Unified System Modeling and Simulation via Constraint Hypergraphs

John Morris Gregory Mocko John Wagner, P.E.
Department of Mechanical Engineering
Clemson University
South Carolina, USA
jhmrrs@clemson.edu gmocko@clemson.edu jwagner@clemson.edu

Extended Abstract

The solution of any problem is based on understanding the behavior of the system in which that problem lives. This is as true for scientists understanding global climate change as it is for engineers making nanoscale transistors, policy-makers arguing over nations or doctors diagnosing a newborn infant. Human understanding of these systems is construed in the form of models, which describe how the arrangement of a system's elements produces the behavior observed of the holistic system [28]. Though history provides abundant examples of modelers describing complex relationships such as between space and time or smoking and lung cancer, these models are still limited by one of the most vexing problems of systems engineering [15, 2]: how to have models that can be readily combined to show the behavior of an extended system [20]. This ability for a model to combine with another has been treated under writers referring to it as composability [5, 12, 21], modularity [13, 26, 22, 1], extensibility [24, 30, 7], integration [8, 14], and interoperability [20, 6].

One key to the problem is given by Willems [27], who represented system behavior as a function mapping the state variables of a system. Under this definition, a model of a system consists of a set of variables and the functions mapping between them. This forms a monoidal category [17], with function composition as its joining operation. The insight obtained by treating a system as a function has lead to many domain-specific solutions to interoperability [9, 26, 23, 4, 3, 29, 25]. This paper builds upon the approaches of these scientists by discussing a graphical approach for deconstructing models into the function compositions that describe the system's behavior. First discussed (in an early form) by Friedman [10, 11], this framework represents the state variables of the system as nodes and the functions that constrain them as directed hyperedges (to represent the multiple-arity nature of system relationships). Every path through the hypergraph represents a chain of composed functions. Paths starting from and ending at the same node commute (as in a wiring diagram). The authors refer to this framework as a constraint hypergraph, detailed here [19].

The work of systems engineering is broken into two categories: to define a system and its patterns of behavior, and to simulate a system. The former is the work of theoreticians, the latter is the application of theory. By constructing a system as a hypergraph, the system can be arbitrarily simulated by a pathfinding algorithm that searches for paths from a set of known nodes (inputs) to an unknown node (the output). The provision of an automatic path-constructor allows the system to be simulating generally, allowing for declarative system modeling rather than imperative approaches. In essence, the constraint hypergraph describes all possible simulations that can be conducted upon the system.

The hypergraph framework requires additional measures to correspond to real-world systems. The first major modification is the ability to handle modeling validity frames, where a constraint applies for

Submitted to:
ACT 2025

only a portion of a state variable's possible values, consequently limiting the domain of the corresponding function to a subset of a variable. These cases are handled by live pathfinding of the graph, so that the discovered path is guaranteed to be valid for the simulation's input values. The limitation of this ability is that graphs cannot be pre-solved for various input-combinations, since the existence of a valid path depends upon the input values provided at run time.

The second measure is the ability to handle cycles. Most, if not all [16], real systems are causal. A cycle in a constraint hypergraph indicates that a state variable can be calculated from its own value, a violation of causality. Although this would indicate that cycles should not exist in a hypergraph, in practice most models depict reoccurring phenomena where the behavior of the system does not change over some periodic frame of reference. In imperative modeling frameworks, such as block diagrams, this is handled by updating the state of the system upon reevaluation. Constraint hypergraphs, which are declarative, maintain immutable states to maintain causality. Serialized data is handled by imposing a frame of reference upon each node, so that edges can only be composed between nodes sharing a common frame of reference. Paths that contain cycles are required to update the frame of reference at every iteration of the cycle, allowing a model to express arbitrary periodic behavior without requiring explicit reinstantiation of the variables for frame in the simulation.

These measures are implemented in an open-source Python library called ConstraintHg [18], which allows for modelers to represent any system as a constraint hypergraph then perform arbitrary simulation upon that system via an algorithm employing Breadth-First Search. The implementation handles nested and intercrossing cycles, model validity frames, and weighted edges (for optimized pathfinding). It has been demonstrated in a variety of multi-domain models, including a hybrid elevator model that combines elements of a discrete-event simulation, a continuous PID controller, and a general dynamic model. The software serves as a basis to explore avenues of general system interoperability by enabling general system representation, so that models in disparate worlds can be combined into a single, practical framework.

References

[1] H. Abelson & G.J. Sussman (1993): *Structure and Interpretation of Computer Programs*, 17. print edition. The MIT Electrical Engineering and Computer Science Series, MIT Pr. [u.a.], Cambridge.

[2] A.D. Ames (2006): *A Categorical Theory of Hybrid Systems*. PhD Dissertation UCB/EECS-2006-165, University of California at Berkeley, Berkeley, CA. Available at `http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-165.html`.

[3] J. Baez, X. Li, S. Libkind, N.D. Osgood & E. Patterson (2023): *Compositional Modeling with Stock and Flow Diagrams*. *Electron. Proc. Theor. Comput. Sci.* 380, pp. 77–96, doi:10.4204/EPTCS.380.5. arXiv:2205.08373.

[4] J.C. Baez & B.S. Pollard (2017): *A Compositional Framework for Reaction Networks*. *Rev. Math. Phys.* 29(09), p. 1750028, doi:10.1142/S0129055X17500283.

[5] K. Brown, T. Hanks & J. Fairbanks (2022): *Compositional Exploration of Combinatorial Scientific Models*, doi:10.48550/arXiv.2206.08755. arXiv:2206.08755.

[6] A. Budiardjo & D. Migliori (2021): *Digital Twin System Interoperability Framework*. Technical Report, Digital Twin Consortium. Available at `https://www.digitaltwinconsortium.org/wp-content/uploads/sites/3/2022/06/Digital-Twin-System-Interoperability-Framework-12072021.pdf`.

[7] W.R. Cook (1991): *Object-Oriented Programming versus Abstract Data Types*. In J.W. de Bakker, W.P. de Roever & G. Rozenberg, editors: *Foundations of Object-Oriented Languages*, *Lecture Notes in Computer Science* 489, Springer, pp. 151–178, doi:10.1007/BFb0019443.

[8] D.R. Dolk & J.E. Kottemann (1993): *Model Integration and a Theory of Models*. *Decision Support Systems* 9(1), pp. 51–63, doi:10.1016/0167-9236(93)90022-U.

[9] B. Fong (2016): *The Algebra of Open and Interconnected Systems*. Ph.D. thesis, arXiv, Oxford University, doi:10.48550/arXiv.1609.05382. arXiv:1609.05382.

[10] G.J. Friedman & C.T. Leondes (1969): *Constraint Theory, Part I: Fundamentals*. *IEEE Transactions on Systems Science and Cybernetics* 5(1), pp. 48–56, doi:10.1109/TSSC.1969.300244.

[11] G.J. Friedman & P. Phan (2017): *Constraint Theory*. *IFSR International Series on Systems Science and Engineering* 23, Springer International Publishing, Cham, doi:10.1007/978-3-319-54792-3.

[12] J. Hedges (2018): *Towards Compositional Game Theory*. Ph.D. thesis, Queen Mary University of London, London. Available at `http://qmro.qmul.ac.uk/xmlui/handle/123456789/23259`.

[13] J. Hughes (1989): *Why Functional Programming Matters*. *The Computer Journal* 32(2), pp. 98–107, doi:10.1093/comjnl/32.2.98.

[14] C. Human, A.H. Basson & K. Kruger (2023): *A Design Framework for a System of Digital Twins and Services*. *Computers in Industry* 144, p. 103796, doi:10.1016/j.compind.2022.103796.

[15] INCOSE (2021): *Systems Engineering Vision 2035*. Technical Report. Available at `https://www.incose.org/2021-redesign/load-test/systems-engineering-vision-2035`.

[16] E.A. Lee (2021): *Determinism*. *ACM Trans. Embed. Comput. Syst.* 20(5), pp. 38:1–38:34, doi:10.1145/3453652.

[17] S. Mac Lane (1971): *Categories for the Working Mathematician*. *Graduate Texts in Mathematics* 5, Springer-Verlag, New York.

[18] J. Morris (2024): *ConstraintHg*. Available at `https://github.com/jmorris335/ConstraintHg`.

[19] J. Morris, G. Mocko & J. Wagner (2025): *Unified System Modeling and Simulation via Constraint Hypergraphs*. *[Manuscript submitted for publication in the Journal of Computing and Information Science in Engineering]* (Special Issue on Networks and Graphs for Engineering Systems and Design).

[20] C.B. Nielsen, P.G. Larsen, J. Fitzgerald, J. Woodcock & J. Peleska (2015): *Systems of Systems Engineering: Basic Concepts, Model-Based Techniques, and Research Directions*. *ACM Comput. Surv.* 48(2), pp. 18:1–18:41, doi:10.1145/2794381.

[21] C. Paredis, A. Diaz-Calderon, R. Sinha & P. Khosla (2001): *Composable Models for Simulation-Based Design*. *EWC* 17(2), pp. 112–128, doi:10.1007/PL00007197.

[22] D.L. Parnas (1972): *On the Criteria to Be Used in Decomposing Systems into Modules*. *Commun. ACM* 15(12), pp. 1053–1058, doi:10.1145/361598.361623.

[23] E. Patterson, D.I. Spivak & D. Vagner (2021): *Wiring Diagrams as Normal Forms for Computing in Symmetric Monoidal Categories*. *Electron. Proc. Theor. Comput. Sci.* 333, pp. 49–64, doi:10.4204/EPTCS.333.4. arXiv:2101.12046.

[24] C. Reade (1989): *Elements of Functional Programming*. International Computer Science Series, Addison-Wesley, Wokingham, England ; Reading, Mass.

[25] P. Schultz, D.I. Spivak & C. Vasilakopoulou (2019): *Dynamical Systems and Sheaves*, doi:10.48550/arXiv.1609.08086. arXiv:1609.08086.

[26] D.I. Spivak (2015): *The Steady States of Coupled Dynamical Systems Compose According to Matrix Arithmetic*, doi:10.48550/arXiv.1512.00802. arXiv:1512.00802.

[27] J.C. Willems (2007): *The Behavioral Approach to Open and Interconnected Systems*. *IEEE Control Systems Magazine* 27(6), pp. 46–99, doi:10.1109/MCS.2007.906923.

[28] A.W. Wymore (1993): *Model-Based Systems Engineering: An Introduction to the Mathematical Theory of Discrete Systems and to the Tricotyledon Theory of System Design*. Systems Engineering Series, CRC Press, Boca Raton, Fla.

[29] G. Zardini, D.I. Spivak, A. Censi & E. Frazzoli (2021): *A Compositional Sheaf-Theoretic Framework for Event-Based Systems (Extended Version)*. *Electron. Proc. Theor. Comput. Sci.* 333, pp. 139–153, doi:10.4204/EPTCS.333.10. arXiv:2005.04715.

[30] M. Zenger (2004): *Programming Language Abstractions for Extensible Software Components*. Ph.D. thesis, EPFL, Lausanne, Switzerland, doi:10.5075/epfl-thesis-2930.