From Formal Specifications to Executable Simulations: A Computation-Driven Metasystem for Agent-Based Modeling

Francisco Mesas[1], Manel Taboada[1], Francisco Epelde[2], Eduardo Cabrera[3], Alvaro Wong[3], and Dolores Rexachs[3]

[1] Escuelas Universitarias Gimbernat (EUG), Computer Science School, Universitat Autonoma de Barcelona, Sant Cugat del Vallès, Barcelona, Spain
{francisco.mesas,manel.taboada}@eug.es
[2] Consultant Internal Medicine, University Hospital Parc Tauli, Universitat Autonoma de Barcelona Sabadell, Barcelona, Spain
fepelde@tauli.cat
[3] Computer Architecture and Operating System Department, Universitat Autonoma de Barcelona, Barcelona, Spain
{eduardocesar.cabrera,alvaro.wong,dolores.rexachs}@uab.cat
https://webs.uab.cat/hpc4eas/

Abstract. Agent-Based Modeling and Simulation (ABMS) has become a widely used approach for analyzing complex systems in multidisciplinary fields such as healthcare and hospital Emergency Departments (EDs). However, the adoption of this methodology is often hampered by monolithic implementations in which domain knowledge is tightly intertwined with computational logic, limiting the long-term reusability and adaptability of simulation models. Inspired by Lego®'s modularity, this paper presents a metasystem based on a modular architecture centered on an agent metagenerator. The proposed approach conceptually encapsulates the definitions of the agents in brick-style agents, decomposing each agent into six canonical blocks. These blocks are independent of the target programming language, ensuring a clear separation between conceptual specifications defined by domain experts and their computational implementation by engineers and technicians. This separation of concerns facilitates multidisciplinary collaboration by enabling experts to explicitly define agent behavior through standardized specifications. Unlike large-scale data-driven approaches, all agent decisions are explicitly defined and calibrated using a small, controlled dataset, preserving transparency and traceability between the conceptual model and the resulting computational behavior. The proposed metasystem is validated through a proof-of-concept implementation using a simplified ED case study. The results suggest that the architecture prevents re-monolithization while enhancing modularity, providing a solid foundation for reusable, traceable, and scientifically grounded ABMS.

Keywords: Agent Based Modeling · Software Architecture for Simulation · Complex Systems Simulation · Healthcare Computing · Digital Twins.

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29915-4_3

1 Introduction

Simulation allows the study of complex systems, including heterogeneous and uncertain systems, and communication between multiple entities answering "what-if" questions that would be difficult to explore. Agent-Based Modeling and Simulation (ABMS) represents actors individually, each with its own internal state and interaction capability, enabling the simulated system to capture emergent properties arising from individual interactions [3,11].

It is important to clarify that ABMS is a computation-driven simulation technique: agent behavior is derived from formally specified computational rules rather than inferred from large data-driven models. ABMS prioritizes explicit representation of mechanisms and causal interactions [7], and can work with limited data to validate system behavior.

ABMS has been successfully applied in numerous studies in various domains, including social, biophysical, industrial and in the field of health to support decision making [5,9]. In hospital Emergency Departments (EDs), the presence of diverse patients with varying needs, medical staff with different roles, shared resources, and limited decision-making time creates a complex system that can be represented using ABMS [12]. In this paper, EDs are used as a representative case study to illustrate the proposed approach.

This modeling approach allows the creation of customized environments based on the requirements of the system under study and supports the replication of real-world behavior, separating the conceptual description of the system provided by domain experts from its computational implementation while preserving their correspondence.

However, most ABMS implementations are monolithic, with agent behavior, interaction logic, and environment configuration tightly coupled within a single codebase, making it difficult to reuse and reconfigure components, as noted by several authors who also explore different modular approaches [1,8,6,2]. For example, in EDs, phlebotomy certification is required in the United States but not in Spain, where nurses perform the same task. In monolithic systems, supporting such differences requires changes across multiple interconnected agents, complicating adaptation to different regulations, workflows, and organizational structures [6,15].

The concept of a metasystem architecture and the notion of agent bricks as encapsulated, reusable components, analogous to Lego® pieces, were introduced. A methodology based on standardized specification tables was formalized to capture expert knowledge, agent behavior, and interaction logic. A new component has been incorporated into the architecture: the Metagenerator, the automated mechanism that transforms specification tables into operational agent bricks. The term "meta" denotes that this component operates at a higher level of abstraction, generating executable agents from formal specifications rather than directly encoding agent behavior.

This paper introduces the Metagenerator and describes how it completes the metasystem pipeline. First, it defines a formal six-block model for agent specification: State, Inputs, Decision Logic, Actions, Interactions, and Lifecycle, pro-

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29915-4_3

viding a complete description of any agent within the metasystem. Second, it formalizes the Metagenerator as the component that automates the transformation of specification tables into agent bricks. Finally, although the proof-of-concept is illustrated using a NetLogo-based [14] ED model, the architectural issues are not specific to this platform or domain but are common ABMS challenges.

The remainder of this article is structured as follows: Section 2 analyzes the limitations of monolithic ABMS using an ED case study and introduces a metasystem aimed at generating independent agents to achieve modularity. Section 3 presents the Metagenerator and describes its architecture, as well as the mechanisms used to prevent re-monolithization. Section 4 details the internal architecture and operation of an agent brick. Section 5 presents a simplified case implemented using the proposed metasystem architecture. Finally, Section 6 summarizes the main contributions and outlines future work.

2 From Monolithic Simulators to Modular Metasystems

This section analyzes a monolithic simulation system to motivate the need for an architectural change and the introduction of a Metagenerator. The analysis starts from a different case study of an existing ABMS. The objective is to highlight the difficulties that arise when such models must be adapted, extended, or reused in contexts different from those for which they were originally developed.

2.1 The Monolithic Simulator

Despite the satisfactory results obtained in multiple studies, it remains extremely difficult to adapt many ABMS to new contexts with different behaviors, regulations, and organizational structures [2,6]. This limitation is widely observed across ABMS models reported in the literature, many of which operate in a predominantly monolithic manner [1,8,13]. Such designs introduce significant constraints in terms of adaptability, extensibility, and reuse.

To illustrate this general limitation, we consider a case study based on a previously developed ABMS [4]. However, as in many existing ABMS implementations, adapting the model to different contexts remains costly and complex. These limitations motivate the modular approach introduced next (the metasystem of agent bricks) and the Metagenerator described in Section 3.

2.2 The Metasystem Concept

To address these limitations, a modular architectural approach is introduced, inspired by the composability of Lego® blocks. This approach defines agent-based models in terms of standardized (i.e., internally consistent and formally defined within the metasystem), self-contained components called agent bricks, whose relationships are explicitly specified through formal specification tables. Each agent brick encapsulates all the elements that define an agent: State, Inputs, Decision Logic, Actions, Interactions, and Lifecycle as detailed in Section 4.

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29915-4_3

3 The Metagenerator: Automated Agent Brick Generation

The modular architecture proposed in the previous section requires mechanisms to automate the transformation of high-level definitions into executable specifications. This is the role of the Agent Metagenerator. This work contributes an architectural and methodological advance for ABMS, rather than an algorithmic one, by defining a formally constrained pipeline that prevents implicit dependencies regardless of implementation details. This section describes how information is introduced through specification tables, transformed into independent agent bricks, and kept from re-monolithizing.

3.1 Architectural Overview

The Metagenerator bridges the conceptual and computational models by transforming domain-expert specification tables into a canonical agent description (state variables, admissible inputs, decision rules, actions, interaction channels, and lifecycle policies) before any simulator-specific code is produced. It then validates the tables, identifies dependencies, and generates independent modules materialized as agent bricks. With a communication layer for message exchange between agents (the *message hub*), these modules are assembled into a functional simulator; the *message hub* is an architectural communication abstraction introduced by the metasystem, independent of NetLogo's native communication mechanisms.

3.2 Metagenerator Pipeline

At the core of the metasystem is the Metagenerator, which defines agents in a platform-agnostic language. For example, a rule describing how a patient reacts to a triage notification is translated into an explicit decision procedure and a corresponding message-handling interface in the generated agent brick. The resulting bricks implement each agent type's formal specification while remaining platform-independent.

The construction of canonical agent bricks follows two main processing stages:

Stage 1: Table Parsing and Validation. The Metagenerator processes all the information from the specification tables and compiles it into canonical JSON for each agent type. The JSON contains the agent state (internal variables and attributes), inputs, decision logic, actions, interactions, and lifecycle. At this stage, consistency is ensured to avoid errors during the specification process.

Stage 2: Dependency Resolution. Interactions between agents create dependencies between agent bricks. The Metagenerator builds a dependency graph to capture these relationships and determine which interfaces each agent brick must expose. Dependencies are resolved at the interface level, ensuring that agent bricks remain independent and interact through well-defined communication channels,

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29915-4_3

enabling modular composition and preventing hidden coupling. Conceptually, the Metagenerator can be seen as analogous to an oracle, in the sense that it has a global view of all specifications and resolves dependencies prior to execution, while remaining external to the runtime simulation.

Once dependencies have been resolved at the conceptual level, the canonical representation is transformed into executable code through backend-specific generation stages:

Stage 3: Code Generation. For each agent type, the Metagenerator produces a module consisting of four files, following the naming conventions defined in previous work [10]. Although this step may differ for other target languages, the current implementation uses NetLogo.

Stage 4: Assembly. A Python-based assembler script collects all generated modules, resolves `__includes` directives, merges global variable declarations, and produces a single executable `main.nlogo` file. This step is necessary because NetLogo does not natively support multi-file project structures [10].

4 Agent Bricks: The Block Architecture

In ABMS, agents are represented by their states, their ability to perceive the environment, make decisions, and interact. Fig. 1 shows the architecture defined for the agent brick generated by the Agent Metagenerator. This architecture provides a formal description of the agent, allowing clean, traceable, and reproducible behavior analysis, and avoiding the black-box approach commonly found in other systems.

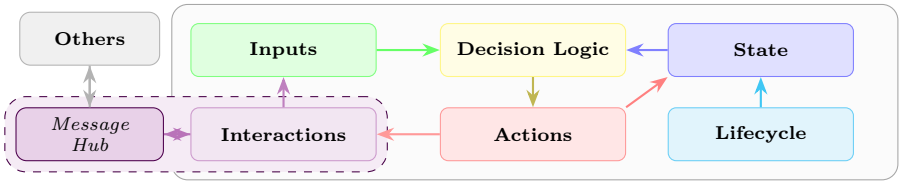

Fig. 1. Internal architecture of an Agent Brick

At the core of the agent is the **Lifecycle**, which controls when agents exist, whether they are created dynamically or remain fixed, and the conditions under which they are activated or change roles (for example, an experienced doctor reasoning differently from an inexperienced one).

Once agents are created, their **State** captures all conditions at a given time, including state variables, their attributes, and internal memory. The state reflects the sequence of events that has led to the current situation and can, in turn, trigger reactions through the decision logic.

Decision Logic contains the complete agent logic, including rules, finite-state machines, and constraints defined in the specification tables. At each simulation step, the agent determines whether to act reactively or proactively.

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29915-4_3

Once a decision is selected, its effects are realized through the actions block. **Actions** constitute the only mechanism through which an agent can affect the system and include movement, resource requests, and mediated updates to other agents. The Metagenerator generates a dedicated procedure for each declared action, ensuring that all effects on the simulation remain explicit and auditable.

Communication between agents is handled through the **Interactions block**, which specifies communication partners and channels, referred to in this article as the *message hub*. For example, in a waiting room scenario, a message broadcast through loudspeakers may be perceived by multiple agents, even if it is only relevant to one, allowing others to ignore it or react differently.

Finally, agents only have access to the information defined in the **Input** block, which determines what they can perceive and interpret. A *message hub*-based communication mechanism is used to deliver events to the agent, ensuring that it only receives input that it is allowed to observe.

Together, these six blocks define everything an agent requires to function independently. This structure also enables the creation of controlled test environments in which inputs and outputs can be simulated using an oracle that injects predefined messages. Individual agents can then be executed and validated in isolation, ensuring that they correctly process messages and make decisions, while accounting for the stochastic nature of the system.

5 Methodology and Proof-of-Concept Validation

The metasystem is validated through a proof of concept that targets the agent blocks. Accordingly, the case study should be interpreted as a structural validation of the architecture. Three aspects are analyzed: whether the model effectively supports agent specification, whether the separation between the conceptual and computational layers is maintained in practice, and whether execution can occur without hidden monolithic dependencies.

The scenario used is an ED. A simplified state machine is implemented for a patient, and all interactions are handled through message centers, enabling full traceability. Each simulation cycle represents one minute. Patient arrivals are generated from a real dataset fitted to a Poisson distribution. The simulation runs for 6400 cycles, with adjustable initial conditions. The patient transitions through four states: waiting, queuing, being seen, and discharged. State transitions occur either due to internal changes or incoming messages. The inputs include the environmental context, the current simulation time, and received messages. Interactions are intentionally kept to a minimum.

The same six-block logic is also applied to resources. The simplest resources are represented as part of the environmental state, while more complex ones, those requiring their own decision logic, are modeled as service agents. This provides a practical solution that avoids making the abstraction overly rigid.

Although code generation is intended to be automatic, the proof-of-concept implements the final step manually to validate the abstraction; artifacts and datasets will be released in future work.

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29915-4_3

6 Conclusions

The difficulty of reusing ABMS-based simulators with monolithic structures has increased over time. Based on both the literature and the accumulated experience with the ED model, a metasystem is proposed that formally separates the conceptual model from the computational model, preventing implementations that diverge from the underlying conceptual specification.

Six blocks have been introduced to describe the behavior of the agent. This structure helps to formalize expert knowledge and facilitates traceability between conceptual and computational models. All agent-independent decisions are made explicit, distinguishing this approach from others based on large volumes of data and untraceable models. The approach relies on calibration using a small, controlled dataset, positioning itself at the intersection of computation and data as a driver of scientific progress, while avoiding methods in which the decision logic cannot be traced back to the conceptual level of the model.

The Metagenerator enforces this architecture throughout the analysis process, from formal data collection in specification tables to the final simulator. It validates definitions, resolves dependencies, and generates reusable, encapsulated modules. This reduces the risk of reintroducing monolithic behavior by maintaining a clear separation between agents. This is particularly important in distributed decision domains such as EDs, where regulations can change and significantly affect agents and, consequently, the model.

The results suggest that the proposed six-block agent abstraction is sufficient to support executable simulations in the investigated scenario. At this stage, the translation of specifications into code is performed manually to avoid dependence on tooling, reinforcing the idea that each agent is an independent entity with a communication mechanism (*message hub*) and maintains a contract between the conceptual model and the simulation. Further validation across domains and platforms remains future work.

Future work involves implementing automatic generation from the specification tables of the already developed digital twin model and extending the metasystem to other platforms and simulation languages. This opens the door to create digital twins and performing new analyzes in other domains such as urban planning and public transportation systems.

Acknowledgments. This research has been supported by the Agencia Estatal de Investigación (AEI), Spain and the Fondo Europeo de Desarrollo Regional (FEDER) UE, under contract PID2023-146978OB-I00.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Abo-Hamad, W., Arisha, A.: Simulation-based framework to improve patient experience in an emergency department. European Journal of Operational Research **224**(1), 154–166 (2013). https://doi.org/10.1016/j.ejor.2012.07.028

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29915-4_3

2. Berger, U., et al.: Towards reusable building blocks for agent-based modelling and theory development. Environmental Modelling & Software **175**, 106003 (2024). https://doi.org/10.1016/j.envsoft.2024.106003
3. Bonabeau, E.: Agent-based modeling: methods and techniques for simulating human systems. Proceedings of the National Academy of Sciences of the United States of America **99**(Suppl 3), 7280–7287 (May 2002). https://doi.org/10.1073/pnas.082080899
4. Cabrera, E., Taboada, M., Iglesias, M.L., Epelde, F., Luque, E.: Optimization of healthcare emergency departments by agent-based simulation. Procedia CS **4**, 1880–1889 (12 2011). https://doi.org/10.1016/j.procs.2011.04.204
5. Comis, M., Cleophas, C., Büsing, C.: Patients, primary care, and policy: Agent-based simulation modeling for health care decision support. Health Care Management Science **24**(4), 799–826 (December 2021). https://doi.org/10.1007/s10729-021-09556-2
6. Filatova, T., et al.: Agentblocks: A community platform for sharing, comparing, and improving reusable building blocks for (agent-based) models. Journal of Artificial Societies and Social Simulation **28**(4), 11 (2025). https://doi.org/10.18564/jasss.5831
7. Gao, C., et al.: Large language models empowered agent-based modeling and simulation: a survey and perspectives. Humanities and Social Sciences Communications **11**(1), 1259 (2024). https://doi.org/10.1057/s41599-024-03611-3
8. Godfrey, T., et al.: Supporting emergency department risk mitigation with a modular and reusable agent-based simulation infrastructure. In: 2023 Winter Simulation Conference (WSC). pp. 162–173. IEEE (2023). https://doi.org/10.1109/WSC60868.2023.10407894
9. He, J., Hou, X.Y., Toloo, S., Patrick, J.R., Fitz Gerald, G.: Demand for hospital emergency departments: a conceptual understanding. World Journal of Emergency Medicine **2**(4), 253–261 (2011). https://doi.org/10.5847/wjem.j.1920-8642.2011.04.002
10. Mesas, F., Taboada, M., Rexachs, D., Epelde, F., Wong, A., Luque, E.: A customizable agent-based simulation framework for emergency departments. In: Computational Science – ICCS 2025. pp. 46–53 (2025). https://doi.org/10.1007/978-3-031-97635-3_6
11. Monks, T., Currie, C.S.M., Onggo, B.S., Robinson, S., Kunc, M., Taylor, S.J.E.: Strengthening the reporting of empirical simulation studies: Introducing the stress guidelines. Journal of Simulation **13**(1), 55–67 (2019). https://doi.org/10.1080/17477778.2018.1442155
12. Samadbeik, M., et al.: Patient flow in emergency departments: a comprehensive umbrella review of solutions and challenges across the health system. BMC Health Services Research **24**(1), 274 (2024). https://doi.org/10.1186/s12913-024-10725-6
13. Taboada, M., Cabrera, E., Luque, E., Epelde, F., Iglesias, M.L.: A decision support system for hospital emergency departments designed using agent-based modeling and simulation. 2012 IEEE 13th International Conference on Information Reuse & Integration (IRI) (2012). https://doi.org/10.1109/IRI.2012.6303032
14. Wilensky, U.: Netlogo. http://ccl.northwestern.edu/netlogo/ (1999), center for Connected Learning and Computer-Based Modeling, Northwestern University. Evanston, IL
15. Zschaler, S., Mustafee, N., Harper, A., Monks, T., Onggo, B.S., Currie, C.S.M., Polack, F.: On simulation reuse in healthcare applications. Simulation **102**(2), 149–165 (Feb 2026). https://doi.org/10.1177/00375497251383912

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29915-4_3