

# Challenges in Modeling and Unmodeling Emergence, Rule Composition, and Networked Interactions in Complex Reactive Systems

Assaf Marron<sup>1</sup>, Irun Cohen<sup>2</sup>, Guy Frankel<sup>1</sup>, David Harel<sup>1</sup> and Smadar Szekely<sup>1</sup>

<sup>1</sup>Department of Computer Science and Applied Mathematics, Weizmann Institute of Science, Rehovot, 76100, Israel

<sup>2</sup>Department of Immunology and Regenerative Biology, Weizmann Institute of Science, Rehovot, 76100, Israel

**Keywords:** Modeling, Simulation, Emergent Entities, Rule-Based Specifications, Incremental Development, Evolution.

**Abstract:** Models of complex systems, both human-made and natural, assist in making critical decisions with regard to function, safety, economy, the environment and more. In this position paper, we explore the difficulty in modeling several innate properties of such systems, including: (i) the frequent emergence of new entities (like group effects and temporal patterns) and the system's reaction to such emergence; (ii) the essence of the system's reactive behavior as a rich composition of stand-alone rules; and (iii) the vast number of internal and external interactions that the system engages in. For each of these challenges we propose some implications to modeling—methodological approaches that can help address it and potential support in modeling languages and tools. We introduce the concept of *unmodeling*—formally defining model entities and behaviors that are excluded from model execution—and discuss how unmodeling enhances model quality, supports incremental enhancement, and facilitates evaluation. This report emanates from our research and development in modeling languages and methods and our present research in biological evolution. We believe that analysis and implementation of these principles have general applicability in model development and assessment.

## 1 INTRODUCTION

Complex systems, both human-made and natural, are often studied with the aid of computer models and simulations (Saprykin et al., 2019; Mencuccini et al., 2019; Liu et al., 2021). Common uses are exploration and prediction of system behaviors under various conditions and the testing of theories about mechanisms underlying the system. For an engineered system, performing tests on models can enhance overall testing and validation (Briand et al., 2016). Models and simulations also play a growing role in education, in teaching and learning of STEM skills like abstraction, computational thinking, mechanistic reasoning, analytical observation and more (Armoni et al., 2021; Haskel-Ittah, 2022).


When using a model for critical decisions, its correctness in representing the modeled system is important. To this end, various techniques are ap-


plied including testing and formal verification of components that simulate well-understood system elements, and checking that model predictions align with already-known real-world effects (Christin et al., 2021; Sankararaman and Mahadevan, 2015). However, system complexity often prevents complete validation or even measurement of the quality of a model.


Solving modeling difficulties requires recognition and analysis of specific issues. In this paper, we document three properties that are common to many complex reactive systems and argue that even though these properties are hard to model, they must be captured in models that are used in critical decisions. For each of these properties of the modeled system we propose some techniques that modelers can apply to deal with the challenge, and we outline possible features in modeling languages (Including UML, SysML, Statecharts, Live Sequence Charts, MATLAB/Simulink, NetLogo, etc.) and associated modeling platforms that can help modelers in these tasks.


We focus on three challenges:


**Reflective Emergence.** Complex systems always give rise to emergent properties and entities. These in turn are detected and reacted to not only by external observers, but also by the system itself. How can

<sup>a</sup> <https://orcid.org/0000-0001-5904-5105>

<sup>b</sup> <https://orcid.org/0000-0002-3906-6993>

<sup>c</sup> <https://orcid.org/0000-0001-5809-3455>

<sup>d</sup> <https://orcid.org/0000-0001-7240-3931>

<sup>e</sup> <https://orcid.org/0000-0003-1361-1575>

such delayed reactions be programmed in a model?

**Diversity of Rule-Composition Semantics.** The mechanisms driving complex reactive systems, and hence their models, are subject to description or specification as a set of rules, each specifying a behavior or a constraint under certain conditions or following a certain sequence of events. These rules are subject to complex, diverse, sometimes tacit, composition semantics. Presently, modeling languages may enforce fixed, universal, "one size fits all" semantics for all rule composition. Human-written rule-based descriptions of, or requirements for, systems and models are rarely associated with any formal semantics. That being the case, how can a system's behavior be accurately specified or understood?

**Networked Interactions.** Complex systems operating in the real world are part of a vast network of internal and external interactions. Clearly, not all of these interactions can be included in one model. How can stakeholders assess the final choices of what to model and what to abstract away?

While there is much research on model driven engineering and related challenges (Bucchiarone et al., 2020; Troya et al., 2021), and on modeling and simulation of complex engineering systems like digital twins (Liu et al., 2021), we have not seen detailed, focused discussions of the system properties and ensuing challenges that are discussed here.

Additional system properties and modeling challenges that deserve similar treatment surfaced during our work. We defer such discussions to future work. These include: (i) The pervasiveness of order, and the recognition that randomness is at times only a convenient simplifying abstraction; (ii) the unending chains of causes and effects; (iii) the importance of distinguishing individual entities, regardless of how similar other entities of a given type may be; (iv) The coexistence and interaction of events, actions and processes that operate at different time scales.

Some modeling requirements can be handled through *unmodeling*: documenting, and even representing as model elements, entities and effects that are excluded from model execution, model transformation, etc. Unmodeling is actually a modeling activity in which modelers delineate the model's scope by detailing what will be included in simulations and formal analysis and processing and what will be excluded. By explicitly specifying this scope, unmodeling can reduce accidental omissions of important aspects, support planning and incremental development, and add depth to the assessment of a model's applicability to a given decision task.

As described in Section 2, this paper was triggered by our research in modeling biological evolution and

by our engagement with the use of modeling in science education. Yet, we believe that the ideas presented here transcend our specific contexts and can be applied in systems ranging from the global economy through transportation and autonomous vehicles to biological contexts like the immune or nervous systems of organisms; indeed, the examples throughout the paper come from diverse domains. Furthermore, while some of the points we raise may be familiar, we believe that these challenges deserve additional attention and a more central role in methodologies for system modeling and in system engineering at large.

## 2 MOTIVATION

### 2.1 Modeling Biological Evolution

A key question in natural sciences is how can order emerge from disorder considering the universal tendency toward disorder dictated by the second law of thermodynamics (Schrödinger, 1944). Furthermore, one asks, why did certain forms of order evolve, like the particular species of our biosphere, and not others. Study of the respective theories is often supported a variety of models and modeling methods. Our own research (Cohen and Marron, 2020; Cohen and Marron, 2022) is inspired by many questions that the classical Darwinian theory of natural selection and survival of the fittest, and its synthesis with genetics, leave open. We argue that the emergence, sustainment, and evolution of species is driven by the ability of networks of repeating interactions to retard the inevitable destruction of organized structures and processes. To put the discussion in context, consider the dependence of all multi-cellular organisms on a diverse microbiome (Blaser, 2014), the networks in which trees and fungi exchange energy, matter and information (Simard, 2018), and the rich diversity and symbiosis in a coral colony within and among species (Rosenberg et al., 2007). The networks are sustained not only by the value of what is exchanged or done in each interaction, but by the very repetition of patterned interactions. When such networks experience innovations like genetic mutations or environmental changes, they may be reshaped into different sustained configurations. This is evolution. Furthermore, we show that a process we term *natural autoencoding* is the mechanism by which typed structures are created, each with its *species interaction code*, a set of shared essential interactions that enable individual organisms to fulfil their sustaining role in their internal and external networks. Winning in competitions over limited resources and having reproduc-

tive advantages are only part of the picture. The implications of this theory of evolution, extend beyond pure biology, affecting human social, economic, governance and education perspectives.

Modeling natural autoencoding, with its emphasis on the emergence of patterns subject to numerous laws of nature and the unbounded web of natural interactions, internal and external to each organism and species, presents intriguing technical challenges. In this paper we outline domain-independent methodologies for tackling three of these difficulties.

## 2.2 Tools for Science Education

Our group is also associated with a variety of efforts in science education (See, e.g., (Armoni et al., 2021); <https://www.iamplethora.com>).

Consider the following simplified account of developing a scenario-based model for educational purposes, using the Plethora Science platform (presently under development) describing certain aspects of an ecological niche. Initially the model consists of rules, or scenarios, that can be summarized as:

1. When the days are longer than  $h_{min}$  hours, flowers bloom; otherwise, flowers do not bloom.
2. When the temperature is above  $t_{min}$  degrees, bees are active; otherwise, bees hibernate.
3. For months  $m_1$ - $m_{12}$ , the number of daylight hours  $h_1$ - $h_{12}$ , respectively, is specified.
4. For months  $m_1$ - $m_{12}$ , the temperatures  $t_1$ - $t_{12}$ , respectively, are specified.
5. Active bees are attracted to flowers and interact with them.

Students experiment with various parameters to visualize a range of behavior patterns for bees and flowers. However, for the model to show that if the temperature rises above  $t_{min}$  in months when daylight hours are fewer than  $h_{min}$ , the now active bees die of hunger, additional rules are needed, like:

6. After interacting with a flower, a bee gains energy.
7. An active bee constantly loses energy.
8. If a bee loses all its energy, it dies.
9. After dying, a bee cannot interact with flowers.

Rule #9, about the implications of dying, is of particular interest. For a model to be correct, or useful, tacit assumptions must often be explicitly modeled. Through counterexamples to asserted properties, formal verification of models can help uncover such missing rules. We also note that the dependency of flowers on bee pollination is not presently modeled.

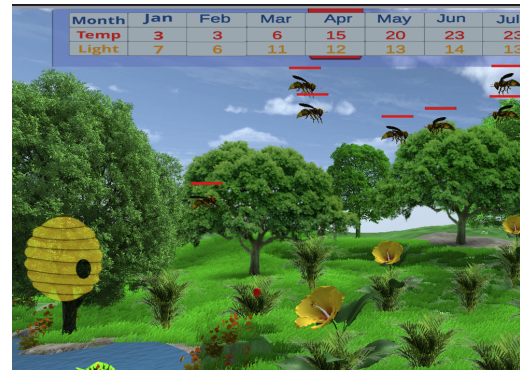


Figure 1: A snippet from a scenario-based model of climate dependent interaction of bees and flowers. Model development is an incremental discovery and specification process. (Image credit: Plethora Science learning platform).

One then asks whether such model completeness aspects can be supported, at least partially, by modeling tools. The coming sections contribute to an affirmative answer.

## 3 REACTION TO EMERGENCE

**Assertion:** Complex real-world systems detect the emergence of entities and behavior patterns and react to it.

**Rationale:**

A key goal of modeling complex systems is the detection of emergent properties. For example, modeling the gravity and inertia affecting a metal rod hanging diagonally on a nail should yield a swinging pendulum motion with properties like period, amplitude or trajectory; and, when modeling the behavior of vehicles on a highway, one may be interested in seeing where, when and what kinds of traffic jams are formed. But, if an emergent entity or property  $P$  in a model of a system  $S$  is of interest to humans, it is likely that  $P$  affects the world outside of  $S$ . It is thus likely that  $S$  will also detect  $P$  and react to it, changing  $S$ 's own behavior, and possibly affecting  $P$  too. For example, drivers react to traffic congestion and change their routes or travel schedules accordingly. Such reactions are themselves emergent entities that are reacted to; for example, when many drivers choose the same alternate route to avoid delays, the detour may become congested as well, and predicting such secondary congestion may further affect driver behaviors. In a biological context, the emergent gathering of organisms around a source of food may cause overcrowding, reduction in populations in other areas, and other effects, triggering other chains of events that the model may not have been prepared for, like the ac-

cumulation of toxic refuse, or limitations in motion or in reproduction, migration of other organisms, etc.

The modeling challenge here is as follows: If the emergent entity is planned for in detail, this may guide and constrain the model, make its predictions self-fulfilling prophecies, and hinder the perception of other relevant emergence. But, if the emergent entity is unexpected, there may be no sensors in the model for its effects, and it may be missed altogether, alongside with the reactions to it.

#### Some Implications to Modeling:

**Anticipating Emergence.** For types of emergent properties and entities that the modelers expect and that may persist in the system, add the ability to sense them and to react to them into the basic model. The required sensors may be reused from programmed components that aid in automated simulation analysis. One still has to model and simulate the system's reactions to these emergent properties. For example, in detecting the emergence of traffic jams in transportation models, the model must be able to detect not only long lines of slow cars, but also secondary effects like route changes or blocked intersections. However reaction to emergent properties that are undesired or do not exist in the real system, may not need be modeled. For example, in modeling a chemical reaction that might cause an explosion, if in simulations explosions indeed occur, one is likely to change the real system, or the real environment, or the specification of the model.

**Responding to Emergence.** When noticing an emergent entity that was not expected: (i) discover properties of this entity and interactions and behaviors that it is capable of; for example, if it is discovered that certain model entities assemble into clusters, these clusters have properties like size and speed and they may interact with other entities, say, by serving as an obstacle to motion or by creating a shadow; (ii) determine the system's reaction to these properties and behaviors. Where applicable, and when not already implied directly by lower-level rules, the model could include all these new entities, properties, and behaviors. Note that real world processes too often evolve in such an incremental reactive manner.

**Unmodeling Emergence.** Such incremental modeling may be unbounded, almost Sisyphean, and must stop at some point. The decision to stop or suspend this iterative incremental modeling should be justified and documented; justifications may be related to time scales or abstraction levels that are larger or finer than what is necessary for the purposes of the model.

## 4 REACTIVE RULE COMPOSITION

**Assertion:** Specifying or describing the behavior of a complex reactive system requires composition of self-standing rules.

#### Rationale:

Reactive systems respond to stimuli based on the nature of the input and the current system state. For example, the software for a web application includes specification of the responses to different combinations of user inputs and system states. In some cases, the responses are specified in procedural programs, checking various aspects of the input and the state before making a choice. However, in models of complex systems, this may not be possible, since the action is not a single choice but the result of the composition of multiple concurrent rules specifying actions and constraints. Each such rule has a narrow view of the system and the environment, never checking all possible state variables. Naïvely, a rule may be seen as stating "Always, when condition *C* holds, take action *A*!" or "Always, when *C* holds, forbid *A*!". However, despite the word "Always" (which is often only implied) such rules are frequently subject to exceptions, probabilistic choice, and overriding priorities.

These rules may be coded in the executable rule-based and scenario-based specification of a model, or they may comprise the essence of the requirements for development or validation of such a model. Such rules consolidate scientific knowledge about nature, or expertise in some engineering domain. As such, knowledge bases are incrementally built, new rules of reactive behavior may be added in a stand-alone manner, considering only a small number of other related requirements. For example, consider regulators, or QA engineers trying to find out whether an autonomous vehicle (AV) will "always obey a red traffic light", "always obey the instructions of a police person", and "always put the safety of humans first". At various states, these requirements may conflict with each other, and some priority order or other composition semantics must be specified, for example:

**Collective Execution.** This is probably the most standard composition. All actions triggered by a program are eventually carried out in the real world in some order; for example, consider an AV activating its signal light, slowing down and turning right, all at the same time. Documentation of the semantics of the execution environment, like Rhapsody for statecharts (Harel and Kugler, 2004), should specify this execution order, be it sequential (say, by time stamp of activation command), random, priority-based, or arbitrary (say, by rule sequence number), etc.

**Aggregate Execution.** Multiple actions are carried out in a way that their effects are joined or summed up. For example, consider two AVs joining forces in pulling a load in the same direction. The effect experienced by the load is the sum of the two forces.

**Exclusionary Execution.** Only one of a set of tentative actions is carried out. For example, an AV may have to choose between stopping at a red traffic light, or driving forward in response to a concurrent police person directive; or, two collaborating robots that intend to remove the same faulty widget from a conveyor belt, may have to decide which of them should act, and which one should retreat. Such choices may be based on priorities, probabilities, vote counting, and more. A variant of exclusionary execution is **unification**. Two nearly identical actions are triggered by different rules or software components, only one of them is executed, but both components continue as if “their” action was executed, unaware that the two actions were unified into one. For example, two components concurrently delete a database record; the record is deleted once, but each component proceeds as if “its own action” was successful.

**Intersection of Conditions.** Self-standing rules may specify multiple conditions that should be in effect concurrently. For example, specifying for an AV multiple constraints for minimum and maximum speeds, minimum and maximum distances from certain objects, etc., and all have to hold at the same time.

**Meta Specifications.** Rule specification may include references to other rules. For example, a rule  $R_1$  may specify that when unit  $U_2$  is in state *failed* other units should not comply with rule  $R_2$  associated with  $U_2$ .

The details of composition semantics may be critical for model correctness and/or usefulness. For example, in the physical world, when two equal opposing mechanical forces operate on an object, the object remains at rest; in a model, depending on the time scale and synchronization and visualization of processes, the forces may appear to work one at a time, causing the object visualization to wobble.

In theory, one would need to specify the composition semantics between each pair, or even each set, of rules in the model’s specification. This applies both to the executable specifications that drive actions in a simulation run, and to descriptive or interpretive specifications explaining to stakeholders like modelers, customers, users, and regulators, what the system does or should do under different conditions.

When domain experts examine a model, they need to be convinced that their requirements and assumptions are implemented in a manner aligned with their domain knowledge. Also, for every behavior observed in a simulation, one needs to know if it has

been programmed directly or has emerged from programming of other entities.

### Some Implications to Modeling:

**Pairwise Rule Composition Analysis.** Consider all actions that the system may trigger and all constraints that it may impose. For every pair, or set of such actions and/or constraints, determine and document how they should be composed if they occur together.

**Defining Concurrency.** As a sub-task of the above, for every such pair or set, document what “occurring together” means. Various possibilities include: (i) for systems that have synchronization points, being initiated at the same synchronization point; (ii) for actions and conditions that have a non-zero duration, having some of their duration overlap; (iii) for serial actions that may be queued or delayed, being queued and awaiting execution at the same time.

**Group-Wise Rule Composition Analysis.** Organizing rules in groups (say, based on common effects, or common resources) may reduce the complexity of rule-composition analysis. Since examining all possible pairs of actions may be impractical, determining fine grain composition only within each group, and for each pair of groups, coarse-grain composition for all pairs of actions that belong to these two groups.

**Semantics Refinement.** The varieties of composition semantics listed above (concurrent, aggregate, exclusionary, intersection and meta specification) do not cover all possibilities and variants within these categories can also be formalized. For example, consider logical operations between rules (AND, OR, and fuzzy logic), weighted voting, maximum or minimum consensus, Etc.

## 5 NETWORKED INTERACTIONS

**Assertion:** Complex real-world systems participate in a vast number of networked interactions. Many interactions will be unmodeled.

### Rationale:

Almost by definition, a model that is implemented in a computer program represents a closed system. When the environment of that system is included in the model, this environment is regularly subjected to a limited number of behavioral assumptions, and the system of interest and its environment are modeled as a closed system. However, any system that operates in the real world, interacts with numerous other systems. For example, an AV is exposed to other road users, weather conditions, road wear and tear due to environmental conditions and due to use and abuse

by other vehicles, construction projects, etc.; furthermore, such a vehicle leaves its own mark on its environment: most importantly, its presence and behavior affects the behavior of other road users; then there are the wear and tear, pollution, and noise that it creates; and, we should not forget the effects of stalled vehicles (including the modeled vehicle), accidents, commands imposed by passengers and external controllers, effects of changing fuel costs, and more.

And, the biological realm, as mentioned in Section 2 the biosphere consists of countless rich networks. When modeling a cell in an organism, a major factor is its dependence on the function of various organs for providing nutrients, water and oxygen, removing outputs and refuse, maintaining temperature, exchanging sensory information, and much more. Furthermore, within the cell, innumerable biochemical processes concurrently operate (Karr et al., 2015).

Even a seemingly closed human-made real-world system, say, of gas in an insulated tank, is subject to wear and tear, leaks, fluctuations of external temperature, presence of contaminants, and more (Woodward and Pitbaldo, 2010). Stated differently, everything in nature, in the real world, is connected.

The modeling challenges presented by these unbounded interaction networks include the sheer number of different types of interactions, the variability and unpredictability of many of the parameters of these interactions, and, the multiple time scales within which they operate.

#### Some Implications to Modeling:

**Cataloging Interactions.** A model cannot use a blanket assumption that all external influences are accounted for. All assumptions about internal and external influences and interfaces with other systems should be documented, detailing which ones were incorporated and which ones were ignored or abstracted away, with proper justification.

**Outward Effects.** In incremental development of the above interaction catalog, the model should document how the modeled system affects other entities, including additional instances of itself, and how these effects can in turn affect the system in new ways.

**Fixed Conditions.** A preprogrammed condition that stays fixed throughout a simulation, may point at a reductive or simplified assumption about the environment. Identifying and documenting such conditions can help identify overlooked external interactions.

**Frame Management.** In models that are simulated within some frame or canvas, behaviors around the frame of the model should receive special attention—checking what entities, energy and information may

pass through the frame, how agents interact with the frame itself (like bouncing back, or departing from the model), and whether and which properties throughout the model area depend on distance from the frame. For example, when the temperature of the system is regulated by an external source, the temperature may still vary within the modeled space.

**Multi-Model Integration.** When models for relevant entities outside of the system, or of components of the system, are available, modelers should consider connecting the model at hand to such external models. When this is not practical, summaries and abstractions of the behaviors of these distinct models should be added to the model as separate agents. For example, a model of a truck platoon may be connected to larger traffic models, to models of individual other vehicles and of individual trucks, and to additional instances of itself.

**Incremental Interaction Development.** It is impractical to cover all the interactions of a given real-world system in any one modeling project. While some of the above steps may trigger unmodeling, others may draw attention to interactions that were omitted but that should be programmed and incorporated later.

## 6 LANGUAGE AND TOOL SUPPORT

Once the methodological implications and use cases of such issues are well established, modeling languages and tools can be enhanced to help deal with the challenges. The tool-support examples below may provide some directions and may also shed light on the manual processes described in previous sections. These directions can be more systematically extended and refined by examining each modeling issue in the light of each step in accepted modeling methodologies.

**Unmodeling Support.** Going beyond ordinary documentation, the language and modeling platform should accommodate entities that do not participate in the simulation. These may range from brief topic headings to detailed object specifications with properties and method names. Detailed unmodeling can help detect aspects that should be modeled after all, preventing them from “falling through the cracks”.

#### Support for Reacting to Emergence:

**Dynamism and Reflection.** Many programming and modeling languages support dynamic entities: from forking a new process through creating new object instances in defined classes to creating new classes or

changing properties of classes at run time. Reflection allows the running programs to examine their current definitions. Such dynamism and reflection would be mandatory in any modeling platform for almost all model entities.

**Write-Only Variables.** Modeling platforms should enable listing modeled variables or properties that are computed, but that no behavioral rule takes as input. Such data fields may represent emergent properties whose effects on the system should be specified.

**Catalog Anticipated Observed Emergence.** Modeling platforms should prompt the modeler to formally document, or even program as test cases, modeling goals and questions that successful simulation runs should answer, and then suggest adding related emergent entities and properties.

**Automated Detection of Emergence.** Modeling platforms should provide built-in components for the detection of patterns and trajectories. For example, a model for an AV that maps a certain terrain and collects soil samples should be able to automatically detect when the vehicle is moving in circles—revisiting certain areas while missing others—or that in long runs or in large areas the vehicle runs out of fuel or its sample container fills up before the task is done. Modelers may be able to direct the tools to data fields whose behavior should be monitored.

#### Support for Rule-Composition Semantics:

**Visibility of Composition Semantics.** Fine operational semantics of modeling languages and execution environments are often specified in arcane documentation, or have to be learned through experimentation. Instead, they should be clearly documented with executable examples.

**Plug-In Compositional Semantics.** Modeling platforms should accommodate and assist modelers in specifying, documenting and generating examples for extensions and refinements to the platform’s composition semantics.

**Dependency Analysis.** Modeling platforms should support listing model variables/fields that are affected by more than one action or function and resources that are used by multiple rules. Prompt the modeler to specify composition semantics for related rules.

**Concurrency Experimentation.** Manually programming test cases to force concurrent triggering of certain actions and conditions is difficult. Modeling platforms should support specifying such simultaneity, and pace parallel execution of test cases such that the specified events indeed occur together.

#### Support for Modeling Interaction Networks:

**Catalog of Initial Conditions and Fixed Fields.** Modeling platforms should automatically highlight variables and conditions that, once set, do not change, and prompt the modeler to check if they represent closed-system assumptions that should be replaced, or justified, or elaborated upon in unmodeling.

**Separation of Model Observation.** Modeling platforms should clearly distinguish model entities that are part of the modeled system, model entities that simulate the environment of the modeled system, and entities that only help humans observe the model.

**Domain Knowledge.** The modeling platform should include domain specific expertise, like physical laws or traffic regulations, and enable modelers to incorporate these into the models, specify how the system is affected by these environmental characteristics, or unmodel the related effects.

**Rule-Based Summaries.** Regardless of whether the underlying execution engine is rule based or not, the modeling platform should enable modelers to create rule-based model specifications or descriptions at various abstraction levels (with the necessary rule composition semantics). This will support development, validation and use of the model, and streamline integration of model components and of gained knowledge into other models.

## 7 CONCLUSION AND FUTURE WORK

We have shown that unbounded interaction with the environment, reaction to emergence, and the different ways that driving mechanisms are composed are important properties of complex systems that can greatly influence the outcome and usefulness of models. When a given model is not able to predict observed results, handling of these properties should be among the issues that are closely examined.

Even with extensive tool support, the sheer number of entities and interactions that must be addressed is a key challenge. This may be tackled by combination of: (i) abstraction: handling of entities and interactions in groups that share common traits; (ii) scalable artifacts: additional tool support for scalability, like databases, or hierarchical diagrams, adopting, among others, techniques from genetics and biochemistry that facilitate dealing with myriad sequences, molecules and interactions; (iii) sharing and reuse: many of the real-world issues that a complex system must contend with are likely to be relevant in other systems too; sharing domain expertise and model ar-

tifacts can help streamlining and accelerating model development; (iv) training, and "mind set" on behalf of engineers, domain experts and other stakeholders, recognizing in advance the magnitude of the modeling task, and the number and size of the artifacts involved; (v) building domain specific solutions and tools as a step toward more general solutions.

Future tasks include the study additional such issues and challenges, some of which are listed in Section 1, and the development of complex models subject to the informally documented approaches and techniques; these can serve as proofs-of-concept to the incipient modeling methodologies.

Such research and development should contribute to the methodologies, languages and tools of model development and model assessment, and hence, to the usefulness of models in science and society.

## ACKNOWLEDGMENTS

We thank the anonymous reviewers for their insightful comments and suggestions. This work was partially supported by a research grants to David Harel from the Estate of Harry Levine, the Estate of Abraham Rothstein, Brenda Gruss and Daniel Hirsch, the One8 Foundation, Rina Mayer, Maurice Levy, and the Estate of Bernice Bernath, a grant 3698/21 from the ISF-NSFC joint to the Israel Science Foundation and the National Science Foundation of China, and a grant from the Minerva foundation.

## REFERENCES

- Armoni, M., Gal-Ezer, J., Ittah, M. H., Marelly, R., and Szekeley, S. (2021). Computational problem solving in plethora. In *Informatics in Schools: Situation, Evolution, and Perspectives, ISSEP*.
- Blaser, M. J. (2014). The microbiome revolution. *J. of Clinical Investigation*, 124(10):4162–4165.
- Briand, L., Nejati, S., Sabetzadeh, M., and Bianculli, D. (2016). Testing the untestable: model testing of complex software-intensive systems. In *ICSE Companion*, pages 789–792.
- Bucchiarone, A., Cabot, J., Paige, R. F., and Pierantonio, A. (2020). Grand challenges in model-driven engineering: an analysis of the state of the research. *Softw. and Sys. Modeling*, 19(1):5–13.
- Christin, S., Hervet, É., and Lecomte, N. (2021). Going further with model verification and deep learning. *Methods in Ecology and Evolution*, 12(1):130–134.
- Cohen, I. R. and Marron, A. (2020). The evolution of universal adaptations of life is driven by universal properties of matter: energy, entropy, and interaction. *F1000Research*, 9.
- Cohen, I. R. and Marron, A. (2022). The biosphere computes evolution by autoencoding interacting organisms into species and decoding species into ecosystems. *arXiv preprint arXiv:2203.11891*.
- Harel, D. and Kugler, H. (2004). The rhapsody semantics of statecharts (or, on the executable core of the UML). In *Integration of Software Specification Techniques for Applications in Engineering*, pages 325–354. Springer.
- Haskel-Ittah, M. (2022). Explanatory black boxes and mechanistic reasoning. *Journal of Research in Science Teaching*.
- Karr, J. R., Takahashi, K., and Funahashi, A. (2015). The principles of whole-cell modeling. *Current opinion in microbiology*, 27:18–24.
- Liu, M., Fang, S., Dong, H., and Xu, C. (2021). Review of digital twin about concepts, technologies, and industrial applications. *Journal of Manufacturing Systems*, 58:346–361.
- Mencuccini, M., Manzoni, S., and Christoffersen, B. (2019). Modelling water fluxes in plants: from tissues to biosphere. *New Phytologist*, 222(3):1207–1222.
- Rosenberg, E., Koren, O., Reshef, L., Efrony, R., and Zilber-Rosenberg, I. (2007). The role of microorganisms in coral health, disease and evolution. *Nature Rev. Microbiology*, 5(5):355–362.
- Sankararaman, S. and Mahadevan, S. (2015). Integration of model verification, validation, and calibration for uncertainty quantification in engineering systems. *Reliability Engineering & System Safety*, 138:194–209.
- Saprykin, A., Chokani, N., and Abhari, R. S. (2019). Gem-sim: A GPU-accelerated multi-modal mobility simulator for large-scale scenarios. *Simulation Modelling Practice and Theory*, 94:199–214.
- Schrödinger, E. (1944). *What is life? The physical aspect of the living cell and mind*. Cambridge University Press Cambridge.
- Simard, S. W. (2018). Mycorrhizal networks facilitate tree communication, learning, and memory. In *Memory and learning in plants*, pages 191–213. Springer.
- Troya, J., Moreno, N., Bertoa, M. F., and Vallecillo, A. (2021). Uncertainty representation in software models: a survey. *Software and Systems Modeling*, 20(4):1183–1213.
- Woodward, J. L. and Pitbaldo, R. (2010). *LNG risk based safety: modeling and consequence analysis*. John Wiley & Sons.