique [12], [14] to keep track of the recurrence of the phenomena and determine if they are worth switching. Chen et al. [15] provide a framework to observe properties by formally describing events in a hyperspace, allowing them to include any dimensions. Complex behaviors can be described with *complex event types*, which formally relate a sequence of events together with a temporal operator.

To our knowledge, the third step is usually implicit as all approaches found in the literature were developed in an ad-hoc way, involved only two abstractions and had an implicit static set of properties of interest.

Finally, the last step consists in initializing states of the newly instantiated models before performing the required switches. While the abstraction transformation is trivial, the refinement is much more challenging since it requires information to be reconstructed. In any case, such a process is domain-specific and thus, cannot be fully automated without an explicit description. A switch may involve replacing the whole running system, possibly by an abstraction expressed in a different formalism, e.g. from an agent-based model to a system dynamics model. However, if the running system is expressed in a formalism that supports hierarchical constructs, a switch may only occur at the component level, meaning different abstractions -possibly expressed in different formalisms- can interact with each other in an integrated way, eventually sharing other entities. For example, spatial clustering in an agent-based model lies in this category.

For such systems, the question of how to represent aggregates has been studied. Mathieu et al. [16] describe several design patterns generalised from recurring solutions found in the literature, namely the *zoom* pattern, the *puppeteer* pattern and the *view* pattern. Figure 2 illustrates all three. The *view* pattern (Figure 2.c) provides visual insights to the modeller by only highlighting the involved entities when an emergent property is detected. The *zoom* pattern (Figure 2.a) is the most challenging pattern, since its purpose is to completely substitute the original abstraction by another one. Finally, the *puppeteer* pattern (Figure 2.b) also substitutes the original model(s), but preserves the states of lumped entities.

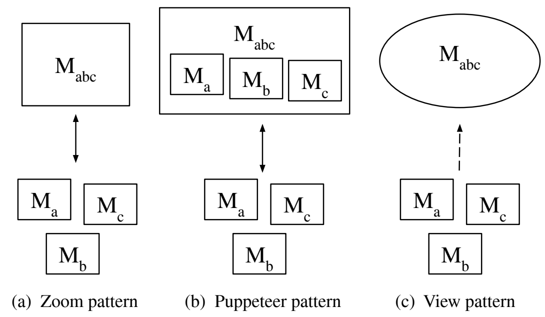

Fig. 2: Design patterns identified for adaptive abstraction [16].

III. Motivating example

We present a road traffic network as a concrete adaptive abstraction case study, which is used throughout the remainder of the paper as a running example. Road traffic systems can be tackled at many levels of details from microscopic to macroscopic and features complex interactions between vehicles, which can lead to emergent behaviors such as traffic jams. As such, it is a good candidate for adaptive abstraction.

We focus on a highway cloverleaf interchange system described as an agent-based model. The environment represents the interchange as a graph, where edges are passive road segments and the vertices are intersections. The road segments are associated with real-world data exported from a graphical information system including segment locations, number of lanes, speed limits, and direction.

Three distinct abstractions are used to represent moving vehicles. Abstractions can be blended: they interact with each other and share the same environment. Figure 3 shows screenshots of the running system, with two zoomed regions illustrating the three abstractions. The most detailed abstraction (M_1) describes individual vehicles as agents featuring a car-following behavior, where vehicles try to maintain their preferred speed, adjust it to avoid collisions and speeding, and look after lane switching opportunities. A more abstract

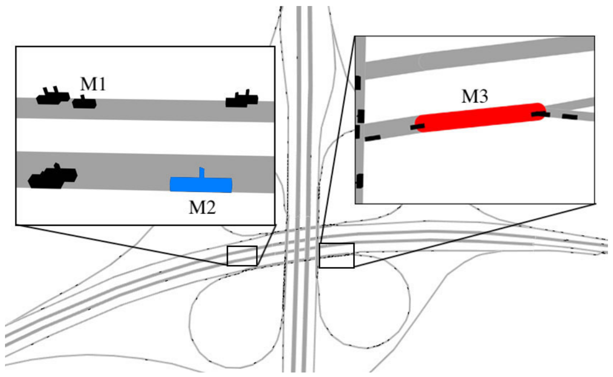

Fig. 3: Illustration of the traffic network system with three distinct abstraction levels.

model captures groups of close vehicles as agents to represent traffic jams (M_2), approximating the behavior of the "flock". The last abstraction (M_3) takes a different approach and approximates certain areas by representing road segments as active entities using a Discrete Event model, similarly as the work of Bosmans et al. [17]. Vehicles become passive entities when entering an active road segment and are released at their expected crossing time based on their speed.

To leverage these type of models, the adaptive abstraction system may aggregate M_1 models to a M_2 model when a traffic jam is detected, until it holds. The switch from a passive road segment to an active one (M_3) can be realized when the segment is dense enough to be considered as a queuing system.

Adaptive abstraction for this example currently has to be specifically tailored since no related works provide a generic solution for this problem. In the next section, we present our MAPE-K based conceptual framework intended to generalize adaptive abstraction, where this motivating example is used to better understand the introduced concepts. The study of the usefulness or the performances improvements of adaptive abstraction was already shown in the literature [5] and is not addressed in this paper.

IV. MAPE-K based Adaptive Abstraction

This section presents a conceptual and methodological framework for reasoning in terms of adaptive abstraction. We argue that a MAPE-K (Monitor-Analyze-Plan-Execute over a shared Knowledge) control loop naturally fits the requirements of adaptive abstraction. As such, it is a promising basis towards the automated support of adaptive abstraction.

Adaptive abstraction can be seen as a self-managed autonomic system with self-optimization and self-configuration capabilities, whose main purpose is to "continually seek opportunities to improve its own performance and efficiency" with "automated configuration of components" following high-level policies [18], i.e., the properties of interest. Figure 4 gives a structural overview of such system, where a managed system is queried and updated through a MAPE-K feedback control loop. The system continuously polls or listens to events relative to the managed system through the Monitor component (2) and

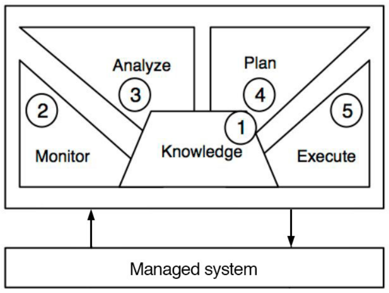

Fig. 4: MAPE-K control loop [18].

updates the shared Knowledge (1). Knowledge is evaluated by the Analyze component (3) to detect opportunities for the managed system to be updated according to the optimization goals. In case there are such opportunities, they are examined by the Plan (4) component to check whether they are compatible and to determine an update plan. Finally, the Execute (5) component updates the managed system according to the plan.

The remainder of this section specifies how the adaptive abstraction process as presented in section II-B relates to each MAPE-K component.

A. Knowledge

While Knowledge is not an active entity of the control loop, it is shared by the MAPE components and provide means to reason about the monitored information with respect to the self-management goals. Given the distinct concerns of adaptive abstraction, we identified several Knowledge entities.

1) Properties of interest: Adaptive abstraction's main goal can be described as providing insights and improving runtime performances while producing similar results with respect to a set of properties. As such, this set should be modeled so that the MAPE components can reason in terms of those properties.

For our road network, the following set can be considered:

- p_1: from any direction, vehicles can reach any direction.
- p_2: vehicles flow in one direction.
- p_3: vehicles have a fixed path to their target destination.

2) Abstractions relations: Abstractions relations which are described in section II-A, suggest that abstractions can be related thanks to the properties they are assumed to satisfy.

For our road network, the relations between the three abstractions M_1, M_2 and M_3 (illustrated in Figure 3) are also represented as boxes in Figure 5 with their respective formalisms in the top-right corner and their set of assumed satisfied properties (P_A, P_B and P_C, respectively) in the bottom-right corner. All abstractions are assumed to satisfy the properties of interest: $\{p_1, p_2, p_3\} \in P_A \cap P_B \cap P_C$.

3) Enabling properties: The *Enabling properties* represent the sequence of events expected to occur for model(s) to trigger a switch (including emergent properties or known invariants). In our example, a property enabling the abstraction or the refinement to another model is associated with each relation in Figure 5. A switch from M_1 to M_2 is triggered

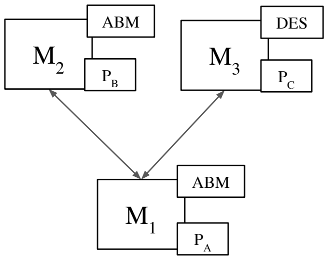

Fig. 5: Relations between traffic network abstractions.

when vehicles are on the same lane and are close enough from each other for a given period. The opposite, i.e. the refinement from M_2 to M_1, is triggered when the traffic jam reaches the next intersection so that p_3 is still satisfied. A switch from M_1 to M_2 is triggered when the number of vehicles on a road segment, i.e. its density, reaches a certain threshold for a given period. Individual vehicles are replaced by internal events within the discrete event model of the road segment, scheduled at the time they are supposed to leave the segment given their speed. M_2 models are refined to M_1 models when the number of cars arriving at the cars segment decreases and thus, reduces the density down to a threshold for a given amount of time.

4) Model knowledge: This represents relevant information about the entities at play in the system, such as their states or their interactions.

5) Abstraction knowledge: This part keeps maintains a list of models susceptible to be abstracted or refined.

B. Monitor

The Monitor component naturally fits with the first step of adaptive abstraction described in section II-B. Data is gathered from the models currently executed or simulated in the managed system and the *model knowledge* base is updated. The actual information that is collected may concern states and/or particular interactions between several models.

As the monitoring process can be costly, several ways to lower its overhead can be considered. The first concern is the sampling rate, allowing either a 1:1 relation between state transitions and sampling or a coarser resolution to be used, allowing some states to be missed. The second one is to make use of the *relations between abstractions* from the Knowledge to only monitor the type of models that can be abstracted or refined. Finally, to lower the amount of information gathered for each type of model, the appropriate dimensions to monitor (e.g. space, time, states) can be deduced from the *enabling properties* description.

In our example, the monitor component uses a weighted graph approach similar to [14] to update the *model knowledge*. The weights are increased when an enabling property is satisfied and decreased otherwise.

C. Analyze

The Analyze component regularly examines the *Model knowledge* to detect adaptation opportunities of the managed system models. Detection is about matching a required events sequence for an enabling property to be satisfied. This consists of evaluating data from the model knowledge. While the Monitor role is to select relevant information, the Analyze component role is to harness it to detect patterns that hold for a given amount of time. Checking enabling properties over time is important to ensure the investigated phenomenon is consistent. For example, vehicles that are close together for a brief period of time are not worth switching to a traffic jam.

Depending on how precise the modeler expresses the enabling properties, the detection can be considered more or less exploratory. If one tries to gather new insights about the system under study, the property can be described approximately. For example, to check whether spatial clusters occur, a closeness property involving extracting spatial locations as performance values (see Figure 1) can be evaluated with an arbitrary threshold. Conversely, if one has accurate knowledge over the system under study, the enabling property can be described by precisely defining complex temporal patterns and complex combinations of states leading to well-known phenomena. Although, accurate knowledge does not necessarily result in a complex sequence of events. The framework proposed by Chen et al. [15] can be considered for such purposes. In any case, once switching opportunities are found, the *Abstraction knowledge* is updated accordingly.

Besides the evaluation of monitored data w.r.t. enabling properties to detect abstraction or refinement opportunities, the Analyze component may also be responsible for regularly checking whether a model in the managed system satisfies the properties of interest. Although this is only necessary if the abstractions are known to have edge cases where the associated properties can be infringed. In this case, properties of interest can be checked on the current execution path. In case one or more properties are not satisfied, the Analyze component can determine another appropriate abstraction to switch to.

For the road network case, clustering algorithms can be applied to the weighted graph to isolate models or groups of models above a certain threshold.

D. Plan

If the *Abstraction knowledge* has been filled during analysis, the Plan component evaluates the different opportunities of abstracting/refining the running models and checks whether they are compatible with the properties of interest and if they are not conflicting with each other. Eventually, an enacting plan is produced and the abstraction knowledge is updated. As far as we know, such an activity has never been addressed in the literature, since the previous works only involve two possible abstractions and an implicit, fixed set of properties of interest. However, it is an important part of the adaptive abstraction to support complex abstraction hierarchies as well as a dynamic set of properties of interest.

With respect to the properties of interest, each abstraction/refinement candidate is evaluated twice: (1) to ensure that an appropriate target abstraction exists based on the abstraction relations found in the Knowledge and (2), to ensure the target abstraction is able to satisfy the properties of interest. If more than two distinct abstractions are available in the abstraction hierarchy, conflicts between potential switches may occur and should be addressed during the planning phase. Even though a simple abstraction hierarchy is featured in our motivational example (see Figure 5, conflicts appear and have to be resolved during the Plan phase. We distinguish two conflicting situations.

1) Overlapping switch conflict: The conflict occurs when overlapping switching opportunities are detected during the analysis phase, e.g. the same running model or group of models can be switched to two different abstractions. In such cases, the Plan component has to determine which abstraction is the most appropriate. As long as the considered abstraction types are assumed to satisfy the properties of interest, the most abstract one could be promoted. However, abstractions' properties only allow establishing partial ordering relations between abstractions. As such, some abstractions cannot be compared with each other and lie at the same "level". In this case, the most abstract type cannot be determined without introducing domain-specific hints.

Such conflict can appear in our road traffic example since the enabling properties describe situations that can occur simultaneously. For example, if both a group of vehicles is detected as a traffic jam while the vehicles are situated on a road segment dense enough to be abstracted to an active road segment, a choice has to be made. In this case, the DES abstraction can be preferred since it should yield better performances.

2) Heterogeneous abstractions conflict: This type of conflict occurs when a group of heterogeneous abstractions is considered for aggregation or disaggregation. Models that belong to different types of abstractions may be conflicting and require a resolving plan. In this regards, two categories can be distinguished.

The first category occurs when different abstractions "levels" are considered to be merged, e.g. individual vehicles (M_1) with traffic jams (M_2). In this case, the new abstraction has to be initialized from the detailed models as well as from the coarser ones for all entities to be merged. However, the coarser abstraction has to support this particular initialization. In our example, traffic jams (M_2) have to support initialization from traffic jams and vehicles. Otherwise, the problem falls into the second category of heterogeneous abstractions conflict.

The second category occurs when there is an opportunity for models to be switched to an abstraction having no direct transformation relation, in which case the two abstraction types are considered at the same "level" and there is no known way to initialize the new abstraction from the existing one. To solve this problem, the running model(s) first have to be refined (possibly recursively) to a more detailed abstraction, from which the involved entities can be re-abstracted to the target abstraction. This can occur in the road traffic example. For example, a traffic jam (M_2) entering an active road segment (M_3) lies in this category since those two abstractions have no relations (see Figure 5). They cannot be "merged" directly, the traffic jam has to be refined to vehicles first so that those can be merged into the active road segment. This is also necessary if traffic jams do not support initialization both from vehicles (M_1) and traffic jams (M_2): a first refinement has to be done before abstracting all vehicles.

E. Execute

The Execute component eventually enacts the plan established during the Plan phase, if any. During this phase, the main challenge is related to the state initialization. Moving from a detailed abstraction to a coarser one requires the faithful capturing of the available information by selecting the appropriate dimensions and by aggregating it. For refinement, information has to be reconstructed from an incomplete information representation. While in some cases information is still available, e.g. if using the *puppeteer* or the *view* patterns, the *zoom* pattern is destructive. This requires domain-specific knowledge to be associated with each abstraction/refinement relation. The previous works explored two approaches: abstracting via statistics or equilibrium states [5].

V. Conclusion and future works

In this paper we looked into the challenges of adaptive abstraction, a technique taking advantage of multi-level models to dynamically switch between abstractions during execution or simulation. Multi-level models provide several abstractions over a system under study, that typically describes partially overlapping details. As most existing works propose tailored solutions and only consider two types of abstractions to switch to, we lay the basis for a generic adaptive abstraction framework based on a MAPE-K (Monitor-Analyze-Plan-Execute over a shared Knowledge) self-management loop. After providing some background relative to abstraction and adaptive abstraction and presenting a road network running example, our contribution describes the challenges for a generic adaptive abstraction framework based on MAPE-K components.

As a perspective, three axes can be investigated. The first one is related to the relations established between abstractions. Since a partial order is defined based on the properties each abstraction is assumed to satisfy, many abstractions are considered at the same level. For greater granularity, relations could also be established by analyzing the scales and precision units used across the model dimensions in addition to the properties, as Iwasaki [3] did to relate abstractions with the temporal dimension. This idea is related to the concept of approximation, which is presented as a transformation intent by Lúcio et al. [8] and should be formally defined as an additional model relation [7]. The second axis is about tackling adaptive abstraction from a language engineering perspective to provide fully automated support. While we developed our MAPE-K based traffic model in an ad-hoc way, the approach could be generalized by providing an adaptive abstraction meta-model as well as an associated transformation language. Finally, the concept of experimental frame could be integrated into the framework. First introduced by B.P. Zeigler, experimental frames provide an explicit specification over models context, which can be used to validate model outputs. As adaptive abstraction is only relevant w.r.t. properties of interest, we believe experimental frames are an appropriate way to detect whether the managed system is still valid after abstraction switches.

Acknowledgement

The content of this paper was discussed at the Computer Automated Multi-Paradigm Modeling 2019 workshop. We also thank Bentley J. Oakes for his insightful comments.

References

[1] H. Stachowiak, *Allgemeine Modelltheorie*. Springer, 1973.

[2] P. Benjamin, M. Erraguntla, D. Delen, and R. Mayer, "Simulation modeling at multiple levels of abstraction," in *Proceedings of the 30th Conference on Winter Simulation*. Washington, DC, USA: IEEE, Dec. 1998, pp. 391–398.

[3] Y. Iwasaki, "Reasoning with multiple abstraction models," in *Recent Advances in Qualitative Physics*. MIT Press, Jan. 1993, pp. 67–82.

[4] P. Caillou and J. Gil-Quijano, "SimAnalyzer: automated description of groups dynamics in agent-based simulations," in *AAMAS '12: Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems*, Valencia, Spain, Jun. 2012, pp. 1353–1354.

[5] A. Sharpanskykh and J. Treur, "Group Abstraction for Large-Scale Agent-Based Social Diffusion Models with Unaffected Agents," in *Agents in Principle, Agents in Practice*. Springer, 2011, pp. 129–142.

[6] G. Morvan, "Multi-level agent-based modeling - A literature survey," *arXiv*, vol. 1205.0561v7, 2013.

[7] H. Giese, T. Levendovszky, and H. Vangheluwe, "Summary of the Workshop on Multi-Paradigm Modeling: Concepts and Tools," in *Models in Software Engineering*. Springer, 2007, pp. 252–262.

[8] L. Lúcio, M. Amrani, J. Dingel, L. Lambers, R. Salay, G. M. K. Selim, E. Syriani, and M. Wimmer, "Model transformation intents and their properties," *Software & Systems Modeling*, vol. 15, no. 3, pp. 647–684, Jul. 2014.

[9] B. Barroca, T. Kühne, and H. Vangheluwe, "Integrating Language and Ontology Engineering," in *Proceedings of Multi-Paradigm Modelling workshop MoDELS 2014*, 2014, pp. 77–86.

[10] M. Amrani, B. Combemale, L. Lúcio, G. Selim, J. Dingel, Y. Le Traon, H. Vangheluwe, and J. R. Cordy, "Formal Verification Techniques for Model Transformations: A Tridimensional Classification." *The Journal of Object Technology*, vol. 14, no. 3, pp. 1–43, Aug. 2015.

[11] T. Kühne, "Matters of (Meta-) Modeling," *Software & Systems Modeling*, vol. 5, no. 4, pp. 369–385, Jul. 2006.

[12] T. Moncion, P. Amar, and G. Hutzler, "Automatic characterization of emergent phenomena in complex systems ," *Journal of Biological Physics and Chemistry*, vol. 10, pp. 16–23, 2010.

[13] J. Gil-Quijano, T. Louail, and G. Hutzler, "From Biological to Urban Cells: Lessons from Three Multilevel Agent-Based Models," in *Principles and Practice of Multi-Agent Systems*. Springer, 2012, pp. 620–635.

[14] A. Sarraf Shirazi, T. Davison, S. von Mammen, J. Denzinger, and C. Jacob, "Adaptive agent abstractions to speed up spatial agent-based simulations," *Simulation Modelling Practice and Theory*, vol. 40, pp. 144–160, Jan. 2014.

[15] C.-C. Chen, C. D. Clack, and S. B. Nagl, "Identifying Multi-Level Emergent Behaviors in Agent-Directed Simulations using Complex Event Type Specifications," *SIMULATION*, vol. 86, no. 1, pp. 41–51, Dec. 2009.

[16] 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, Apr. 2018.

[17] S. Bosmans, S. Mercelis, P. Hellinckx, and J. Denil, "Reducing Computational Cost Of Large-Scale Simulations Using Opportunistic Model Approximation," in *2019 Spring Simulation Conference (SpringSim)*. IEEE, Apr. 2019, pp. 1–12.

[18] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, Jan. 2003.