Going beyond structure - Can Dynamic Multi-Layer Algebra Support Multi-Level Simulation?

Sebastian Weber
sebastian.weber@fzi.de
FZI Research Center for Information Technology
Karlsruhe, Germany

Thomas Weber
thomas.weber@kit.edu
KASTEL
Karlsruhe Institute of Technology
Karlsruhe, Germany

Arne Lange
arne.lange@kit.edu
KASTEL
Karlsruhe Institute of Technology
Karlsruhe, Germany

Jörg Henß
henss@fzi.de
FZI Research Center for Information Technology
Karlsruhe, Germany

Robert Heinrich
robert.heinrich@uni-ulm.de
Institute of Software Engineering and Programming Languages
Ulm University
Ulm, Germany

Abstract

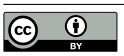
Multi-level simulation (MLS) combines simulation models that differ in abstraction, resolution, timing, or computational cost. This makes MLS attractive for complex systems, but simulation artefacts must be selected, coupled, and exchanged without losing semantic compatibility. At the same time, "level" denotes different things in MLS and in multi-level modeling (MLM), and is already ambiguous within MLS. We therefore first separate the notions of level involved and show that the orderings used in MLS induce no instantiation relation, whereas the instantiation chains that do exist in MLS run orthogonally to them, inside a single simulation stage. Against this background we examine Dynamic Multi-Layer Algebra (DMLA) as a modeling basis for MLS compatibility: not as an encoding of MLS levels, but as a level-blind, refinement-based formalism for representing simulation artefacts, simulator capabilities, compatibility constraints, and validation operations. An ECU software architecture example and a DMLA model excerpt illustrate how validation operations expose missing inputs, semantic mismatches, and unjustified simulator switches. We condense recurring challenges into six modeling needs, identify which of them any MLM approach with a constraint language can address and which benefit specifically from DMLA, and state where further work is required.

CCS Concepts

• **Computing methodologies** → **Model verification and validation**; *Multiscale systems*; • **Software and its engineering** → *Model-driven software engineering*.

Keywords

multi-level modeling, multi-level simulation, Dynamic Multi-Layer Algebra, model validation



MODELS Companion 2026, Málaga, Spain

ACM ISBN 979-8-4007-2903-4/2026/10
https://doi.org/10.1145/3837062.3838698

ACM Reference Format:
Sebastian Weber, Thomas Weber, Arne Lange, Jörg Henß, and Robert Heinrich. 2026. Going beyond structure - Can Dynamic Multi-Layer Algebra Support Multi-Level Simulation?. In *ACM/IEEE 29th International Conference on Model Driven Engineering Languages and Systems (MODELS Companion 2026), October 04–09, 2026, Málaga, Spain.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3837062.3838698

1 Introduction

Developing complex cyber-physical systems increasingly relies on combinations of models and simulations, and early validation and verification depend on whether the available models and analyses suit the questions asked during development [4]. In practice, this is judged through expert knowledge and project-specific assumptions about required inputs, transferable results, and permissible simulator substitutions.

Multi-level simulation (MLS) addresses part of this problem by combining simulations that operate at different resolutions, abstractions, time scales, or costs. An analysis may, for example, switch between timing simulations at different abstraction levels when more accuracy is required, while a fast functional simulation maintains the hardware state across such switches [22]. Beyond having several simulators, MLS requires precise knowledge about model inputs, outputs, state mappings, synchronization assumptions, and validity conditions.

Applying multi-level modeling (MLM) to MLS requires care because "level" denotes different things in the two communities, and is already ambiguous within MLS itself. In MLS, one paper's levels are spatial scales, another's are time granularities, a third's are verification stages of increasing fidelity, and a fourth's are simply cheap versus expensive models of the same system. In MLM, levels denote classification through instantiation, linguistic or ontological typing, or—in DMLA—refinement depth. Table 1 contrasts these notions and states what relates two adjacent levels in each case. Its central observation is that the MLS notions induce no typing relation between the artefacts they separate. Equating them with MLM classification levels therefore makes an MLS architecture look formally structured while hiding the semantic questions: which simulator accepts which artefact, which state representation maps

onto another, and which abstraction suffices for a given goal. Terminology is only one recurring MLS challenge; coupling also hinges on data availability, consistency of state mappings, synchronization, and the growth of compatibility rules (Sections 2 and 3).

We position Dynamic Multi-Layer Algebra (DMLA) as a candidate formalism for modeling and validating the artefacts around MLS, rather than as a way of turning MLS levels into MLM levels. DMLA is relevant because it is level-blind, refinement-based, and expresses validation logic and operations inside the modeling framework itself [2, 14, 20, 21]. We are explicit about the scope of that claim: stating and executing compatibility conditions is possible in other MLM approaches with a constraint or action language, and Section 5 identifies which parts of our argument hold for MLM in general and which are specific to DMLA.

We contribute, first, a review of MLS and MLM work (Section 2) condensed into six modeling needs, N1–N6 (Section 3); second, a separation of the notions of level involved, including where instantiation chains actually occur in MLS (Sections 3 and 5); and third, an ECU example with a DMLA model excerpt that makes simulator compatibility explicit (Section 4). Section 6 concludes with limitations.

2 Related Work

2.1 Multi-Level Simulation

MLS is used when no single model suffices for all analysis concerns, combining coarse, fast models for broad exploration with detailed ones for timing, communication, physical, or statistical accuracy. In software architecture and embedded systems, MLS combines functional simulation with timing-oriented simulation to support extensibility and refinement of analyses [22]. Automotive practice shows a related but distinct use of "level": model-in-the-loop, software-in-the-loop, and virtual-ECU stages replay the same test scenarios while fidelity increases toward production timing behavior [23]. In IoT scenarios, D'Angelo et al. combine coarse-grained simulations of smart territories with detailed communication simulations, requiring MLS to coordinate simulators, synchronize state, and manage interoperability at scale [7]. In manufacturing, Thiede et al. discuss multi-level simulation across manufacturing system levels, where coupling depends on the simulation goal, transferred parameters, synchronization, and available data [19]. Agent-based and multi-resolution research adds further variants: Morvan et al. specify inter-level interactions including perception and influence [16]; Reynolds et al. show that aggregation and disaggregation across resolutions can create invalid states when corresponding attributes are not kept coherent [17]; and Serena et al. find terminology and integration mechanisms diverse across domains [18]. Scalability recurs both for large entity counts [7] and for machines with many components [11].

Established co-simulation standards address a different problem. FMI defines a tool-independent packaging format and two coupling modes for exchanging executable simulation units [3]; HLA defines rules, an interface specification, and an object model template for federating distributed simulations [6]. Both specify how already-selected simulators exchange data and synchronize execution, not whether the coupled artefacts are semantically compatible. The HLA authors state that the architecture "in itself is insufficient to guarantee interoperability" and requires additional consistency in the internal representation of federated simulations [6], and FMI packages a Functional Mock-up Unit without determining whether combining units from different abstraction levels yields a valid simulation [3]. What MLS lacks is a way to check compatibility preconditions, independently of the coupling standard.

2.2 Multi-Level Modeling and DMLA

MLM addresses situations in which the two-level distinction between types and instances is too restrictive. Atkinson et al. compare MLM approaches and discuss recurring requirements such as classification over more than one level and deep characterization [1]. The meaning of a level is itself debated: Kühne argues that "level" is used in several ways and that unclear level semantics can obscure the modeling problem rather than clarify it [12]. De Lara et al. advise applying MLM where patterns such as type-object structures and extensible classification are actually present [9]. Lange and Atkinson emphasize that specialization has its own semantics and is not identical to classification through instantiation chains [13].

Several MLM environments already combine modeling with a language for constraints or behavior. MetaDepth supports deep metamodeling with potency and hosts OCL-derived constraint and action languages, so constraints can span meta-levels [8]. The FMML^x is realized in the XModeler, where models and code share one representation and XOCL serves as a programming and model-analysis language [10]. Stating and executing a compatibility condition is therefore not by itself distinctive of DMLA; Section 5 returns to what is.

Less common is DMLA's level organization. Urbán, Mezei, and Theisz present DMLA as a formal multi-level modeling approach based on a small core, a bootstrap, and validation formulae for instantiation semantics [15, 20], later extended so that operations and validation logic are expressed in the same framework [21]. Bácsi et al. characterize DMLA as *level-blind*: levels are not explicitly modeled, each entity may refer to any other entity along the meta-hierarchy as long as the validation rules hold, and although every entity has a refinement depth, DMLA imposes no global level organization that would require refinement relationships to align with each other [2]. Level-blindness is thus not the absence of a fixed level count, but the absence of a mandatory alignment between independent refinement chains. Mezei et al. apply this style in the MULTI Warehouse Challenge, using entities, slots, operations, annotations, and custom validation to model flexible refinement chains [14].

3 Challenges and Modeling Needs

The work reviewed in Section 2.1 exhibits recurring challenges. The relevant "levels" are not uniform, as Table 1 summarizes, and they differ in what must be exchanged: events, state variables, aggregated quantities, traces, or analysis results. Coupling depends on available data and synchronization assumptions [19], consistency must hold across resolutions and simulator switches [17], growing scenarios raise scalability concerns [7, 11], and coupling standards leave semantic compatibility unchecked.

The MLS rows of Table 1 share a property that matters for MLM: none is an instantiation relation. A fast functional simulator and a

Table 1: Notions of "level" in MLS and MLM, and what relates two adjacent levels.

Notion of "level"	Relation between adjacent levels	Example from Section 4
MLS: resolution, fidelity, cost	none; alternative views of one system	transaction-level vs. instruction-accurate run
MLS: time granularity	sampling or aggregation	20 ms task period vs. bus transactions
MLS: verification stage	artefacts replaced, scenario kept	MiL / SiL / virtual ECU [23]
MLM: classification	instance-of	`PlatformDescription` / `RenodeCortexR52`
MLM: specialization	is-a	`Simulator` / `FirmwareEmulator`
DMLA: refinement depth	refines, without global alignment	`ModelArtefact` / `FirmwareBinary` / `ArmFirmware`

detailed instruction-set simulator differ in fidelity and cost without one being an instance of the other. This does not mean that MLS contains no instantiation chains; they run *orthogonally* to the resolution ordering, inside a single MLS stage. A simulator kind such as `FirmwareEmulator` is instantiated by a configured emulator, which produces a concrete run; `FirmwareBinary` is instantiated by `ArmFirmware`, which is instantiated by the binary of a particular build. These chains differ in length per stage and never cross resolutions; in a DMLA model they appear as refinement chains (Table 1). An MLM formalism for MLS must therefore keep classification, specialization, refinement, and cross-resolution compatibility apart rather than compressing them into one relation [1, 9, 13].

We condense these challenges into six modeling needs for MLS compatibility modeling:

(N1) explicit descriptions of simulator inputs, outputs, states, and assumptions;
(N2) compatibility checks between model artefacts and simulators;
(N3) *traceable mappings* between state representations at different resolutions, i.e., mappings that are themselves modeled elements, so that source and target state, preconditions, and consumed artefacts can be inspected and re-checked rather than buried in glue code;
(N4) explicit representation of coupling and synchronization requirements;
(N5) terminology that distinguishes simulation resolution from modeling classification;
(N6) *modeling scalability*: adding a simulator, artefact type, or rule should mean adding model elements, not editing the checking logic, so that the model grows with the number of simulators rather than of simulator pairs.

These needs motivate a formalism expressing both structure and validation logic, without treating MLS itself as an MLM problem in the strict sense.

4 ECU Software Architecture Example

Figure 1 illustrates the proposed use of DMLA for an automotive ECU analysis chain: a lane-keeping function that processes camera input, detects lane markings, and issues steering commands. Engineers may first analyze the software architecture, then inspect communication and timing on a transaction-level hardware model, and finally execute the firmware on an emulated platform. These steps are often called simulation levels, but per Table 1 they are not instantiation levels: they are analysis views with different required artefacts, state representations, and validity assumptions.

In a DMLA-style model, the three simulators refine a generic `Simulator` entity and the inputs refine `ModelArtefact`; these refinement chains are independent of the MLS execution order. The relation between the stages is compatibility: a simulator consumes particular artefacts, produces particular states, and supports a switch only when a state mapping exists.

Listing 1 sketches the model in the style used for the MULTI Warehouse Challenge [14]. A simulator is an entity whose slots hold the artefact types it requires, the state kinds it produces and consumes, and its semantic constraints; a concrete simulator refines that entity and fills the slots (N1). `CanRun` checks input completeness against `Requires` and then the constraint slots (N2). `CanSwitchTo` looks for a modeled state mapping between producing and consuming simulator and checks that the artefacts the mapping requires are present (N3, N4). Both operations traverse slots instead of enumerating simulator pairs, so a new simulator is one more entity and a new coupling one more `StateMapping`, without touching the operations (N6).

Concretely, the firmware emulator's constraints demand that firmware and platform instruction sets agree, that the platform provides camera, timer, and bus peripherals, and that trace and platform share a time unit; the mapping from `TLMState` to `FirmwareState` requires a symbol map and a timer configuration.

`CanSwitchTo` is a precondition check evaluated on the model: it decides whether a switch is justified by an explicitly modeled state mapping. It neither performs the switch nor transfers state; executing it remains the task of the simulation infrastructure, for which FMI or HLA are the appropriate mechanisms.

The benefit is explicit, automatable compatibility reasoning, not faster simulation. A missing sensor trace becomes a failed completeness check, a `riscv32` binary on an `armv8-r` platform a failed constraint, an absent symbol map a failed switch check, and a deadline violation in an executed trace a failed post-execution check.[1]

5 DMLA as a Candidate for MLS Compatibility Modeling

What is, and what is not, an argument for DMLA. One argument should be discarded first: that MLS artefacts do not share a fixed number of levels is not discriminating, since most MLM approaches prescribe no level count either. The relevant property is stronger: DMLA imposes no global level organization at all, so independent refinement chains need not be aligned [2]. This matches Table 1: instantiation chains inside an MLS stage differ in length, and a shared level index would relate a firmware build to a bus transaction merely because both sit three steps below their root. Level-adjuvant approaches can express such settings, but require a decision about level assignment before the compatibility question is asked.

The ability to state and execute a compatibility condition is likewise not specific to DMLA. Constraints over an executable

[1] A reference implementation validating one valid and four deliberately invalid scenarios is available at https://github.com/SebastianWeberFZI/dmla-mls-ecu-example.

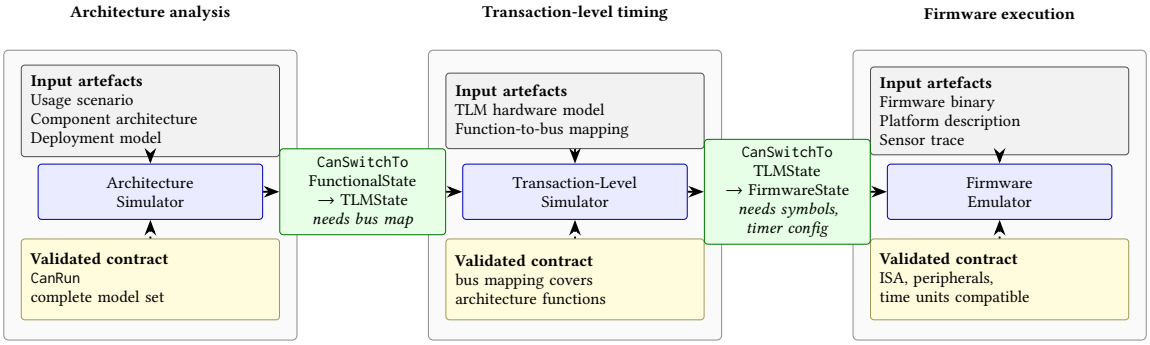


Figure 1: DMLA-style compatibility model for the ECU chain: lanes group artefacts, simulator, and validity conditions; mapping nodes make switches explicit.

```
entity Simulator refines Base {
  slot {T:ModelArtefact} Requires;
  slot {T:Constraint}    Constraints;
  slot {T:StateKind}     Produces, Consumes; }
entity FirmwareEmulator refines Simulator {
  slot Requires = {FirmwareBinary, PlatformDescription,
                   SensorTrace};
  slot Consumes = {FirmwareState};
  slot Constraints = {SameProperty(FirmwareBinary.isa,
                       PlatformDescription.isa), ...}; }

operation Bool CanRun(Simulator s, Base[] given) {
  foreach (Base r in s->Requires)             // N1
    if (FindByType(given,r)==null) return false;
  foreach (Base c in s->Constraints)          // N2
    if (!c.Check(given)) return false;
  return true; }

operation Bool CanSwitchTo(Simulator a, Simulator b,
                           Base[] given) {
  StateMapping m = FindMapping(a->Produces, b->Consumes);
  if (m==null) return false;                  // N3
  foreach (Base r in m->Requires)             // N4
    if (FindByType(given,r)==null) return false;
  return true; }
```

Listing 1: DMLA model excerpt for the ECU chain: the simulator entity, one refinement, and the two validation operations.

model—as in MetaDepth with its OCL-derived languages [8] or in the FMML^x and XModeler with XOCL [10]—can express the checks of Listing 1. Needs N1–N4 are therefore addressable by any MLM approach pairing a modeling language with a constraint or action language, and our discussion of them is an argument for MLM in general. Specific to DMLA is the combination with level-blindness (N5) and validation formulae belonging to the same self-describing model rather than a separate constraint layer [20, 21].

Clarifying level semantics (N5). MLS levels capture analysis concerns such as speed, detail, resolution, and synchronization, whereas DMLA refinement chains describe how entities become more concrete or constrained. They may coincide in a domain model but should not be assumed to. Since DMLA has no explicit levels, the guidance is not to seek a DMLA level per MLS resolution, but to model resolutions as distinct simulator and artefact entities related by compatibility, reserving refinement for genuine concretization [14].

Separating modeling relations (N2). An MLS model should distinguish refinement between artefacts, specialization between simulator kinds, compatibility between a simulator and its inputs, and mapping between state representations, since instantiation, classification, and inheritance differ semantically [1, 13]. DMLA supports this by representing relations as slots with associated validation logic (Listing 1), which is where executable metamodeling becomes relevant: static structure alone does not capture behavioral semantics, scheduling, and execution concerns [5]. However, DMLA offers no built-in specialization or inheritance relation, so an is-a relation between simulator kinds must be encoded as a slot plus validation logic rather than being a language primitive. In the ECU example this is workable, but it makes specialization a modeling convention instead of a language guarantee.

Addressing data availability (N1). Coupling often depends on available input data, transferred parameters, and synchronization needs [19]. DMLA cannot create missing data but can make it explicit: a simulator entity specifies required input slots, accepted artefact types, units, timing assumptions, and completeness constraints, and validation rejects a combination when required information is absent. For pure compatibility checking, artefact descriptors can be modeled statically, as in our example; reflecting a running configuration requires acquisition from the tool chain, an integration problem the formalism does not solve.

Addressing consistency and coupling (N3, N4). Consistency in MLS depends on whether state representations stay coherent across resolutions and simulator switches [17]. Modeling the mapping as an entity gives its preconditions a place to be represented, checked, and evolved, and `CanSwitchTo` makes them executable. This does not guarantee that the mapping preserves all relevant semantics and, as noted in Section 4, does not carry out the switch.

Addressing scalability (N6). We claim only modeling scalability as defined in Section 3: because the operations traverse slots, the n-th simulator adds one entity and its mappings instead of a branch in hand-written coupling logic. A `CanRun` check could thus

gate whether an FMU is imported or a federate joins a federation, complementing standards that solve technical coupling but not the growth of compatibility rules [3, 6]. It says nothing about simulation runtime, and the cost of evaluating validation operations on large models remains unmeasured.

6 Conclusion

DMLA offers a modeling and validation layer for MLS: a place where artefact requirements, simulator constraints, and state-mapping preconditions become model elements checked by operations of the same model. We condensed challenges from selected MLS literature into six modeling needs, separated the notions of level involved (Table 1), located where instantiation chains actually occur in MLS, and instantiated the argument with an ECU example, a DMLA model excerpt, and a reference implementation. Ignoring the distinction between MLS and MLM levels risks modeling simulator taxonomies, artefact refinements, and compatibility relations with the wrong semantics.

The expected benefit is not acceleration but a more explicit and reusable representation of assumptions that remain implicit in many MLS settings. Three limitations qualify this position. First, N1–N4 can be addressed by any MLM approach with a constraint or action language; only the combination with level-blindness is specific to DMLA. Second, DMLA provides neither an explicit specialization relation nor a non-transitive instantiation relation, which some regard as the defining element of MLM, so is-a relations become validated slots. Third, we have not measured the cost of evaluating validation operations on large compatibility models.

Future work will integrate the ECU compatibility model with existing simulation tools, express it in DMLA itself rather than a DMLA-inspired reference implementation, and evaluate whether explicit validation improves reusability and correctness of coupling decisions.

Acknowledgments

This work has received funding from the European Chips Joint Undertaking under Framework Partnership Agreement No. 101139789 (HAL4SDV) including national funding from the German Federal Ministry of Research, Technology and Space (BMFTR) under grant number 16MEE0468. The responsibility for the content of this publication lies with the authors. This work was funded by the DFG (German Research Foundation) – project number 499241390 (FeCoMASS), the Topic Engineering Secure Systems of the Helmholtz Association (HGF), KASTEL Security Research Labs, Karlsruhe, and the DFG – SFB 1608 – 501798263.

References

[1] Colin Atkinson, Ralph Gerbig, and Thomas Kühne. 2014. Comparing Multi-Level Modeling Approaches. In *Proceedings of the 1st Workshop on Multi-Level Modelling (MULTI 2014) (CEUR Workshop Proceedings, Vol. 1286)*. CEUR-WS.org, Valencia, Spain, 53–61. https://ceur-ws.org/Vol-1286/p6.pdf

[2] Sándor Bácsi, Arne Lange, Thomas Kühne, Gergely Mezei, and Colin Atkinson. 2021. Melanee and DMLA: A Contribution to the MULTI 2021 Collaborative Comparison Challenge. In *2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. IEEE, 556–565. doi:10.1109/MODELS-C53483.2021.00086

[3] Torsten Blochwitz, Martin Otter, Martin Arnold, Constanze Bausch, Christoph Clauß, Hilding Elmqvist, Andreas Junghanns, Jakob Mauss, Manuel Monteiro, Thomas Neidhold, et al. 2011. The Functional Mockup Interface for Tool Independent Exchange of Simulation Models. In *Proceedings of the 8th International Modelica Conference*. Linköping University Press, 105–114.

[4] Johan Cederbladh, Antonio Cicchetti, and Jagadish Suryadevara. 2024. Early Validation and Verification of System Behaviour in Model-Based Systems Engineering: A Systematic Literature Review. *ACM Transactions on Software Engineering and Methodology* 33, 3 (2024), 1–67. doi:10.1145/3631976

[5] Benoît Combemale, Cécile Hardebolle, Christophe Jacquet, Frédéric Boulanger, and Benoît Baudry. 2012. Bridging the Chasm Between Executable Metamodeling and Models of Computation. In *Software Language Engineering*. Springer, 184–203. doi:10.1007/978-3-642-36089-3_11

[6] Judith S. Dahmann, Richard M. Fujimoto, and Richard M. Weatherly. 1997. The Department of Defense High Level Architecture. In *Proceedings of the 29th Conference on Winter Simulation*. 142–149.

[7] Gabriele D'Angelo, Stefano Ferretti, and Vittorio Ghini. 2017. Multi-Level Simulation of Internet of Things on Smart Territories. *Simulation Modelling Practice and Theory* 73 (2017), 3–21. doi:10.1016/j.simpat.2016.10.008

[8] Juan de Lara and Esther Guerra. 2010. Deep Meta-Modelling with MetaDepth. In *Objects, Models, Components, Patterns*. Springer, 1–20. doi:10.1007/978-3-642-13953-6_1

[9] Juan de Lara, Esther Guerra, and Jesús Sánchez Cuadrado. 2014. When and How to Use Multilevel Modelling. *ACM Transactions on Software Engineering and Methodology* 24, 2 (2014), 1–46. doi:10.1145/2685615

[10] Ulrich Frank. 2018. *The Flexible Multi-Level Modelling and Execution Language (FMMLx), Version 2.0: Analysis of Requirements and Technical Terminology*. Technical Report ICB-Research Report No. 66. University of Duisburg-Essen, Institute for Computer Science and Business Information Systems. doi:10.17185/duepublico/47506

[11] Thomas Grass, César Allande, Adrià Armejach, Alejandro Rico, Eduard Ayguadé, Jesus Labarta, Mateo Valero, Marc Casas, and Miquel Moreto. 2016. MUSA: A Multi-Level Simulation Approach for Next-Generation HPC Machines. In *SC '16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 526–537. doi:10.1109/SC.2016.44

[12] Thomas Kühne. 2018. A Story of Levels. In *Proceedings of the MODELS 2018 Workshops*. 673–682.

[13] Arne Lange and Colin Atkinson. 2019. On the Rules for Inheritance in LML. In *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. IEEE, 113–118.

[14] Gergely Mezei, Ferenc Attila Somogyi, Norbert Somogyi, and Gergely Gembela. 2024. Multi-Level Modeling with DMLA: A Contribution to the MULTI Warehouse Challenge. In *Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems*. ACM, 790–799. doi:10.1145/3652620.3688211

[15] Gergely Mezei, Dániel Urbán, and Zoltán Theisz. 2017. Validated Multi-Layer Meta-Modeling via Intrinsically Modeled Operations. In *Proceedings of the MODELS 2017 Satellite Event: Workshops (CEUR Workshop Proceedings, Vol. 2019)*. CEUR-WS.org, Austin, TX, USA, 213–217. https://ceur-ws.org/Vol-2019/multi_1.pdf

[16] Gildas Morvan, Alexandre Veremme, and Daniel Dupont. 2011. IRM4MLS: The Influence-Reaction Model for Multi-Level Simulation. In *Multi-Agent-Based Simulation XI*, T. Bosse, A. Geller, and C. Jonker (Eds.). Lecture Notes in Computer Science, Vol. 6532. Springer, 16–27. doi:10.1007/978-3-642-18345-4_2

[17] Paul F. Reynolds, Anand Natrajan, and Sudhir Srinivasan. 1997. Consistency Maintenance in Multiresolution Simulation. *ACM Transactions on Modeling and Computer Simulation* 7, 3 (1997), 368–392. doi:10.1145/259207.259235

[18] Luca Serena, Moreno Marzolla, Gabriele D'Angelo, and Stefano Ferretti. 2023. A Review of Multilevel Modeling and Simulation for Human Mobility and Behavior. *Simulation Modelling Practice and Theory* 127 (2023), 102780. doi:10.1016/j.simpat.2023.102780

[19] Sebastian Thiede, Malte Schönemann, Denis Kurle, and Christoph Herrmann. 2016. Multi-Level Simulation in Manufacturing Companies: The Water-Energy Nexus Case. *Journal of Cleaner Production* 139 (2016), 1118–1127. doi:10.1016/j.jclepro.2016.08.144

[20] Dániel Urbán, Gergely Mezei, and Zoltán Theisz. 2017. Formalism for Static Aspects of Dynamic Metamodeling. *Periodica Polytechnica Electrical Engineering and Computer Science* 61, 1 (2017), 34–47. doi:10.3311/PPee.9547

[21] Dániel Urbán, Zoltán Theisz, and Gergely Mezei. 2018. Self-Describing Operations for Multi-Level Meta-Modeling. In *Proceedings of the 6th International Conference on Model-Driven Engineering and Software Development (MODELSWARD)*. SciTePress, 519–527. doi:10.5220/0006656105190527

[22] Sebastian Weber, Thomas Weber, Robert Heinrich, and Jörg Henß. 2024. Combining a Functional Simulation with Multi-Level Timing Simulation for Software Architecture Models to Improve Extensibility. In *2024 IEEE 21st International Conference on Software Architecture Companion (ICSA-C)*. IEEE, 74–78. doi:10.1109/ICSA-C63560.2024.00019

[23] Anna Yang, Woo Jin Han, Hyun Suk Cho, Dong-Woo Koh, and Jae-Gon Kim. 2026. A Model-Based Design and Verification Framework for Virtual ECUs in Automotive Seat Control Systems. *Electronics* 15, 3 (2026), 569. doi:10.3390/electronics15030569