

Article

Inversion of Initiative in Human–Agentic AI Interaction: Procedure-Aware Workflow Control for Agent-Based Simulation Development

Henrik Schwarz , Zheng Grace Ma  and Bo Nørregaard Jørgensen * 

SDU Center for Energy Informatics, The Maersk Mc-Kinney Møller Institute, Faculty of Engineering, University of Southern Denmark, 5230 Odense, Denmark; hschw@mmmi.sdu.dk (H.S.); zma@mmmi.sdu.dk (Z.G.M.)

* Correspondence: bnj@mmmi.sdu.dk

Abstract

Agentic coding systems increasingly automate repository inspection, tool use, code generation, testing, and repair, but the human commonly remains responsible for decomposing the task, directing workflow progression, and judging completion. This limits their use in specialised domains where users understand the problem but not the development procedure required to solve it. This study introduces inversion of initiative (IoI) as a human–Agentic AI interaction paradigm in which procedural and modelling initiative shifts from the user to a procedure-aware Agentic AI, while human authority over domain meaning, consequential assumptions, validity boundaries, and progression is retained. The paradigm is instantiated for agent-based simulation development. Starting from an ordinary-language account of the problem, intended use, system, and investigative questions, the Agentic AI constructs and evaluates a structured natural-language simulation specification, resolves modelling and technical gaps through encoded skills, documentation, data, and authorised retrieval, and invokes bounded coding and assurance tools without requiring the user to orchestrate development. A browser platform operationalises IoI through an eight-stage deterministic workflow, versioned artefacts, readiness gates, traceability, semantic checks, controlled returns, and remote execution. In one end-to-end session, the platform transformed a neighbourhood energy-balancing brief containing neither an agent-based model nor a development plan into an AI-authored neighbourhood energy-balancing simulation specification, a verified configurable simulation package, and executed scenario comparison. The results support the feasibility of IoI in this specific case and illustrate an interaction pattern for agentic systems in which procedural and modelling initiative can reside with the AI while accountable domain authority remains human.

Keywords: agentic AI; inversion of initiative; human–Agentic AI interaction; human–AI collaboration; agentic coding; workflow control; bounded autonomy; agent-based simulation; procedure-aware AI; human oversight



Academic Editor: Arya Abdollahi

Received: 30 September 2026

Revised: 18 October 2026

Accepted: 22 October 2026

Published: 26 October 2026

Copyright: © 2026 by the authors.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

1. Introduction

Agent-based simulation can support domain experts who need to investigate how interacting actors, technical components, resources, and environmental processes produce system-level behaviour, including when designing and evaluating smart energy systems [1,2]. Such users may understand the real-world problem and the decisions the simulation should inform while having no training in agent-based modelling (ABM) [3].

They may not know what an agent is, which parts of a system should become agents, how agent state and behaviour are represented, or how a credible simulation study is designed and implemented.

Transforming a domain question into an executable agent-based simulation nevertheless requires a connected sequence of specialised decisions. The intended use must be translated into a system boundary and a level of abstraction. The model must identify entities, agency, state ownership, behaviour, interaction, environment, time, scheduling, inputs, outputs, assumptions, and experimental conditions. These decisions must be grounded in domain evidence or made explicit as assumptions, mapped faithfully to software, verified against the specification, and validated only for the intended use.

Large language models and agentic coding systems reduce parts of this burden. Coding agents can inspect repositories, modify files, invoke compilers and tests, interpret diagnostics, search documentation, and repeat development actions with limited intervention [4–6]. Conversational systems can support requirements elicitation and conceptual modelling [7,8], model-feature extraction [9], and simulation-code generation [10]. These capabilities make natural-language-driven simulation development increasingly feasible.

The interaction normally remains user-directed. The user asks for a model, a candidate agent, a behavioural rule, a code change, or a test and then decides what should happen next. This requires the user to recognise the simulation-development activities that are needed and to formulate technically meaningful requests. A domain expert without modelling expertise cannot be expected to supply an agent decomposition, state variables, interaction rules, update ordering, calibration plan, or validation design. A platform that merely asks the user to complete those elements has moved the expert questionnaire into a dialogue without removing the expertise barrier.

This article investigates the inversion of initiative (IoI), an interaction model in which the Agentic AI directs the simulation-development procedure, performs the modelling work, and initiates interaction when domain knowledge or human authority is required. The user begins with an ordinary-language account of the problem, intended use, relevant system, and questions. The Agentic AI then constructs the agent-based representation, acquires technical knowledge, authors the simulation specification, determines what is missing, asks domain-level questions, proposes assumptions for approval, invokes bounded coding and assurance tools, and decides whether the process should progress, repeat, or return.

The model is realised through an implemented browser platform for AgentSim. In this article, Agentic AI denotes the complete procedure-aware capability formed by the deterministic workflow controller, versioned development skills, language-model reasoning, structured natural-language specification, technical search and retrieval, artefact and traceability services, semantic validators, coding-agent runtime, build and test services, and remote execution. The workflow controller and the language-model reasoning are internal components of this one Agentic AI rather than separate systems. The workflow controller is the only component permitted to change the active stage. Language-model reasoning may create and revise the model, search for technical information, formulate questions, and recommend transitions, but it cannot directly change authoritative process state or supply human approval.

The central working artefact is a living, structured natural-language specification of the simulation. Its content is informed by established simulation conceptual-modelling and agent-based model-description literature, but it is authored by the Agentic AI rather than completed by the user. The specification connects the user's domain-level purpose to the agents, states, behaviours, interactions, time semantics, data, scenarios, outputs, assumptions, and assurance conditions required for implementation. The Agentic AI evaluates

this artefact throughout the early workflow and declares it complete enough for first code generation only when all implementation-critical gaps are resolved or explicitly bounded.

The principal contribution is the design, implementation, and functional demonstration of this inverted-initiative paradigm as one integrated development platform. The browser environment, workflow controller, natural-language specification, technical retrieval, skills, gates, AgentSim-constrained coding, local assurance loop, and remote execution facilities are cooperating mechanisms. Together they instantiate a responsibility allocation in which procedural and modelling initiative resides with the Agentic AI while human authority is retained over intended use, corrections, consequential assumptions, limitations, and progression gates.

The interaction model is functionally demonstrated through end-to-end production of a neighbourhood energy-balancing simulation. The neighbourhood energy-balancing simulation is an illustrative artefact rather than the empirical focus or a contribution to neighbourhood energy-balancing modelling. The session shows that a short domain brief can be transformed into a reviewable specification, a bounded AgentSim implementation, a verified package, and remotely executed scenario and comparison views without receiving an agent-based model design through user interaction.

The remainder of the article is organised as follows. Section 2 establishes the simulation description, conversational, agentic-coding, and execution foundations. Section 3 presents AgentSim as the controlled target platform. Section 4 describes the research design and demonstration method. Section 5 defines the inverted initiative and its authority model. Section 6 presents the AI-authored natural-language specification, iterative readiness assessment, and eight-stage procedure. Section 7 describes the implemented architecture, including technical search, bounded coding, and execution. Section 8 demonstrates the transformation of a domain brief into an illustrative AgentSim artefact. Section 9 discusses the contribution and limitations, and Section 10 concludes the article.

2. Background and Related Work

2.1. Agent-Based Simulation Development and Assurance

Simulation conceptual modelling establishes what a model is for, what it represents, what it omits, and how its outputs relate to its intended use. Robinson [11,12] defines a conceptual model as a non-software-specific description covering objectives, inputs, outputs, model content, assumptions, and simplifications. Borshchev [13] identifies practical agent-based modelling decisions concerning agents, state, behaviour, communication, space, environment, time, and outputs. These elements are not implementation details. They determine whether the executable model expresses a coherent abstraction of the problem. Model-driven simulation development further shows that a defined transformation chain from conceptual model to platform-specific implementation can carry model meaning into executable form [14]. Research on multi-agent systems in the energy domain has highlighted methodological challenges concerning systematic agent identification, the representation of agent roles and interactions, reproducible development procedures, and semantic interoperability [15]. Domain-specific research has also investigated systematic approaches for transforming descriptions of energy-business ecosystems into agent-based simulation models. A role-oriented modelling framework has demonstrated how ecosystem actors, roles, and interactions can be translated into reusable agent architectures, illustrated through an electricity distribution system with 137 domestic consumers and increasing electric-vehicle adoption [3]. This earlier framework also identifies the need for systematic verification of the resulting agent-based simulation models.

The ODD protocol addresses this need by providing a readable, platform-independent structure for describing agent-based and other simulation models through purpose, entities

and state variables, temporal and spatial scales, process overview and scheduling, design concepts, initialisation, input data, and submodels [16,17]. ODD has been used as a model-design workflow, although its primary role is consistent model description. ODD + D extends this foundation for models involving human decision-making by making objectives, adaptation, learning, decision processes, and theoretical and empirical foundations more explicit [18]. Pattern-oriented modelling adds obligations concerning target patterns and evidence [19], and TRACE adds obligations concerning testing, analysis, and transparent model development [20]. The STRESS guidelines extend comparable reporting obligations to empirical simulation studies [21].

Together, these approaches establish the substantive coverage required for a credible simulation description, but they do not imply that a novice user should complete an ODD document or answer modelling questions. In the reported platform, the literature defines obligations on the Agentic AI. The AI must transform a domain-level purpose into a structured natural-language specification that covers the conceptual model, evidence, execution semantics, experimentation, and assurance. The user can read and correct the resulting account in ordinary language without knowing the modelling concepts used to produce it. Verification then examines implementation fidelity to that specification, while validation examines adequacy for the stated intended use [22,23]. Implementation has its own canon: practical guidance on building and testing agent-based models [24], and a taxonomy of the errors and artefacts that arise in the passage from model to code, together with activities for detecting them [25].

2.2. Conversational and Language-Model-Supported Simulation Modelling

Large language models have been applied to simulation modelling as conversational assistants, model generators, and feature-extraction tools. Modeller in a Box examines LLM support across several modelling activities [7]. NetLogo Chat shows that conversational assistance can help novices and experts create and understand models, although outcomes depend on user expertise and interaction quality [10]. Siebers [8] examines conversational assistance for agent-based social simulation design, while Khatami and Frantz [9] use language models to extract candidate model features from conceptual descriptions. In adjacent formal-modelling fields, a systematic review of large-language-model approaches to generating business process models from natural-language descriptions documents growing technical capabilities but persistent concerns about semantic quality and the need for human involvement [26].

These systems show that ordinary language can mediate access to modelling and code. They can explain modelling constructs, propose candidate agents and behaviours, and translate selected requirements into executable representations. Most conversational support, however, still expects the user to select the modelling task, judge whether the model description is complete, and decide when to move from one activity to the next. Prompt-level flipped-interaction techniques already assign the questioner role to the language model [27], and mixed-initiative interaction research has long examined how initiative shifts between human and system during a task [28]. These approaches establish important precedents for AI-initiated interaction and shared initiative. The distinction investigated in this article concerns responsibility for the simulation-development procedure and agent-based model synthesis, rather than which participant initiates the next conversational turn.

The unresolved issue is therefore not only conversational accessibility. It is the responsibility of modelling. Software-engineering methods for agent-based simulation provide structured analysis, design, and implementation activities, but they presuppose a trained developer who selects and applies them [29]. Asking a novice to confirm agent types, state variables, scheduling semantics, or validation designs still assumes the expertise the

platform is intended to replace. A genuinely inverted initiative must let the Agentic AI author and evaluate those elements, acquire supporting technical information independently, and involve the user only when real-world knowledge or accountable authority cannot be supplied by technical reasoning.

2.3. Agentic Coding and Software-Development Platforms

Agentic coding systems extend code generation with repository inspection, file modification, terminal use, test execution, browser and documentation access, and iterative repair. SWE-agent demonstrates the importance of an effective agent-computer interface for autonomous software-engineering tasks [6]. The SWE-bench illustrates the difficulty of maintaining context and producing correct repository-level changes [4]. OpenHands provides a general platform for software-development agents that interact with files, terminals, browsers, and execution environments [5]. A mixed-methods study of developer use of GitHub Copilot and ChatGPT identified compatibility with existing development workflows as important for effective adoption [30]. This finding reinforces the importance of examining how AI capabilities are integrated into a development process, rather than considering their technical capabilities in isolation.

Requirements-engineering research identifies potential language-model uses for elicitation, classification, analysis, specification, and validation, while emphasising ambiguity, traceability, hallucination, and human oversight [31]. Iterative self-refinement shows how errors and critiques can inform subsequent outputs [32]. Online retrieval can similarly supply technical documentation, standards, data sheets, and scientific evidence that are not contained in the model context, provided that sources and extracted claims remain traceable and untrusted content cannot redefine the procedure.

Systems such as SWE-agent and OpenHands provide autonomous software-development capabilities, but these capabilities do not, by themselves, define a procedure for developing an agent-based simulation from a domain brief. In the neighbourhood-energy example, they could support implementing EV-charging behaviour, inspecting execution output, and repairing failing tests, but their general coding capabilities do not themselves establish which entities should be represented, whether EV-arrival timing has the intended domain meaning, which assumptions require approval, or when the model is ready for execution. In the reported platform, the procedure-aware controller and modelling skills assume responsibility for these simulation-specific decisions, delegating bounded repository work to the coding agent and seeking human input where domain knowledge or authority is required. This qualitative distinction concerns the allocation of procedural and modelling responsibility, not a measured difference in development effort, defect detection, or accuracy.

2.4. Reproducible Browser-Based Execution

Container-based development environments provide consistent software dependencies and isolate project workspaces from local user configurations [33,34]. Browser-accessible platforms such as Binder [35] and Whole Tale [36] demonstrate how computational environments can be packaged, executed, and shared with their dependencies and data. These systems primarily support accessibility and reproducibility for code and analyses that already exist.

Agentic simulation development introduces an additional requirement. The environment must allow an AI system to generate and revise code, execute build tools, inspect diagnostics, and produce a package that can be handed to a controlled simulation runner. Reproducibility becomes part of the generation, verification, and execution process rather than only a means of reproducing a completed analysis.

Protocol-based integration can reduce dependence on individual coding-agent implementations. The Agent Client Protocol, abbreviated ACP, defines a common interface for session lifecycle, streamed model output, tool calls, file differences, terminal output, and permission requests [37]. A common coding-agent protocol allows the host platform to retain responsibility for workflow state, artefact access, and tool permissions while substituting the coding agent or model used for repository operations.

2.5. Interaction Gap and Research Contribution

Established simulation methods define what credible modelling and assurance require. Model-description protocols provide readable structures. Conversational systems support selected modelling tasks. Coding agents and online retrieval provide implementation and knowledge-acquisition capabilities. Browser environments provide controlled execution. Across these approaches, the user normally remains responsible for converting a domain problem into a modelling process and deciding when the process is complete.

The unresolved interaction problem is how an Agentic AI can accept an ordinary-language domain investigation, construct the agent-based model without relying on user modelling expertise, maintain a reviewable natural-language specification, acquire technical knowledge from appropriate sources, judge readiness for first implementation, and direct the remaining development lifecycle while preserving human authority where evidence and automation are insufficient.

Table 1 positions the reported platform relative to these approaches. Its distinction is the integration of literature-informed modelling obligations, autonomous specification construction, technical retrieval, controller-governed progression, bounded AgentSim implementation, and assurance within an inverted initiative. The user contributes the problem context and authority but does not become the simulation modeller through a sequence of questions.

Table 1. Positioning of the reported platform relative to established approaches.

Approach	Primary Capability	Who Performs the Modelling?	Who Controls the Process?	End-to-End Outcome
Simulation modelling and assurance methods	Conceptual modelling, description, verification, validation, and reproducibility guidance	A trained modeller	The modeller decides when and how activities are applied	Methodological guidance and documentation
Conversational modelling support	Natural-language assistance for selected modelling and generation tasks	The user and assistant jointly, with substantial dependence on user expertise	The user selects tasks and judges completion	Partial models or code generated through dialogue
General coding agents	Repository inspection, source modification, tools, testing, retrieval, and repair	The user or host supplies the model and task	The user or host application defines the task boundary	Strong implementation capability without a simulation-development procedure
Browser-based reproducible environments	Packaged dependencies, accessible execution, and shareable computational environments	The user or an existing project	The user controls development and execution	Reproducibility of code and analyses that already exist

Table 1. Cont.

Approach	Primary Capability	Who Performs the Modelling?	Who Controls the Process?	End-to-End Outcome
Procedure-aware platform reported here	Transforms a domain brief into a natural-language specification, AgentSim implementation, assurance evidence, and execution	The Agentic AI; the user supplies only domain knowledge and authority	The Agentic AI governs progression through its workflow controller and initiates work and interaction	Integrated path from an ordinary-language question to an executable bounded simulation

3. AgentSim as the Target Simulation Platform

AgentSim is a modular Java framework for developing and executing agent-based simulations. The framework is developed in-house at the SDU Center for Energy Informatics at the University of Southern Denmark. It separates the scenario-specific representation of a system from reusable infrastructure for time progression, scheduling, interaction, observation, randomness, persistence, and execution. This separation gives the Agentic AI a bounded implementation target after it has transformed the user's domain brief into an agent-based simulation specification.

The Agentic AI does not generate an unconstrained application or recreate the simulation engine. It determines the required model and implements that model within known AgentSim abstractions and runtime semantics. AgentSim therefore narrows the implementation search space, supplies reusable execution mechanisms, and supports semantic checks that would be unavailable if the generated code could define an arbitrary runtime. Figure 1 shows this separation between the generated scenario-specific model and the reusable modelling and execution infrastructure.

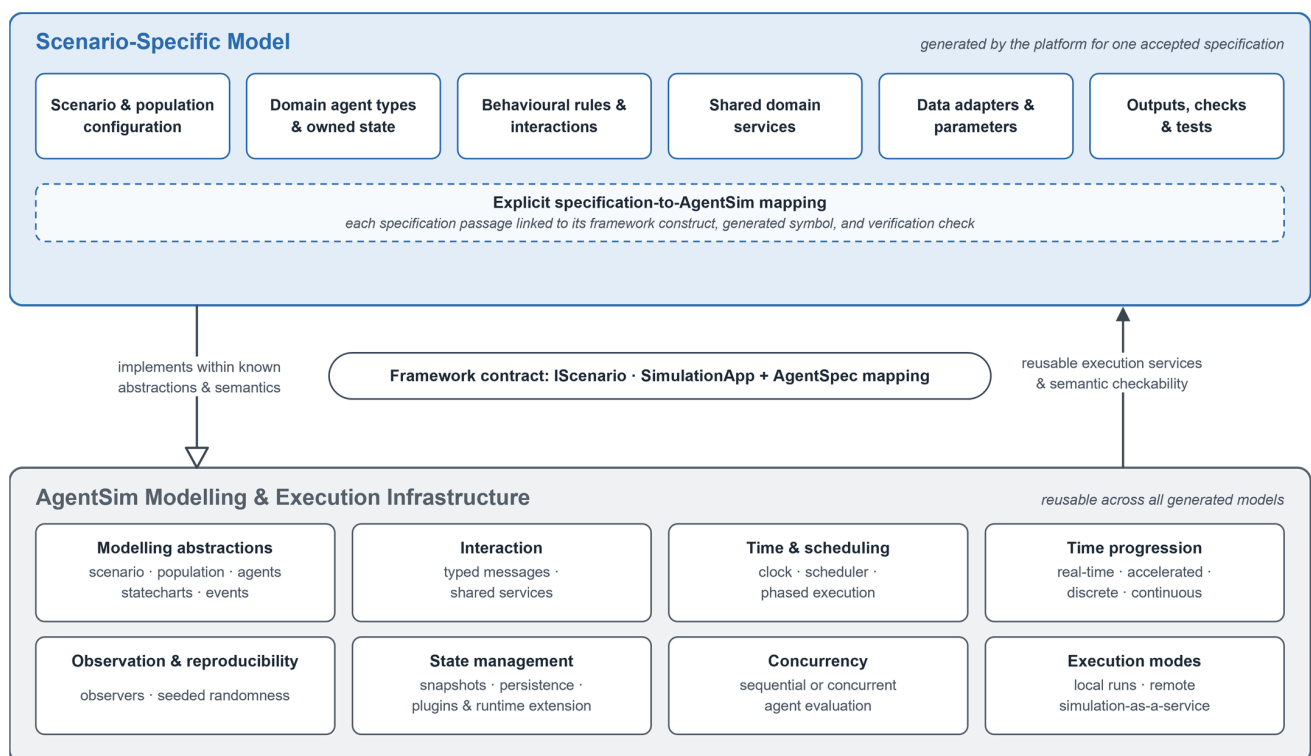


Figure 1. Conceptual separation between the scenario-specific model generated by the platform and the reusable AgentSim modelling and execution infrastructure.

3.1. Agent-Based Modelling in AgentSim

An AgentSim model represents a system as a scenario containing a population of autonomous or semi-autonomous agents. Agents encapsulate the state and behaviour of the entities represented in the model. Depending on the modelling purpose, an agent may represent an individual actor, a technical component such as a photovoltaic installation, transformer, or battery, an organisation, a control function, or an aggregate such as the household demand of a neighbourhood.

The scenario defines the model boundary and establishes the initial conditions under which agents operate. It specifies the simulated time horizon, initial population, external inputs, shared resources, model parameters, and services required by the agents. Agent behaviour can be expressed through actions triggered by simulation time, actions triggered by changes in model state, or optional statecharts with states, transitions, guards, and concurrent or hierarchical processes.

Agents may interact through typed messages or shared domain services. Message-based interaction supports decentralised communication, coordination, requests, responses, and observation of exchanged information. Shared services represent structures that are not naturally owned by a single agent, such as physical networks, markets, environmental conditions, common resources, or institutional constraints. This combination supports both autonomous local behaviour and system-level coordination.

3.2. Execution and Simulation Services

AgentSim provides the execution infrastructure that coordinates the domain model. A simulation clock controls progression of model time, while a scheduler determines when agent behaviours are evaluated and executed. The framework supports real-time, accelerated, discrete, and continuous-time progression, enabling models to represent both time-stepped processes and event-driven changes.

Execution is organised through defined phases that separate time-triggered behaviour, coordination activities, condition-triggered reactions, and publication of resulting state. These phases make the ordering of observations, decisions, interactions, and state changes explicit. Models may execute agents sequentially where deterministic ordering is required or concurrently where agent independence and computational throughput permit parallel execution.

The framework also provides services for observation, seed-controlled randomness, snapshots, plugins, persistence, and runtime extension. Together, these capabilities make an AgentSim model an executable and observable simulation artefact whose temporal behaviour, interactions, state evolution, and outputs are managed through a common runtime.

3.3. AgentSim as the Transformation Target and Semantic Boundary

Within the platform, AgentSim is the controlled transformation target between the AI-authored structured natural-language specification and the executable model. The specification explains in ordinary language what the simulation represents and how it behaves. The implementation skills then translate its statements into AgentSim constructs while retaining relations between each specification element, generated symbol, and verification check.

The generated implementation contains the scenario-specific model, including configuration, agent types, behavioural rules, interactions, shared services, data adapters, parameters, outputs, and tests. AgentSim retains responsibility for simulation time, scheduling, lifecycle management, communication infrastructure, agent discovery, observation, randomness, persistence, and runtime extension. Fixed code-defined models can be exposed through IScenario, while configurable models can implement SimulationApp and

map platform-supplied AgentSpec records to domain agents. The user is not exposed to these choices. The Agentic AI selects them from the accepted specification and the target-platform contract.

Because the framework boundary and execution semantics are known in advance, the platform can perform AgentSim-specific checks concerning behavioural ordering, interaction timing, concurrency, reproducibility, observability, persistence, and packaging. AgentSim therefore functions both as the execution framework for the generated artefact and as the semantic boundary that makes constrained code generation possible.

4. Research Design and Demonstration Method

4.1. Research Design and Evaluation Logic

The study follows an artefact-centred design science approach in which a purposeful software artefact is designed, implemented, operated, and evaluated in relation to a defined problem [38,39]. Problem identification and objectives are established in Sections 1 and 2, design and development are reported in Sections 5–7, demonstration is reported in Section 8, and evaluation applies the specified functional criteria of Section 4.6 to the demonstration evidence. The research artefact is the inverted-initiative model and the complete browser platform that realises it. The demonstration evaluates the platform as the instantiation of this model. Functional feasibility under the demonstrated conditions is evaluated through implementation and operation of a working instantiation against these criteria, while claims beyond the demonstrated scope remain analytical and are examined in Section 9. The platform realises the Agentic AI through the persisted workflow controller, and development skills, the evolving structured natural-language specification, versioned artefact and traceability services, online technical retrieval, the bounded coding-agent runtime, local build and verification services, AgentSim validators, and remote execution, and combines it with the browser interaction environment.

The evaluation asks whether, in one selected session, the implemented system can transform a domain-level investigative need into an executable agent-based simulation while the user interaction remains outside the modelling and coding procedure. The unit of analysis is the versioned interaction session. It includes the initial domain brief, every AI-initiated question or approval request, autonomous search and modelling action, specification revision, readiness assessment, skill invocation, repository change, check, diagnostic, return, and execution event.

A neighbourhood energy-balancing simulation is produced during the session as an illustrative artefact. The domain model is not the unit of analysis, and its energy-system outputs are not treated as the scholarly contribution. The artefact provides a non-trivial development task through which the interaction model can be observed from the first domain description to a remotely executable AgentSim package with a dashboard view.

4.2. Research Questions

RQ1. How can an inverted-initiative interaction be operationalised so that a user can initiate and participate in agent-based simulation development through domain-level input and authority rather than agent-based modelling instructions?

RQ2. How can a procedure-aware Agentic AI incrementally construct, evaluate, and determine the generation readiness of a structured natural-language simulation specification?

RQ3. How can the Agentic AI autonomously obtain technical knowledge, make modelling and implementation decisions, and direct workflow progression while preserving human authority over intended use and consequential assumptions?

RQ4. How can the interaction model be operationalised and functionally demonstrated through production, verification, and execution of a complete AgentSim artefact?

4.3. System Under Evaluation and Evaluation Boundary

The evaluated system begins when a user creates a session and describes a domain problem in ordinary language. No development plan, agent catalogue, behavioural model, simulation specification, or coding instruction is supplied. The Agentic AI activates Stage 1 through its workflow controller and formulates the first domain-level interaction. The evaluation ends when the AI-authored specification has passed its readiness gate, the workflow has completed all required stages, a verified AgentSim package has been executed, and no critical diagnostic or stale required artefact remains.

The boundary includes the Agentic AI with its workflow controller, skill engine, language-model reasoning, technical search and retrieval, specification construction, coding-agent actions through ACP, the workspace container, AgentSim code generation, Maven compilation and testing, semantic validators, package inspection, remote execution, artefact versioning, and traceability. It excludes comparative claims about productivity, usability, or superiority. It also excludes claims that the illustrative neighbourhood energy-balancing simulation produces empirically validated knowledge about a real district. Table 2 summarises the components and their responsibility boundaries in the demonstration.

Table 2. Components and responsibility boundaries in the demonstration.

Component or Role	Responsibility in the Demonstration	Authority Boundary
Workflow controller	Maintains process state, activates stages and skills, evaluates gates, and applies transitions.	No other component may change the active stage or mark completion.
Development and modelling skills	Interpret the domain brief, construct and evaluate the natural-language specification, acquire technical knowledge, initiate domain questions, and produce assurance evidence.	May make supported modelling decisions but cannot supply human approval or apply transitions.
Technical retrieval	Inspects documentation, data, standards, scientific literature, and online technical sources required for model and component design.	Retrieved content is evidence, not instruction or authority.
Coding-agent runtime	Implements the accepted specification in the authorised AgentSim workspace and repairs meaning-preserving technical defects.	Cannot change the specification, intended use, or workflow state.
User or domain expert	States the problem, intended use, relevant scope, and questions; corrects interpretation; supplies inaccessible local facts; approves assumptions, limitations, and gates.	Is not expected to design the ABM, select methods, formulate tests, or issue coding tasks.
Automated validators and execution services	Check specification consistency, AgentSim semantics, builds, tests, packages, outputs, and run completion.	Produce evidence and diagnostics; the controller determines the transition.

4.4. Demonstration Protocol

The demonstration uses a small, low-voltage electricity district because a short domain question can expand into several non-obvious modelling and engineering requirements. The produced artefact needs demand and generation processes, stateful storage, a capacity-constrained upstream connection, coordination, explicit sequencing, physical invariants, time series, and configurable scenarios. The case therefore provides a non-trivial task against which the functional feasibility of the responsibility allocation can be assessed.

Before the run, the controller version, skill-package version, language-model configuration, coding-agent runtime, container image, AgentSim version, Java and Maven versions, input-data manifests, and remote-runner version are frozen in the session manifest. The run starts from an empty workflow state and a clean AgentSim starter workspace. The controller continues until all eight stages are completed or the workflow reaches a blocked state that cannot be resolved through the permitted interactions and tools.

The protocol prohibits unrecorded manual intervention. After the initial brief, user input is limited to domain information, corrections, assumption or limitation acceptance, and gate authorisation. Such input normally follows Agentic AI requests; the user may also initiate a correction when inspected artefacts or results conflict with intended domain meaning. The Agentic AI remains responsible for modelling, retrieval, specification, technical diagnosis, implementation instructions, tests, and progression recommendations. Repository changes enter through bounded coding operations or recorded corrective patches; failed checks, returns, and superseded artefacts remain in the session record. Table 3 summarises the protocol steps and retained evidence.

Table 3. Demonstration protocol and retained evidence.

Protocol Step	Procedure	Retained Evidence
1. Session initialisation	Freeze system versions; create an empty workflow and workspace.	Session manifest, initial state, workspace identity.
2. Domain brief	Obtain only the problem, intended use, relevant boundary, questions, corrections, and authority available from the user.	Domain-language questions, responses, interpretation, corrections, and approvals.
3. Specification construction	Let the Agentic AI synthesise the ABM, acquire technical information, and iteratively construct and evaluate the structured natural-language specification.	Specification versions, search records, sources, assumptions, validators, and readiness assessments.
4. AgentSim implementation	Derive the mapping and invoke bounded AgentSim coding skills.	Mappings, ACP events, file differences, source symbols, and builds.
5. Verification and recovery	Execute specification-derived checks, classify failures, and repair or return to the responsible activity.	Tests, semantic checks, diagnostics, repairs, and return transitions.
6. Validation and experimentation	Define the validity boundary, derive configurations, and execute the accepted package.	Validation dossier, scenarios, package identity, runner events, outputs.
7. Demonstration closure	Assess specified responsibility, readiness, traceability, and execution criteria.	Functional assessment, residual limitations, and final controller state.

The user role in the demonstration was performed by the first author, a software engineer with limited knowledge of energy engineering who also developed the platform. The initial interaction did not provide an agent-based model or simulation-development plan, although the first author subsequently provided technically informed guidance during correction of the EV timing defect, as documented in Section 8.6 and Supplementary Section S6. The demonstration therefore represents a developer-conducted functional assessment rather than an independent study involving representative domain experts or novice users. The produced neighbourhood energy-balancing simulation was reviewed for domain plausibility by the last author, a professor in energy informatics and member of the research team. This review did not constitute an independent external evaluation.

4.5. Instrumentation and Evidence Capture

Instrumentation is built into the platform rather than reconstructed retrospectively. Each controller action and skill result is recorded in an append-only session log linked to the WorkflowState snapshot, together with the questions asked, the assumptions approved, and the code and checks produced.

Questions are stored with their purpose, ordinary-language wording, required domain authority, consequence, target specification passage, and blocking status. Search actions record the query, source, access date, extracted technical statement, affected specification content, and provenance. Specification revisions retain their relation to the initial brief, retrieved sources, AI reasoning, user corrections, and approvals. Coding-agent changes link to accepted specification statements and implementation mappings.

These records identify which work the Agentic AI completed autonomously, where it required user authority, and how the executable artefact emerged from the domain brief.

4.6. Measures and Acceptance Criteria

The assessment examines whether the reported session realised the intended responsibility allocation. The user interaction must not supply agents, state, behaviour, scheduling, AgentSim constructs, tests, or coding tasks. The Agentic AI must construct and assess the specification, obtain technical knowledge, expose consequential assumptions, and carry the accepted lineage into an executable artefact. Neighbourhood energy-balancing simulation outputs provide only evidence of execution and the specified observations. Table 4 states the functional criteria.

Table 4. Functional criteria for evaluating responsibility allocation and artefact completion.

Evaluation Dimension	Evidence Condition	Acceptance Criterion	Evidence Source
Domain-level entry	The session begins from an ordinary-language problem and intended use.	No user-authored development plan or simulation specification is supplied.	Initial brief and Stage 1 record
No ABM-specific user input	The recorded user input contains no instruction defining agents, state, behaviour, scheduling, tests, or framework constructs.	All ABM-specific content is produced by the Agentic AI rather than supplied through user interaction.	Question and response records
Autonomous specification construction	The AI creates and revises all required specification sections.	The specification passes readiness without delegating model design to the user.	Specification history and readiness record
Technical knowledge acquisition	Technical gaps that do not depend on unavailable local facts are resolved through skills, project or framework documentation, data, or authorised search; user-supplied domain facts are recorded separately.	Material technical content is traceable to an appropriate basis: retrieved or project evidence, an explicit user-supplied domain fact, or a bounded assumption. Technical research is not delegated to the user.	Search, source, domain-fact, evidence, and assumption records
Human domain authority	Material interpretations and assumptions are presented in domain language.	The AI does not approve consequential use or assumptions on the user's behalf.	Correction and approval records

Table 4. *Cont.*

Evaluation Dimension	Evidence Condition	Acceptance Criterion	Evidence Source
Procedural control	Every transition is applied through the controller.	No user, skill, search result, or coding-agent action directly changes the active stage.	Transition and WorkflowState history
Semantic consistency	Critical behavioural, temporal, interaction, and invariant checks pass.	No unresolved critical specification diagnostic remains before generation.	Specification validators and traceability
Bounded implementation	Source changes link to accepted mappings and permissions.	No unauthorised change of model meaning occurs.	ACP events, mappings, and diffs
Workflow recovery	Defects are classified and routed to the earliest responsible activity.	No implementation repair bypasses a model or authority gate.	Diagnostics, returns, and repeats
Executable completion	The AgentSim package passes build, discovery, and execution checks.	The package executes remotely and exposes mandatory outputs.	Build, package, runner, and interface records

The ten functional evaluation criteria listed in Table 4 were defined by the research team to assess whether the reported session realises the intended allocation of responsibilities and produces an executable simulation artefact. They operationalise the research questions presented in Section 4.2, while the criteria concerning simulation specification, verification, traceability, and execution are informed by the established simulation-development and assurance principles discussed in Section 2.1. For each criterion, Table 4 specifies an observable evidence condition, an acceptance criterion, and the corresponding evidence source. These criteria are intended to assess functional conformance within the reported demonstration rather than serve as an independently validated evaluation standard or a measure of comparative effectiveness.

4.7. Evaluation Execution and Analysis

The event log is converted into a chronology of domain input, AI modelling actions, search actions, specification versions, questions, corrections, approvals, readiness assessments, skills, checks, returns, repairs, and execution. Each assessment dimension is judged against its stated acceptance condition and linked evidence. Quantitative values are reported only where emitted directly by the build, runner, or interface.

RQ1 is answered through the content requested from the user and the modelling work completed by the Agentic AI. RQ2 is answered through the specification template, iterative gap evaluation, provenance and status records, and generation-readiness gate. RQ3 is answered through technical retrieval, autonomous decisions, approval boundaries, semantic checks, and controller transitions. RQ4 is answered through the complete artefact lineage and the resulting executable AgentSim package.

The generated neighbourhood-energy model is inspected only to confirm that the interaction produced a non-trivial and operational artefact. Inspection covers configured facilities and parameters, hourly demand and generation, EV-port occupancy and charging load, battery state of charge, grid import and export, self-sufficiency, run completion, the battery-capacity sweep, and the recorded weather and feeder sensitivity cases. These outputs are not used to claim empirical validity or an energy-system contribution.

4.8. Demonstration Integrity and Scope

Several controls reduce favourable retrospective interpretation. The controller log is append-only, failed attempts and return paths are retained, and unrecorded source changes are prohibited. Artefact, source, package, configuration, and execution identities are fixed through versioning and hashes. Any changed component creates a new execution identity.

The demonstration provides evidence of functional realisation of the inverted initiative for one complete session, one neighbourhood energy-balancing model, and one target framework. It does not establish comparative productivity, user acceptance across different populations, independence from the selected language model, reproducibility across repeated identical runs, or general performance across all classes of agent-based simulation. These questions require subsequent comparative and multi-session studies.

5. Inversion of Initiative and Integrated Platform Design

5.1. From User-Directed Prompting to Procedure-Aware Interaction

In a conventional coding-agent interaction, the user defines the task boundary and usually supplies the technical framing. The user asks for a model, class, test, repair, or explanation and then decides what should occur next. Even when the agent performs many repository operations autonomously, the user remains the implicit modeller, workflow designer, and completion judge.

The reported platform inverts this relationship at both the procedural and modelling levels. Creating a session activates the workflow controller. From an ordinary-language domain brief, the Agentic AI determines the required modelling activity, authors the evolving simulation specification, resolves gaps through skills, documentation, data, or online search, and decides whether the next action is another autonomous operation, a user question, an approval request, a tool invocation, or a transition recommendation.

The user role is deliberately narrow. The user explains the real-world problem, intended use, relevant scope, and questions; supplies local facts that cannot be obtained elsewhere; corrects the Agentic AI's interpretation; and approves consequential assumptions, limitations, and progression gates. The user is not expected to know what should become an agent, which variables an agent owns, how behaviours interact, which schedule is appropriate, how outputs should be calculated, or how AgentSim should implement the model.

The coding agent has a separate implementation role. It performs repository and terminal work authorised by the active implementation or verification skill. It can repair technical defects when the accepted specification determines the correct solution. It cannot change the intended use, modelling semantics, evidence status, or workflow stage. The resulting interaction separates domain authority, autonomous modelling and procedural initiative, deterministic transition control, and bounded repository capability.

5.2. Integrated Platform Workflow

The platform spans the browser interaction environment, technical knowledge sources, the AgentSim generation workspace, and the simulation-as-a-service execution environment. The Agentic AI coordinates activities across them through its deterministic workflow controller, skill engine, and versioned artefact repository.

A session begins with a short domain brief rather than a simulation specification. The Agentic AI activates Stage 1 through its workflow controller and interprets the purpose, intended use, system boundary, and investigative questions. It creates a provisional structured natural-language specification and repeatedly evaluates it. Each gap is assigned first to autonomous modelling, documentation inspection, data access, or online technical

search. A user question is generated only when the missing content depends on domain knowledge or authority that the system cannot legitimately supply.

When the specification is complete enough for a first implementation, the Agentic AI presents its interpretation, consequential assumptions, validity restrictions, and readiness judgement in readable form and requests approval to pass the generation gate. The controller then supplies the accepted specification and a derived AgentSim mapping plan to the coding workspace. Compilation, tests, and semantic checks provide feedback. Technical defects may be repaired locally, while model-level conflicts return to the responsible specification activity.

A package may be submitted to remote execution only after AgentSim conformance, verification, and package checks have passed. Results return to the same session and can cause the Agentic AI to refine experiments or return to an earlier stage. The environments and artefacts are cooperating parts of one inverted initiative rather than separate contributions. Figure 2 shows the integrated platform through which the Agentic AI carries the session from dialogue to controlled execution.

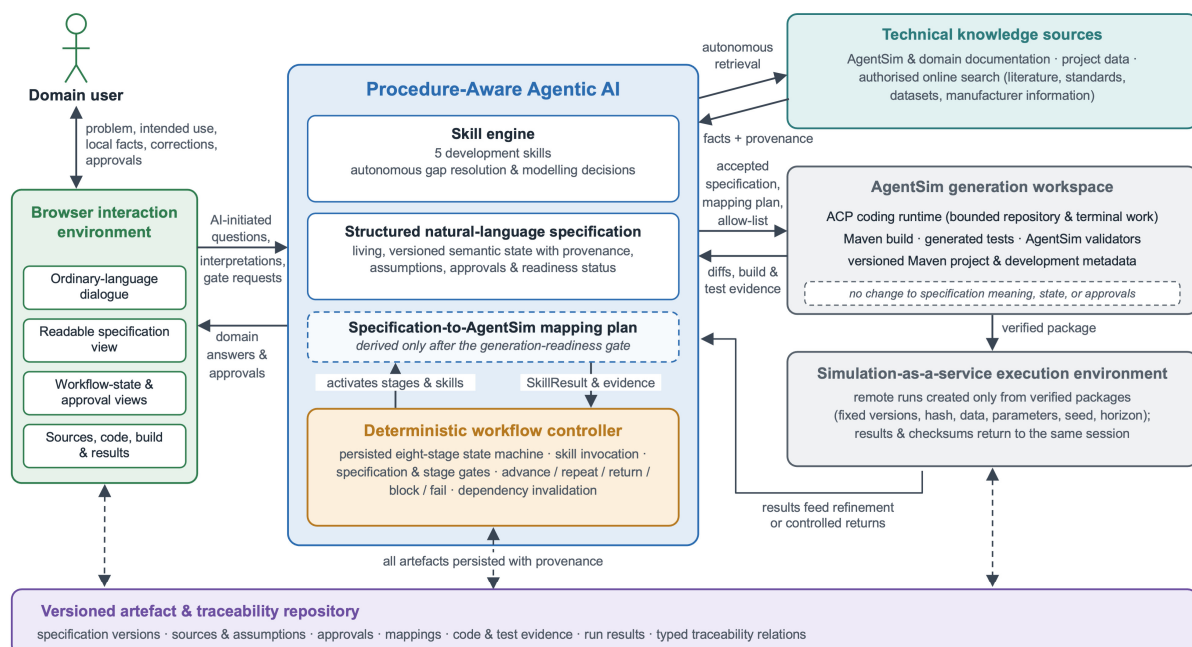


Figure 2. Integrated platform through which the procedure-aware Agentic AI initiates interaction, governs artefact formation, invokes bounded AgentSim coding, and carries the produced package into controlled execution.

5.3. Authority and Control Model

Within the Agentic AI, the deterministic workflow controller is the sole authority over process state. It activates stages, invokes skills, evaluates specification and stage gates, classifies defects, invalidates dependent artefacts, and applies advance, repeat, return, block, or fail outcomes. Language-model reasoning determines and performs the modelling work but cannot mark its own output as procedurally accepted without the controller's gate evaluation.

The Agentic AI is authorised to make modelling and technical decisions when they are supported by the accepted domain brief, evidence, target-framework constraints, or explicit low-consequence defaults. Human approval is reserved for intended-use changes, material domain interpretations, weakly supported consequential assumptions, acceptance of limitations, and release or progression gates. Approval records authority to proceed

under stated conditions but does not convert an assumption into evidence. Table 5 allocates the domain, modelling, procedural, and implementation authority.

Table 5. Allocation of domain, modelling, procedural, and implementation authority.

Decision Domain	Authority Allocation	Required Persistent Evidence
Problem, intended use, scope, and questions	The user supplies ordinary-language domain input; the Agentic AI structures and checks it.	Domain brief, AI interpretation, corrections, and approved intended use.
Agent-based model design	The Agentic AI determines agents, state, behaviours, interactions, time, scenarios, outputs, and assurance needs.	Natural-language specification passages, modelling basis, and consistency checks.
Technical knowledge and parameters	The Agentic AI uses skills, documentation, data, and online search; the user is asked only for inaccessible local facts.	Queries, sources, extracted facts, provenance, and affected passages.
Consequential assumptions and limitations	The Agentic AI proposes and explains; the user accepts, rejects, or corrects in domain terms.	Assumption, consequence, evidence gap, approval, and sensitivity treatment.
Workflow sequencing and progression	The controller activates stages and applies advance, repeat, return, block, or fail.	Workflow state, gate evidence, and transition record.
AgentSim implementation	The Agentic AI selects platform constructs; the coding runtime performs authorised changes.	Specification-to-code mapping, diff, build, and verification evidence.
Validity and supported conclusions	The Agentic AI performs technical assessment; the user supplies accountable acceptance where domain judgement is indispensable.	Validation evidence, validity boundary, approval, and traceable conclusion.

5.4. Artefact-Centred State, Gates, and Controlled Returns

Conversation is an interaction channel, while the AI-authored structured natural-language simulation specification constitutes the principal semantic state, and it remains readable ordinary English organised under stable headings.

User responses enter the authoritative state only as domain facts, corrections, preferences, limitation acceptances, or approvals linked to the affected specification passage. Retrieved technical facts retain their source and extraction context. AI-derived modelling decisions retain their basis and status. Later stages therefore depend on an inspectable specification and provenance record rather than conversational memory or an implicit prompt history.

Progression is determined by completion gates. The generation-readiness gate requires an implementable and internally coherent specification, explicit evidence and assumptions, defined outputs and checks, no unresolved critical gap, and user approval of consequential assumptions and intended-use boundaries. Return routing follows the cause of a defect. An ambiguous behaviour returns to specification construction, while an API mismatch that preserves meaning remains within implementation. Table 6 defines the controller outcomes and their transition conditions.

Table 6. Controller outcomes and transition conditions.

Outcome	Controller Action	Typical Condition
Advance	Close the active stage, persist its completion evidence, and activate the next stage.	All critical artefacts, checks, approvals, and traceability relations are complete.
Repeat	Persist new evidence and reactivate the current skill or another skill within the same stage.	The stage remains valid but needs another question, tool action, correction, or check.
Return	Identify the earliest responsible earlier stage, invalidate dependent artefacts, and activate the corrective skill.	The defect originates in an approved upstream modelling, evidence, or semantic decision.
Block	Persist the unresolved dependency and suspend progression without discarding completed evidence.	Required external evidence, authority, data, or infrastructure is unavailable.
Fail	Terminate the invocation, retain diagnostics and partial outputs, and require explicit recovery or session termination.	A non-recoverable tool, integrity, security, or controller error prevents safe continuation.

5.5. Operational Requirements

The operational requirements translate the inverted initiative into testable behaviour. They require ordinary-language entry, autonomous model construction, a readable living specification, independent technical retrieval, consequence-based human involvement, readiness assessment, bounded AgentSim implementation, assurance, controlled recovery, and an auditable development record. Table 7 states these operational requirements.

Table 7. Operational requirements of the procedure-aware platform.

ID	Operational Requirement
R1 Domain-level entry	The system shall begin with an ordinary-language problem, intended use, relevant system boundary, and investigative questions without requiring simulation terminology or a development plan.
R2 AI-owned modelling	The Agentic AI shall determine the agent-based representation, state, behaviour, interactions, time semantics, data needs, scenarios, outputs, tests, and implementation mappings.
R3 Structured natural-language specification	The authoritative model description shall be readable ordinary English organised under stable literature-informed headings and versioned throughout development.
R4 Iterative specification evaluation	After every material change, the Agentic AI shall evaluate completeness, consistency, implementability, evidence, testability, and readiness and shall identify the earliest responsible gap.
R5 Autonomous knowledge acquisition	The Agentic AI shall use encoded skills, available documentation and data, and authorised online search before requesting technical information from the user.
R6 Domain-level human interaction	Questions shall be asked only for unavailable domain knowledge, correction, material interpretation, assumption or limitation acceptance, or gate authorisation and shall avoid ABM and programming terminology.
R7 Consequence-based approval	Human approval shall be required for intended-use changes, consequential assumptions, validity acceptance, and release gates. Approval shall not convert an unsupported assumption into evidence.

Table 7. *Cont.*

ID	Operational Requirement
R8 Generation readiness	Code generation shall begin only when implementation-critical specification content is coherent, supported or explicitly bounded, testable, and free from unresolved blocking contradictions.
R9 Specification-to-AgentSim fidelity	Every generated AgentSim construct shall implement accepted specification content through an explicit mapping and shall not silently redefine the model.
R10 Bounded coding and retrieval	Search and coding tools shall operate through stage-specific permissions, recorded provenance, controlled workspaces, and explicit prohibited changes.
R11 Layered assurance	Conceptual adequacy, implementation verification, calibration, validation, sensitivity, uncertainty, and validity acceptance shall remain distinct activities with separate evidence.
R12 Responsible-stage return	Critical failures shall be classified and routed to the specification, evidence, implementation, validation, or execution activity capable of correcting their cause.
R13 Auditability	Every specification change, source, assumption, approval, check, and result shall be recorded so that the artefact and its conclusions can be inspected and re-evaluated when an upstream statement changes.
R14 Reproducible experimentation	Experiments shall fix specification, model, AgentSim, data, parameters, seeds where applicable, horizon, environment, outputs, and analysis procedure.

5.6. Design Principles and Integrated Realisation

The first design principle is domain-level entry. The platform must begin with the problem, intended use, relevant system, and investigative questions without assuming simulation terminology. The second is AI-owned modelling. Agentification, state, behaviours, interactions, time, data, experiments, and assurance are responsibilities of the Agentic AI. The third is specification-centred development. A living structured natural-language specification carries model meaning across stages and remains readable to the user.

The fourth principle is autonomous gap resolution. The Agentic AI uses encoded skills, AgentSim knowledge, available documentation and data, and online search before asking the user. The fifth is consequence-based authority. User involvement is limited to unavailable domain knowledge, correction, material interpretations, assumptions, limitations, and gates. The sixth is specification-first implementation. AgentSim code is generated only after the readiness criteria are satisfied and the user approves the first-generation gate.

The seventh principle is embedded assurance. Semantic checks, builds, tests, validation evidence, and reproducibility conditions are progression requirements. The eighth is dependency-aware recovery. A changed upstream statement invalidates dependent mappings, code, tests, and results. The ninth is development-time traceability. Questions, searches, sources, specification changes, assumptions, code, tests, and runs are connected when they occur. These principles are realised jointly through the workflow, skills, persisted controller, artefact repository, retrieval and coding tools, assurance services, and remote runner.

6. Procedure-Aware Workflow and Skill Architecture

The eight-stage workflow operationalises the inverted initiative. The stages are not a task list for the user. They encode what the Agentic AI must understand, model, research, specify, implement, check, validate, and execute. Stages 1–4 progressively construct and

assess one structured natural-language simulation specification. Stages 5–8 transform the accepted specification into an implementation and evaluated simulation artefact. Figure 3 shows the eight stages, the controller gates, and the bounded user contact points.

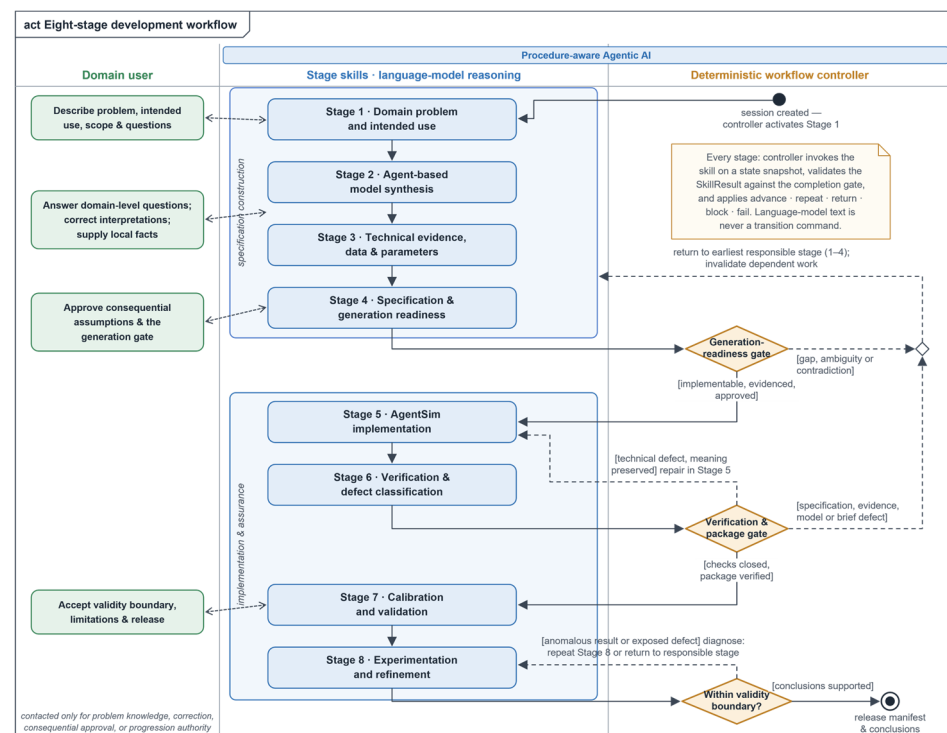


Figure 3. Eight-stage workflow in which the Agentic AI performs the modelling work and contacts the domain user only for problem knowledge, correction, consequential approval, or progression authority.

6.1. Workflow Execution Cycle

Every stage uses a controller-governed cycle. The active skill inspects the current specification and related evidence, identifies the most consequential unresolved gap or inconsistency, and selects a resolution route. It may apply encoded modelling knowledge, inspect AgentSim or domain documentation, analyse available data, perform online technical search, run a validator, formulate a domain-level question, or request approval. The resulting information creates a new specification or evidence version, after which the Agentic AI reevaluates the complete affected context.

User interaction is the final resolution route rather than the default. A question is permitted only when the answer depends on the user's real-world purpose, local system knowledge, preference, correction, or accountable authority. Questions are expressed without ABM or programming terminology. When a modelling choice is consequential but can be proposed from evidence, the Agentic AI explains the real-world effect and requests approval of its proposed interpretation instead of asking the user to design the model. Table 8 summarises the stage responsibilities, accepted evidence, and return triggers. When an accepted change requires revision—a user correction, a retrieved source, a validator or build result, or a simulation output—the controller returns to the earliest responsible specification section or related artefact and marks the dependent work for re-evaluation. Workflow closure is reached when all stage gates are accepted, no critical diagnostic remains open, and every conclusion remains within the accepted validity boundary.

Table 8. AI-directed stage responsibilities, accepted evidence, and return triggers.

Stage	Agent-Directed Work	Accepted Stage Evidence	Principal Return Triggers
1. Domain problem and intended use	Interpret the user’s ordinary-language problem, intended use, scope, and questions.	Coherent domain brief and confirmed interpretation; no ABM design required from the user.	Ambiguous purpose, materially different interpretations, or missing authority.
2. Agent-based model synthesis	Derive agents, passive entities, state, behaviours, interactions, environment, time needs, and outputs.	Provisional model content in the natural-language specification.	Missing mechanism, inconsistent abstraction, unowned state, or changed intended use.
3. Technical evidence and parameters	Resolve technical gaps through skills, documentation, data, and online search; expose residual assumptions.	Source, parameter, uncertainty, and assumption records linked to specification passages.	Unsupported material element, conflicting sources, invalid transformation, or unresolved assumption.
4. Specification and readiness	Integrate, evaluate, and present the structured natural-language specification and decide whether it is complete enough for first generation.	Accepted specification, readiness assessment, approved assumptions, limitations, and generation gate.	Ambiguity, contradiction, missing initialisation, untestable output, or blocking evidence gap.
5. AgentSim implementation	Derive mappings and invoke AgentSim coding skills within the accepted semantic boundary.	Compilable workspace, mapping record, generated tests, and package configuration.	Unfaithful mapping, platform limitation, semantic change required, or build defect.
6. Verification	Execute specification-derived checks, classify failures, and route defects.	Closed critical verification report, conformance evidence, and verified package.	Implementation, specification, evidence, data, or environment defect.
7. Calibration and validation	Determine applicability, perform technical assessment, and define the validity boundary.	Calibration record where applicable and validation dossier with supported and prohibited uses.	Leakage, failed criterion, structural mismatch, or inadequate evidence.
8. Experimentation and refinement	Derive and execute experiments, analyse outputs, and remain within the validity boundary.	Experiment manifests, outputs, analyses, conclusions, and release manifest.	Anomalous result, incomplete run, invalid output, or exposed model defect.

6.2. Stage 1: Domain Problem and Intended Use

Creation of a session automatically activates Stage 1. The Agentic AI asks the user to describe the problem or decision in ordinary language, what the simulation should help investigate, the part of the real system that appears relevant, and any intended or prohibited uses. The user may provide this information in one incomplete account. The Agentic AI reformulates it as a purpose, intended-use statement, preliminary boundary, and set of answerable simulation questions and asks only for clarification or correction where materially different interpretations remain.

Stage 1 does not ask for agents, model variables, behaviours, data structures, schedules, algorithms, tests, or software choices. It advances when the Agentic AI has a coherent domain brief that is sufficient to begin modelling, not when the user has supplied a simulation requirements specification. The domain brief remains visible as the authority source against which later modelling choices are assessed.

6.3. Stage 2: Agent-Based Model Synthesis

Stage 2 transforms the accepted domain brief into a provisional agent-based representation. The Agentic AI identifies relevant entities, passive resources, environmental processes, external drivers, and institutional or physical constraints. It determines which entities require agent representation, which state each agent owns, which behaviour changes that state, how components interact, what heterogeneity matters, and which system-level patterns and outputs answer the stated questions.

These decisions are produced by the Agentic AI using simulation-modelling skills and the intended-use boundary. If a choice depends on a domain distinction that cannot be inferred, the system asks about the real-world situation rather than about modelling constructs. For example, it may ask whether a device can act independently or is centrally controlled, then decide how that fact should be represented. The provisional model becomes new content in the evolving natural-language specification and is evaluated for omissions and contradictions.

6.4. Stage 3: Technical Evidence, Data, and Parameter Acquisition

Stage 3 obtains the technical knowledge required to make the provisional model implementable. The Agentic AI traverses the specification and creates evidence obligations for behaviours, parameters, distributions, capacities, initial conditions, external inputs, physical constraints, and validation patterns. It first uses available project material and AgentSim documentation, then searches authorised online sources such as scientific literature, standards, technical documentation, public datasets, and manufacturer information when additional knowledge is required.

Each retrieved fact retains its source, access date, extracted meaning, units, applicability, and affected specification passage. Conflicting sources and uncertain values are represented explicitly. If adequate evidence cannot be found, the Agentic AI proposes a bounded assumption, explains its consequence in domain language, and requests approval only when the assumption materially affects intended use or interpretation. The user is not expected to research technical parameters or formulate behavioural equations.

6.5. Stage 4: Natural-Language Simulation Specification and Generation Readiness

Stage 4 integrates the preceding work into the authoritative simulation specification. The artefact is normal English organised under stable headings rather than source code, a modelling notation, or a user-completed form. It states the purpose, questions, boundary, model constituents, state, behaviours, interactions, time and scheduling, inputs, scenarios, outputs, evidence, assumptions, assurance conditions, and unresolved matters. Table 9 presents the template derived from simulation conceptual modelling, ODD, ODD + D, TRACE, verification, and validation literature. The complete template structure is provided in Supplementary Section S1.

Table 9. Literature-informed structure of the Agentic AI-authored natural-language simulation specification.

Specification Section	Content Expressed in Structured Natural Language	How the Agentic AI Establishes It
1. Purpose and intended use	Why the simulation is needed, who will use it, supported decisions, and prohibited uses.	Interprets and confirms the user's domain brief.
2. Simulation questions or use cases	The questions, comparisons, or situations the simulation must investigate.	Reformulates user aims into answerable questions and checks output feasibility.

Table 9. *Cont.*

Specification Section	Content Expressed in Structured Natural Language	How the Agentic AI Establishes It
3. System boundary and scope	Included and excluded real-world phenomena, spatial and temporal extent, abstraction, and external influences.	Derives the modelling boundary and asks only about material domain ambiguity.
4. Agents, passive entities, and environment	What is represented, which entities act autonomously, and what remains passive, shared, or external.	Performs the agent-based decomposition using modelling skills and evidence.
5. State variables and parameters	What each model element remembers, units, initial values, limits, distributions, and configurable values.	Derives data needs and obtains technical values from sources or explicit assumptions.
6. Behaviours and decision rules	What each agent does, observes, decides, and changes under specified conditions.	Synthesises mechanisms from the intended use, evidence, and modelling patterns.
7. Interactions and shared resources	How model elements communicate, coordinate, exchange resources, or affect common structures.	Determines interaction semantics and consistency constraints.
8. Time, processes, and execution order	Time resolution, triggers, process order, simultaneity, state visibility, and termination.	Selects implementable temporal semantics and resolves order dependence.
9. Initial conditions and input data	Starting populations and states, external data, transformations, missing-data treatment, and provenance.	Inspects data and sources; records synthetic or assumed inputs explicitly.
10. Scenarios and experimental configurations	Factors, alternatives, parameter changes, replications, seeds, horizons, and comparisons.	Derives experiments from the simulation questions and validity boundary.
11. Outputs and observations	Required time series, indicators, comparisons, visualisations, and their derivations.	Defines outputs that answer the questions and can be verified.
12. Evidence, assumptions, and uncertainty	Basis for mechanisms and values, unresolved uncertainty, simplifications, and consequences of assumptions.	Traces sources and requests approval only for consequential weakly supported choices.
13. Verification, validation, and acceptance	Implementation checks, invariants, validation patterns, acceptance conditions, and supported uses.	Derives assurance obligations and identifies where domain judgement remains necessary.
14. Unresolved matters and readiness	Blocking gaps, non-blocking limitations, stale content, and the current generation-readiness judgement.	Reevaluates the complete specification after every material change.

The Agentic AI repeatedly evaluates the specification rather than waiting until Stage 4 to inspect it. Each addition is checked for completeness relative to the intended use, internal consistency, implementability, evidential support, observable outputs, testability, and consequences for other sections. Statements are marked as supported, AI-derived, user-corrected, approved assumptions, unresolved blocking matters, or non-blocking limitations. A contradiction or newly exposed gap returns to the activity that can resolve its cause.

The specification is declared complete enough for first generation when its purpose, boundary, questions, agent-based representation, behaviours, interactions, time semantics, inputs, outputs, and critical constraints are implementable; material technical claims have evidence or approved assumption status; no unresolved contradiction affects implementation; verification targets and an initial validity boundary are defined; and remaining uncer-

tainty is explicit and non-blocking. The Agentic AI makes the readiness judgement and presents the readable specification to the user. The controller activates Stage 5 only after the user approves the consequential assumptions, stated limitations, and first-generation gate.

6.6. Stage 5: AgentSim Implementation

Stage 5 derives a specification-to-AgentSim mapping plan without asking the user to make software choices. The implementation skill selects an IScenario or SimulationApp entry point, Simulation Builder settings, Agent subclasses, configuration records, mappers, scheduled or conditional events, statecharts, messages, shared services, data adapters, random sources, observability declarations, tests, and package registration. It may inspect current AgentSim documentation, implementation skills, and online technical sources when a component or dependency requires additional information.

The coding agent operates only within the controller-approved mapping. It may inspect the project and APIs, apply authorised changes, compile, and execute selected tests. It may repair syntax, imports, signatures, types, package declarations, and other implementation defects when the repair preserves the specification. It may not change behaviour, ordering, state ownership, parameter meaning, randomness, invariants, outputs, or intended use merely to obtain a passing build.

6.7. Stage 6: Verification

Stage 6 tests whether the generated implementation realises the accepted natural-language specification and the target-platform contract. The verification skill derives executable checks from specification statements, constraints, mappings, and AgentSim semantics. Checks cover project structure, compilation, unit behaviour, state transitions, event ordering, reporting completeness, physical limits, time progression, accounting identities, output derivation, package integrity, application discovery, local execution, and remote-runner compatibility.

Failures are classified before repair, following the established distinction between implementation errors and artefacts [25]. Implementation defects remain in Stage 5. A missing, ambiguous, or contradictory model statement returns to specification construction in Stage 4 or to the earlier modelling or evidence activity responsible for it. The Agentic AI initiates a user request when correction changes intended use, requires unavailable domain knowledge, or introduces a consequential assumption. The user may also initiate a domain-level correction when inspected behaviour or artefacts conflict with the intended real-world meaning, but the user is not required to diagnose the software cause. Table 10 maps each diagnostic class to its controller return target.

Table 10. Diagnostic classes and controller return targets.

Diagnostic Class	Controller Return	Typical Evidence and Required Resolution
Implementation defect	Stage 5	Compiler error, incorrect API use, mapping defect, invalid package, or code that diverges from an accepted specification statement. Repair and rerun affected regression checks.
Specification defect	Stage 4	Ambiguous ordering, incomplete behaviour, invalid interaction, untestable constraint, or inconsistency between specification sections. Revise the natural-language specification and reevaluate readiness.

Table 10. *Cont.*

Diagnostic Class	Controller Return	Typical Evidence and Required Resolution
Evidence or parameter defect	Stage 3	Missing source, invalid transformation, unsupported parameter, inconsistent unit, or uncertainty not represented. Obtain stronger evidence or revise its treatment.
Model-synthesis defect	Stage 2	Missing entity, incorrect agency or state ownership, absent mechanism, inconsistent scale, or output the model cannot produce. Revise the provisional model.
Domain-brief defect	Stage 1	Changed intended use, incomplete boundary, contradictory question, or unavailable authority. Reinterpret the domain brief and obtain correction where necessary.
Environment defect	Repeat or fail	Unavailable dependency, container failure, runner failure, corrupted workspace, or external service interruption. Restore the environment without changing the model.

6.8. Stage 7: Calibration and Validation

Stage 7 assesses whether the verified implementation is adequate for its intended use. The Agentic AI determines whether calibration is applicable, identifies parameters eligible for calibration, separates calibration and validation evidence, defines objective functions and acceptance conditions, and rejects unintended overlap. These are simulation-assurance decisions made by the system, not tasks delegated to the user.

Validation combines machine-checkable criteria, target-pattern comparison, sensitivity and uncertainty analysis, and domain judgement where required. The Agentic AI performs the technical analysis and asks the user only to judge a real-world interpretation or accept a restricted validity boundary when automated evidence cannot supply that authority. The dossier states supported uses, unresolved uncertainty, excluded conditions, and prohibited conclusions.

6.9. Stage 8: Experimentation and Refinement

Stage 8 derives experiment definitions from the accepted questions and validity boundary. The Agentic AI determines scenario configurations, factors, parameter sets, replications, seeds, run lengths, outputs, comparisons, visualisations, and stopping conditions. Each experiment references the verified package, accepted specification, parameter version, input data, AgentSim version, runner version, and validity conditions.

The controller submits the package, monitors execution, records failures, and verifies mandatory outputs before analysis. Unexpected results are not automatically accepted as findings. The Agentic AI first assesses whether they arise from data, experiment configuration, random variation, implementation, specification content, conceptual structure, or use outside the accepted boundary. The diagnosis determines whether Stage 8 repeats or the workflow returns to an earlier activity.

6.10. Skill Execution Architecture

A skill is an executable operation governed by a versioned manifest and invocation contract. The controller verifies its preconditions, grants only the declared artefact, search, data, repository, and execution permissions, and invokes the skill engine. Language-model reasoning may generate modelling content, search queries, source syntheses, questions,

mapping plans, code instructions, or analysis within that boundary, but cannot alter the active stage or grant itself additional authority. Table 11 defines the fields of the skill contract.

Table 11. Skill contract fields supporting autonomous gap resolution and controller use.

Contract Field	Implemented Meaning	Controller Use
skill_id and version	Stable identifier for the executable skill contract	Binds every action and artefact change to a defined skill version.
activation and preconditions	Required stage, specification versions, approvals, and checks	Prevents invocation before dependencies are satisfied.
read_scope and write_scope	Specification sections and related artefacts available to the skill	Enforces least-privilege artefact access.
permitted_tools	Allow-listed modelling, search, data, repository, build, test, and runner operations	Constrains autonomous actions through the gateway.
gap-resolution rules	Priority among autonomous reasoning, documentation, search, checks, user questions, and approvals	Prevents delegation of modelling work to the user.
output contract	Required natural-language content, provenance, internal metadata, and relations	Rejects incomplete or malformed results before persistence.
validators and completion gate	Checks that must pass before completion	Separates content generation from progression authority.
return mappings	Diagnostic classes and earliest responsible activities	Applies controlled returns and invalidation.
SkillResult	Status, artefacts, sources, checks, questions, approvals, diagnostics, and summary	Determines advance, repeat, return, block, or failure.

The platform contains eight development skills corresponding to the workflow stages and eleven AgentSim implementation skills covering project generation, agents and populations, events and statecharts, time and concurrency, messaging, shared services, data and randomness, observability, persistence, verification and repair, and experiment packaging. Development skills transform the domain brief into the natural-language specification and assurance evidence. Implementation skills determine how accepted specification content is realised in AgentSim. Representative skill definitions are provided in Supplementary Section S2.

7. System Architecture and Implementation

7.1. Components and Responsibility Boundaries

In the vocabulary of agentic architectures, the implementation is an orchestrator with specialised subagents. Each stage skill acts as a subagent with a typed contract and runs in its own isolated language-model context: it receives an immutable state snapshot, performs its modelling, retrieval, or assurance work, returns a SkillResult, and its context is then discarded. The coding runtime and the technical retrieval services act as further subagents with narrower permissions. The distinctive decision is that the orchestrator role is filled by the deterministic workflow controller rather than by language-model reasoning, so procedural compliance holds by construction while each subagent retains autonomy within its stage. Because every subagent context is isolated, all continuity

between invocations flows through the state snapshot and the versioned artefacts. Figure 4 shows this orchestrator and subagent arrangement.

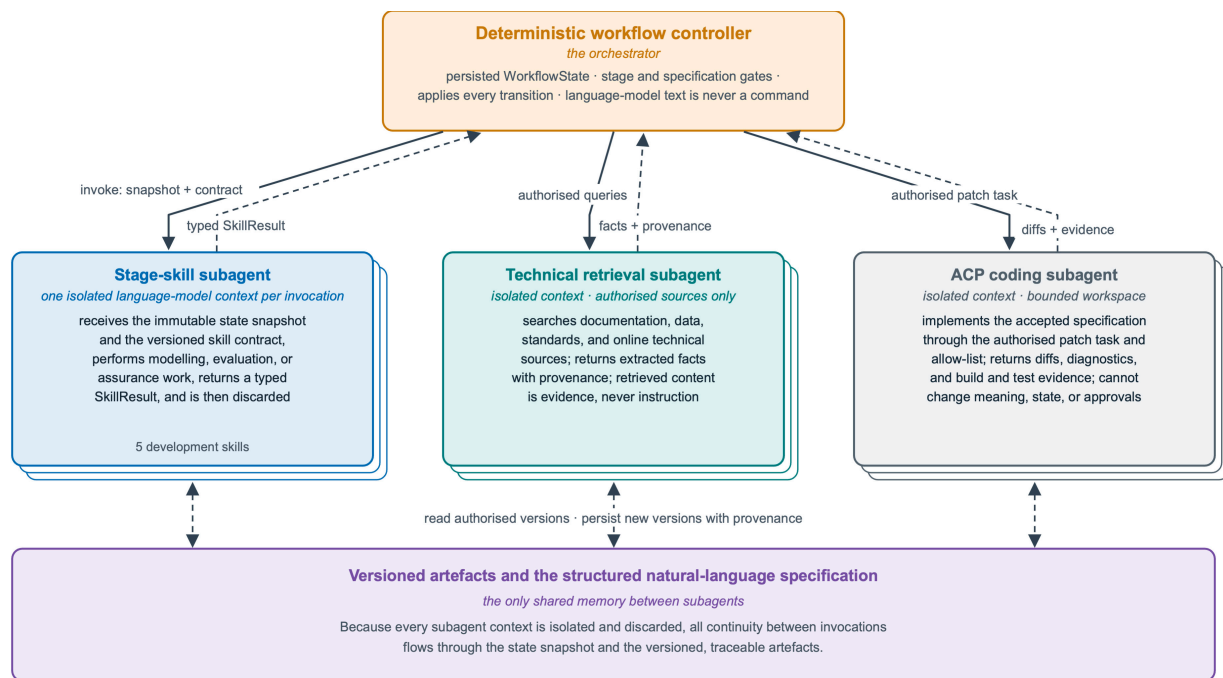


Figure 4. Orchestrator and subagent runtime architecture of the Agentic AI. Each skill invocation runs in an isolated language-model context; continuity between invocations flows through the state snapshot and the versioned artefacts.

The implementation distinguishes deterministic workflow control, modelling and retrieval skills, and language-model reasoning as internal components of the complete procedure-aware Agentic AI, alongside the natural-language specification repository, bounded coding and execution services, and the user. This distinction matters because language-model reasoning performs the modelling work, the controller alone changes workflow state, the coding runtime alone modifies the repository, and the user retains only domain and approval authority. Table 12 and Figure 5 present the implemented components and their control boundaries.

Table 12. Implemented components and control boundaries.

Component	Implemented Responsibility	Control Boundary
Browser interface	Displays domain-language dialogue, specification, sources, assumptions, approvals, code changes, build status, and results.	Presents state but does not select stages or authorise progression.
Workflow controller	Persists the state machine, invokes skills, validates results, applies gates, routes returns, and records transitions.	Only the component is permitted to change the workflow stage.
Modelling and skill engine	Interprets the brief, authors and evaluates the specification, resolves gaps, and orchestrates permitted tools.	Cannot supply human approval or directly change workflow state.
Technical retrieval	Searches local and online technical sources and records extracted facts and provenance.	Source content cannot act as instructions or grant permissions.

Table 12. Cont.

Component	Implemented Responsibility	Control Boundary
Artefact and traceability repository	Stores specification versions, sources, assumptions, approvals, mappings, diagnostics, and run evidence.	Rejects unversioned mutation and records dependency invalidation.
ACP coding runtime	Reads and edits the AgentSim workspace and returns diffs and diagnostics.	Cannot change specification meaning, workflow state, or approvals.
Execution services	Run validators, Maven builds, tests, JAR inspection, and remote simulations.	Return evidence to the active skill and controller.
User or domain expert	Supplies problem context, corrections, inaccessible facts, and consequential approvals.	Does not design the ABM or issue development tasks.

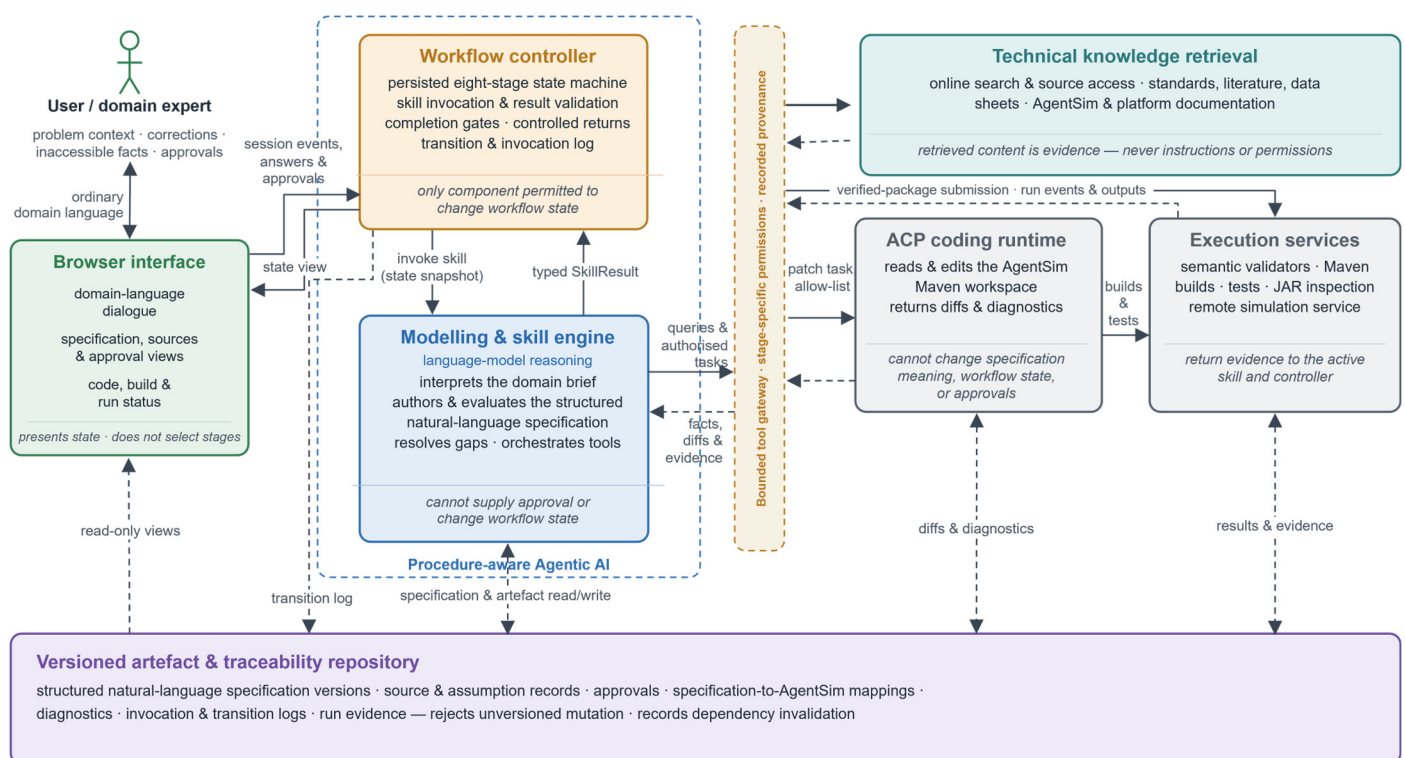


Figure 5. Implemented architecture through which a domain brief is transformed into an AI-authored simulation specification, an AgentSim implementation, and controlled execution evidence.

7.2. Browser Interaction Environment

The browser environment provides the surface for the complete session. It contains an ordinary-language dialogue panel, a readable specification view, workflow-state and approval views, source and assumption records, a file view for the generated AgentSim project, build and diagnostic views, and submission and result areas. Figure 6 shows the complete session view.

The dialogue panel presents domain-level questions, interpretations, proposed assumptions, limitations, and gate requests initiated by the Agentic AI. It may also expose autonomous search summaries, modelling decisions, coding-agent output, file changes, and terminal evidence for inspection. The user can correct meaning without operating the development tools, and the dialogue itself never substitutes for the versioned specification. Figure 7 shows the readable specification view.

The screenshot displays a web application interface for "Neighbourhood energy & EV charging". At the top, a status bar indicates the session is "RUNNING" and shows "2/8 Agent-based model synthesis". A "Stop agent" button is visible. Below this is a navigation bar with tabs for Chat, Specification, Traceability, Files, Settings, and Simulation UI.

The main interface is divided into two panels. The left panel, titled "Workflow", shows an eight-stage process:

- 1. Domain problem and intended use** (Completed): Purpose, questions, boundary drafted from user answers. Sub-tasks include extracting purpose, drafting intended-use statements, drafting system boundaries, reformulating questions, confirming interpretation, and populating spec sections 1-3.
- 2. Agent-based model synthesis** (Current stage, marked "YOU ARE HERE"): Deriving agents, state, behaviours, time, outputs (ODD). Sub-tasks include agentification (households, EVs, chargers, battery, turbine), assigning state ownership, drafting behaviours, drafting interactions, time semantics, and deriving outputs.
- 3. Technical evidence, data, and parameters**
- 4. Specification integration and generation readiness** (Gated by "Generation gate")
- 5. Specification-to-framework mapping**
- 6. Verification and defect classification** (Gated by "Package gate")
- 7. Calibration and validation**
- 8. Experimentation and refinement**

The right panel shows a chat interface with the agent. The chat history includes:

- Initial query:** "We're planning a smart neighbourhood — about 150 homes with rooftop solar, a shared battery and 8 shared EV chargers. Can it stay mostly self-sufficient through a winter month, and how big should the battery be?" (09:36 PM)
- Activities:** 11 steps (agentsim_workflow_status, agentsim_workflow_set_stage, agentsim_specification_set_section, execute) (09:36 PM)
- Response:** "The first call went through with one passage — my earlier attempts used the wrong passage shape. Writing sections 1–3 now." (09:37 PM)
- Activities:** 4 steps (agentsim_specification_set_section) (09:38 PM)
- Response:** "The parameter is passages (my earlier calls silently dropped a misnamed argument). Rewriting sections 1–3 properly." (09:38 PM)
- Activities:** 4 steps (agentsim_specification_set_section, agentsim_specification_check) (09:38 PM)
- Response:** "Stage 1 is drafted — the specification now states the purpose, the two questions, and the boundary (sections 1–3 written and checked). Before I confirm it and move to model synthesis, four domain facts would materially change the model:
 - Climate/location** — winter solar differs enormously between, say, the UK and central Europe. Which region?
 - Home size/usage** — roughly what annual electricity use per home (e.g. ~3,000–4,500 kWh, all-electric ~8,000+)? Any heat pumps or electric heating?
 - Typical PV array** — typical rooftop system size per home (e.g. 4 kWp)?
 - The 80% reading** — I've operationalised "mostly self-sufficient" as $\geq 80\%$ of monthly demand met from local solar + battery. Is 80% the right bar, or different?
 Everything else I can source or assume explicitly." (09:40 PM)
- Response:** "1. Denmark, near Odense. 2. All-electric homes with heat pumps, so winter use is high. 3. About 4 kWp per home. There is also one small shared wind turbine, around 250 kW. The tricky part is the evening — people get home around 18:00 and plug in their EVs right at the demand peak. 4. 80% is a fine stretch target, but what I really want to know is how much the battery size changes the answer." (09:40 PM)
- Response:** "Four answers noted — Denmark (Odense), all-electric homes with heat pumps, 4 kWp per home, a new shared 250 kW wind turbine, and the 18:00 EV-plugging peak being the crux. Updating the specification, then moving to model synthesis." (09:41 PM)
- Activities:** 6 steps (agentsim_specification_set_section, agentsim_workflow_set_stage, read) (09:41 PM)

At the bottom, there is a text input field for "Message the agent..." and a "Send" button.

Figure 6. Browser session view combining the ordinary-language dialogue, the eight-stage workflow with its gates, and the session state.

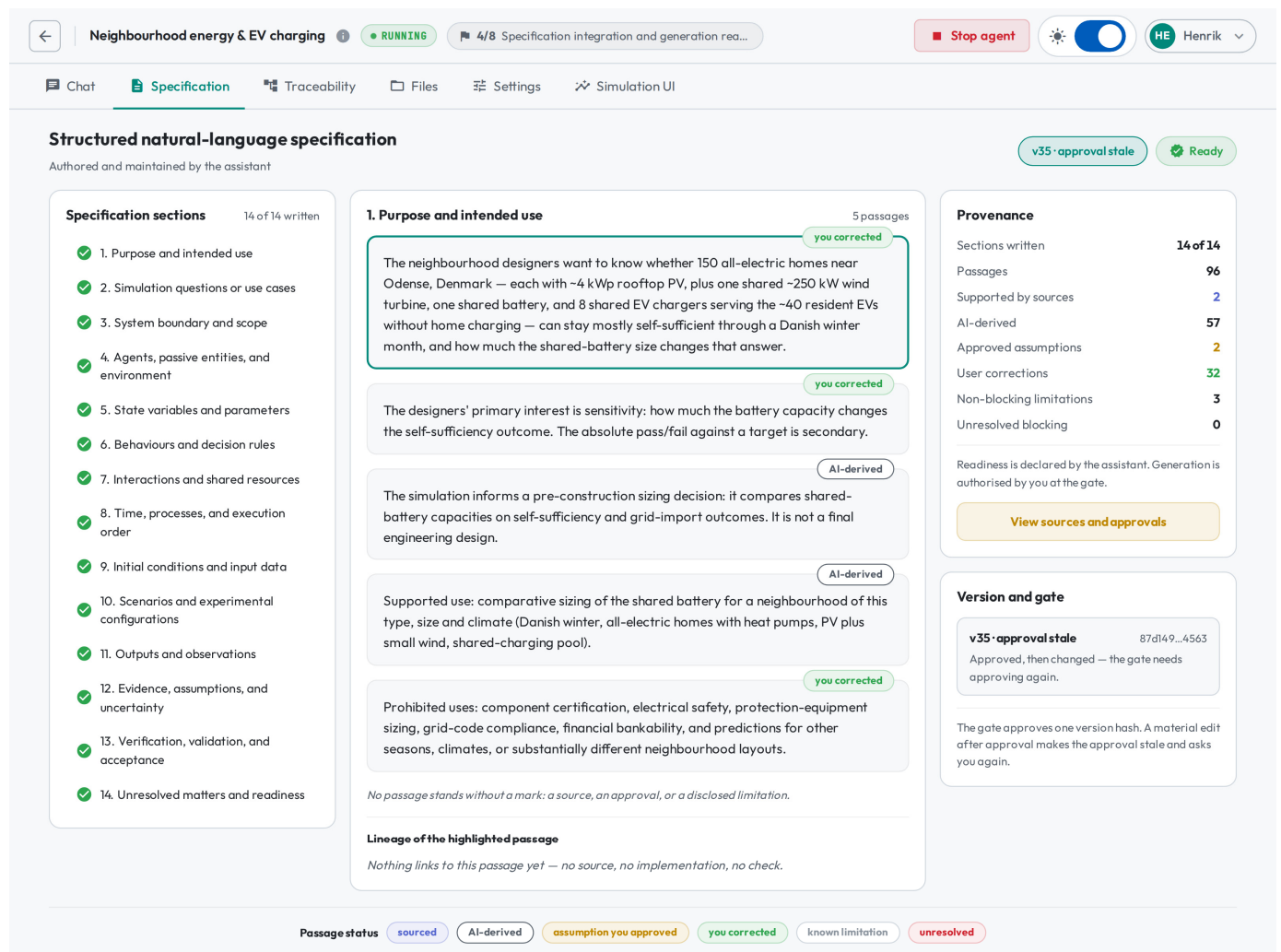


Figure 7. Readable specification view with per-passage status, provenance, and approval marks.

The workflow view identifies the active stage and skill, current specification status, unresolved gaps, pending approvals, failed checks, stale descendants, and previous returns. The user can inspect progress without selecting the next activity. The specification view separates domain input, AI-derived modelling, sourced technical content, assumptions, and approvals so that autonomous work remains reviewable.

7.3. Persisted State Machine and Deterministic Progression

The controller represents the development procedure as eight ordered stages and persists a WorkflowState for every session. The state contains the active stage and skill, immutable artefact references, completed stages, unresolved questions, pending approvals, completed checks, permitted transitions, runtime configuration, and transition history. Each skill invocation reads a snapshot of this state, and the controller persists both the invocation and result before applying a transition.

Language-model text is never interpreted as a transition command. A completed result advances only when every critical check passes and no blocking question or approval remains. A return result must identify an earlier responsible stage, after which the controller invalidates dependent artefacts. This deterministic boundary allows the Agentic AI to reason flexibly while preserving testable process authority. Table 13 defines the deterministic interpretation of each SkillResult status.

Table 13. Deterministic interpretation of skill results.

Skill Result Status	Controller Action	Persisted Effect
completed	Evaluate completion gate and activate the next stage.	Current stage marked complete; produced artefacts and gate evidence retained.
repeat	Reactivate the current skill.	New evidence retained; stage remains active.
return	Validate earlier target, invalidate dependent artefacts, and activate target stage.	Return reason and stale downstream versions recorded.
blocked	Suspend execution until the stated dependency is resolved.	Blocking condition, questions, or approval request retained.
failed	Stop the invocation and expose failure to the system operator.	Diagnostics, tool output, and unchanged workflow state are retained.

7.4. Agent-Initiated Elicitation

At activation, the skill engine evaluates the current specification and related evidence against the stage obligations. Each gap becomes one of five actions: autonomous modelling, documentation or data inspection, online technical search, automated checking, or a user request. A user request must state why domain knowledge or authority is indispensable and must use ordinary domain language. The resulting answer, correction, or approval is linked to the affected specification passage.

The human-readable specification is stored alongside the Maven project as immutable, status-annotated versions, and an upstream change marks dependent content stale until the Agentic AI reevaluates it.

7.5. Technical Retrieval, Coding Runtime, and Bounded Tool Gateway

The tool gateway exposes online search and source retrieval, file and repository access, an ACP coding runtime, terminal commands, Maven builds, test execution, package inspection, and remote submission. Permissions depend on the active skill. Search is available when the Agentic AI needs technical information to define or implement a simulation component, including scientific mechanisms, standards, component characteristics, datasets, dependencies, and AgentSim or Java APIs.

Retrieved web content is treated as evidence, never as a workflow command. Repository operations receive the accepted specification versions, mapping plan, workspace path, allow-list, success conditions, and prohibited semantic changes. Neither retrieval content nor the coding agent can change workflow state or approve a model assumption.

7.6. Natural-Language Specification to AgentSim Transformation and Local Repair

The implementation skill translates accepted natural-language statements into explicit AgentSim mappings. Each mapping identifies the specification passage, selected framework construct, target file, generated symbol, semantic constraint, implementation skill, and verification check. Model-specific mappings determine the appropriate AgentSim entry point and configuration mechanism. In the reported demonstration, the generated source implements IScenario and registers the scenario through the framework's service-provider mechanism. Experimental configurations are supplied through run parameters for battery

capacity, feeder limit, random seed, and weather variation. The coding agent then writes the authorised patch, builds the project, and returns evidence.

Diagnostics are classified before repair. API mismatches, imports, signatures, configuration conversions, and packaging defects can be corrected locally when meaning is unchanged. A validator or test that exposes an incomplete or unimplementable statement causes a controlled return to the relevant specification activity. Repair is rejected when it changes behaviour, removes a constraint, or alters a required output without a new accepted specification version. Figure 8 shows the runtime sequence of one controller-governed skill invocation.

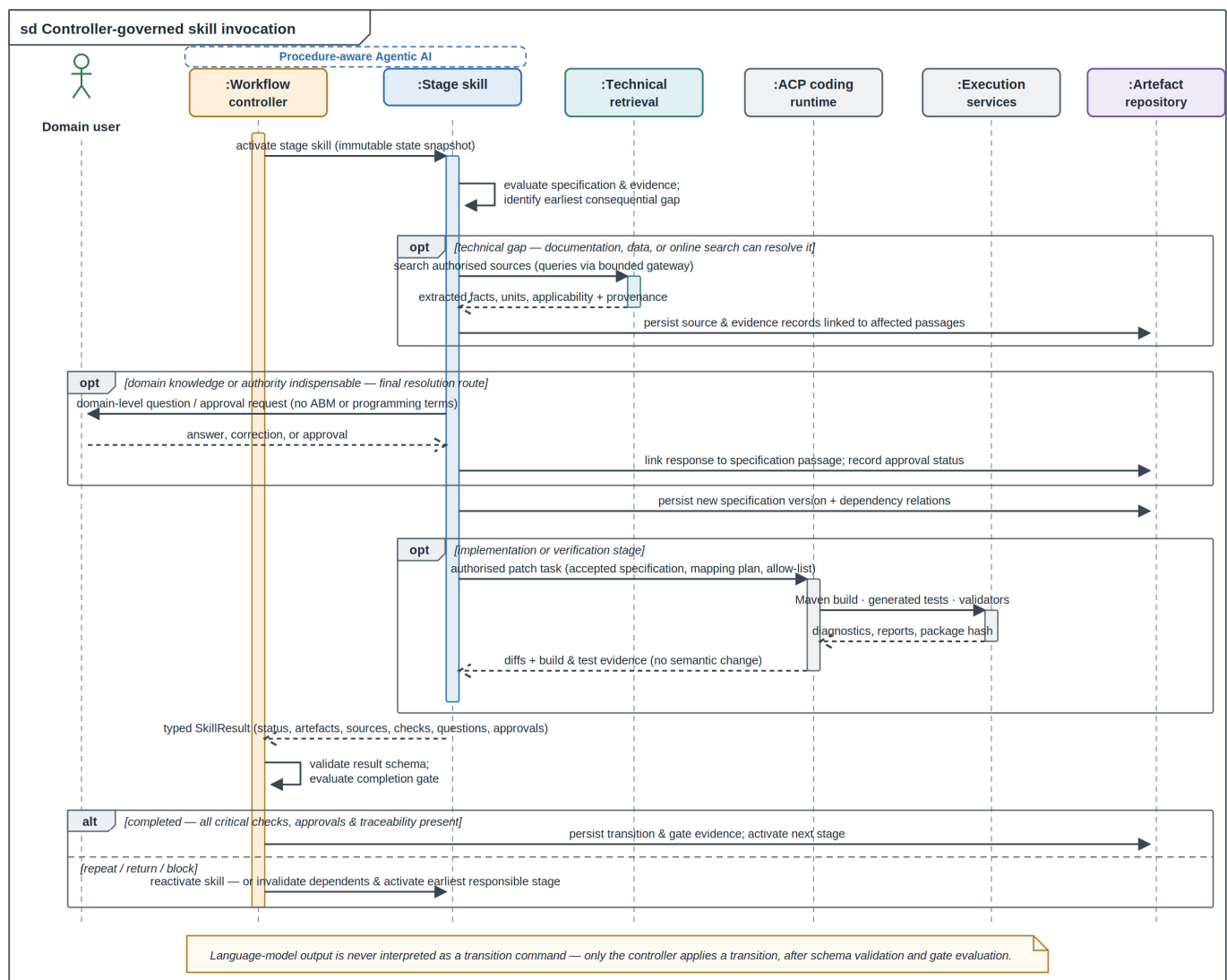


Figure 8. Runtime sequence through which the Agentic AI resolves specification gaps, requests domain input or approval only when necessary, invokes bounded repository work, and submits evidence to controller-governed progression.

7.7. Verified Packaging and Remote Execution

A remote run is created only from a verified package. The package inspector verifies the service-provider declaration and referenced classes before submission. The runner returns lifecycle events, metrics, logs, and outputs under a run identifier. Figure 9 shows the verification and packaging evidence retained in the workspace.

The screenshot displays the 'Neighbourhood energy & EV charging' workspace chat. The top bar shows the workspace name, a 'RUNNING' status, and a '6/8 Verification and defect classification' tab. The left sidebar contains a 'Workflow' section with stages 6, 7, and 8. Stage 6, 'Verification and defect classification', is the current focus. It lists several checks: Model verified post-EV-fix, Post-EV-fix: battery SoC-window invariant, Post-EV-fix: warm-up, Post-EV-fix: balance-residual sweep, Post-EV-fix: mappings and manifest convention re-checked, and Present corrected package to user. A 'Package gate: re-approval needed' button is visible. The chat area on the right shows messages from 'agentsim_workflow_status' and 'agentsim_evidence_check', indicating that the generation gate is approved and the verification pass is complete. The chat also includes a detailed explanation of the Package gate and the requirements for acceptance and commitment.

Figure 9. Verification and defect classification in the workspace chat, showing completed checks, recorded verification status, and progression towards the Package gate.

The active validation or experimentation skill evaluates returned evidence. Runtime or output failures are classified through the same diagnostic mechanism used by the local loop. The controller may repeat execution, return to implementation, return to the relevant specification activity, or request domain authority if the resolution changes intended use or a consequential assumption.

7.8. Security and Reproducibility Boundaries

Each session uses a dedicated workspace created from a controlled AgentSim starter project and a recorded container image. Network access, online retrieval, dependency installation, repository writes, and remote submission are mediated through the tool gateway. Source allow-lists, provenance checks, content isolation, and instruction filtering reduce the risk that untrusted web pages or artefacts influence workflow control. Containers support reproducibility and isolate workspace effects [33,34], but are not treated as a complete security boundary; stronger isolation may be required for untrusted generated code [40].

7.9. AgentSim Semantic Validators

AgentSim-specific validators translate framework knowledge into executable constraints. They check phase ordering, conditional quiescence, concurrency safety, time precision, event identity, message delivery, address syntax, observability, randomness, persistence, and packaging. These checks are important because a compilation-successful

implementation may still violate the execution semantics of the target simulation framework. Table 14 lists the implemented validators.

Table 14. AgentSim semantic validators.

Concern	Implemented Validator
Execution order	Scheduled events execute before conditional events; pre-step, mid-step, and post-step activities must be assigned explicitly.
Conditional quiescence	Persistent conditional rules must not create unbounded within-step cascades.
Concurrency	Shared mutable state and handlers must be thread-safe in multi-threaded mode; deterministic cases may require single-threaded execution.
Time precision	Sub-millisecond scheduling is rejected because effective runtime precision is in milliseconds.
Event identity	Repeated occurrences require distinct event instances or a recurring event; scheduling the same object twice does not duplicate it.
Messaging	Synchronous emission, queued delivery, request-response, address scope, and wildcard behaviour are specified and tested.
Addresses	Generated address segments use valid lowercase syntax and defined wildcard semantics.
Observability	Publication phase and change-detection behaviour are explicit; mutable values require defensive copying.
Randomness	Seed source, stream policy, threading interaction, and draw reproducibility are recorded and tested.
Persistence	Serialisable actions and predicates do not capture runtime-only resources; restore semantics are verified.
Packaging	The JAR contains the required service-provider entry and all referenced implementation classes.

8. Demonstration Through an AI-Authored Neighbourhood-Energy Artefact

8.1. Demonstration Purpose and Artefact Scope

The demonstration produced a simulation of a small residential district and its shared energy assets from a domain-level brief. The chosen case is an illustrative smart-neighbourhood scenario of approximately 150 all-electric homes on one low-voltage feeder, with rooftop photovoltaic generation, a 250 kW wind turbine, a shared community battery, eight shared electric-vehicle charging ports rated at 7.4 kW each and used by about forty electric vehicles, and a feeder whose grid import is limited to approximately 400 kW. The salient scenario condition is a daily coincidence: residents return home in the early evening and plug their vehicles in at the same hour that household demand peaks, so the shared chargers and household load compete for the same generation and storage. The user supplied this domain context, the scenario conditions, and the investigation objectives in ordinary language. The Agentic AI determined how they should be represented as agents, state, behaviours, interactions, execution order, scenarios, outputs, and verification conditions; the user did not supply a simulation design.

The neighbourhood is an illustrative artefact rather than the research focus. The model was not developed to certify a real feeder's performance, size protection equipment, establish grid-code compliance, or support an investment decision. These limitations were stated explicitly during the session (Section 8.5). Its purpose is to reveal whether the platform can take responsibility for agent-based modelling and implementation while the user remains able to state the domain question, correct its interpretation, approve consequential assumptions, and inspect the resulting artefact.

The base configuration represents approximately 150 all-electric homes with heat pumps and about 4 kWp of rooftop photovoltaics per home, a 250 kW wind turbine, a shared battery, eight 7.4 kW charging ports used by approximately forty EVs, and a 400 kW import-limited feeder. The corrected experiment uses one-hour discrete resolution over a 744-h winter reporting window following a two-week warm-up, with seeded synthetic weather and demand profiles. Measured metering and weather data were not available to the session, so the intended use is restricted to structural inspection of self-sufficiency, battery-capacity effects, and the specified sensitivity cases rather than empirical prediction. Table 15 summarises the configuration and its demonstration role.

Table 15. Configuration and intended role of the illustrative neighbourhood-energy artefact.

Case Element	Configured Value	Role in the Demonstration
System boundary	Approximately 150 all-electric homes on one low-voltage feeder sharing rooftop photovoltaics, a 250 kW wind turbine, a community battery, eight 7.4 kW charging ports used by about forty EVs, and a grid connection	Provides household demand, shared generation, storage, EV charging load, and an external balancing connection.
Simulation period	A 744-h winter reporting window at one-hour discrete resolution following a two-week warm-up, with an exclusive end	Exercises warm-up, recurring hourly events, seeded runs, and execution over a monthly-scale reporting horizon.
Household demand	Aggregate hourly electrical demand for approximately 150 all-electric homes with heat pumps	Creates the residential base load and pronounced evening peak used in the structural balancing study.
Rooftop photovoltaic generation	Approximately 4 kWp per home represented through a seeded synthetic winter daylight profile	Creates zero overnight output and a daytime production window without claiming measured local irradiance.
Wind turbine	One shared 250 kW turbine with a seeded synthetic generation profile	Adds a weather-driven generation component that is independent of photovoltaic daylight production.
Shared battery	Community store with 90% usable capacity, 0.5 C power limit, 90% one-way efficiency, and capacity varied from 0 to 3000 kWh	Carries state between hours and exposes how self-sufficiency changes across the tested capacity sweep.
EV chargers	Eight shared 7.4 kW ports used by approximately forty EVs with an 18:00 arrival coincidence	Concentrate additional load at the household evening peak and create port contention when arrivals exceed availability.
Grid connection	Approximately 400 kW import-limited feeder in the base case; exports are recorded without an implemented export cap	Balances the residual; the additional 300 kW sensitivity tests the behaviour of an active import constraint.

Table 15. *Cont.*

Case Element	Configured Value	Role in the Demonstration
Input basis	Seeded synthetic winter weather and demand profiles rather than measured metering or weather data	Bounds the case to structural inspection rather than empirical prediction.
Principal outputs	Household demand, shared generation, battery state of charge, EV charging load and port use, grid import and export, self-sufficiency, and battery cycling	Shows whether the generated artefact exposes the observations required by the specification and experiment.
Comparison variants	Seven battery capacities from 0 to 3000 kWh across base-sweep seeds 42, 7, and 99, plus two stormier-wind runs and a 300 kW feeder sensitivity	Demonstrates reuse of one verified model package across corrected runs 24–47 without redefining the model; exact run-level settings are retained in the supplied manifests.

8.2. Interaction Chronology

The session began with an ordinary-language brief concerning an illustrative smart neighbourhood of about 150 homes with rooftop solar, a shared battery, and eight shared EV chargers. The two investigation questions were whether the neighbourhood could remain mostly self-sufficient through a winter reporting period and how self-sufficiency changes as shared-battery capacity is increased. In response to the Agentic AI's draft domain brief, the user supplied scenario context and conditions that were not modelling instructions: an Odense-area climate context, all-electric homes with heat pumps, approximately 4 kWp of rooftop photovoltaics per home, a 250 kW shared wind turbine, the 18:00 EV-arrival coincidence, and an 80% self-sufficiency stretch target. The user did not provide an agent catalogue, component classes, state variables, a capacity model, a scheduling order, profile equations, a test plan, or an AgentSim design. The first author served as the user and personally answered Agentic AI questions and approval requests and could also correct observable domain meaning when needed.

The Agentic AI opened by stating the responsibility split in its own words: "I'll run this as an eight-stage development workflow; you own the meaning, I own the process, and you're asked only when your knowledge or approval is needed." It then structured the domain brief and model synthesis around hourly balancing of one low-voltage district across a winter reporting period. The 80% value was retained as a user-supplied stretch target for the simulated system rather than as a software acceptance criterion. From the user-supplied domain context and scenario conditions, the Agentic AI derived the agent-based representation comprising household, shared-EV, and battery agents, together with passive wind-generation, shared-charging, and feeder components, state ownership, hourly interaction sequence, scenarios, outputs, and verification obligations. User participation supplied domain context and scenario conditions, corrected the feeder limit and shared-EV interpretation, restricted the permitted uses, inspected observable behaviour, and authorised the Generation and Package gates; it did not supply the agent-based model design or technical repair procedure. Figure 6 (Section 7.2) shows this interaction and the eight-stage rail, while Table 16 summarises the responsibility allocation.

Table 16. Responsibility allocation during the neighbourhood-energy workflow.

Stage	Case-Specific Agentic AI work	User Contribution	Principal Accepted Result
1. Domain brief	Frame the purpose, boundary, and investigation questions from the ordinary-language account without requesting ABM design.	Describe the neighbourhood, scenario context and conditions, the 18:00 EV coincidence, and the 80% stretch target; confirm or correct the interpretation.	Domain brief for structural inspection of self-sufficiency and battery-capacity effects, not operational certification or price forecasting.
2. Model synthesis	Derive the three domain-agent types and identify the passive generation, charging, and feeder components, state ownership, behaviours, interactions, hourly ordering, and required outputs from the accepted domain context and scenario conditions.	Correct only if the intended scenario interpretation is materially wrong; no agent or behaviour design is requested.	Provisional neighbourhood model in structured English.
3. Technical evidence	Resolve technical gaps through skills, AgentSim and project documentation, data, and authorised sources; preserve user-supplied domain facts and explicit assumptions as separate provenance.	Supply inaccessible domain information when asked; no technical research, parameter derivation, or modelling task is delegated to the user.	Source, domain-context, parameter, synthetic-input, and assumption records linked to the specification.
4. Specification & readiness	Integrate the complete hourly semantics, invariants, outputs, evidence, and checks and decide generation readiness.	Approve consequential assumptions and limitations and authorise the Generation gate; later material edits require renewed authority.	Complete fourteen-section specification and readiness record with version-sensitive gate state.
5. AgentSim implementation	Select the application, configuration mapping, agents, events, observability, packaging, and dashboard specification; invoke bounded coding operations.	None unless an implementation issue exposes a new domain ambiguity or consequential assumption.	Configurable AgentSim project, dashboard definition, and specification-to-code mappings.
6. Verification	Check mappings, reports, capacity and state bounds, energy accounting, timing, package discovery, and corrected EV coincidence; classify defects and route returns.	Inspect observable behaviour and request correction when the intended evening charging pattern is absent; no software diagnosis is required.	Verification record, controlled implementation return, repaired package, and Package-gate evidence.
7. Calibration & validation	Restrict assessment to invariants, qualitative patterns, sensitivity evidence, and a bounded validity statement appropriate to synthetic inputs.	Accept the restricted validity boundary.	Validation dossier excluding empirical and operational claims.
8. Experimentation	Execute the corrected seven-capacity, three-seed sweep and the recorded weather and feeder sensitivities; inspect and report results within the validity boundary.	Authorise progression and inspect the resulting comparison views.	Corrected runs 24–47, time series, dashboard comparisons, and bounded conclusions.

8.3. AI-Authored Specification and Generation Readiness

The specification surface (Figure 7, Section 7.2) shows the artefact the Agentic AI authors and maintains: a structured natural-language specification with fourteen sections covering purpose and intended use; simulation questions; system boundary and scope; agents, passive entities, and environment; state variables and parameters; behaviours and decision rules; interactions and shared resources; time, processes, and execution order; initial conditions and input data; scenarios and experimental configurations; outputs and observations; evidence, assumptions, and uncertainty; verification, validation, and acceptance; and unresolved matters and readiness. The updated live-workspace capture contains all fourteen sections across ninety-six passages.

Every passage carries an explicit status or provenance classification: sourced, AI-derived, approved assumption, user correction, known limitation, or unresolved. Specification v35 contains two sourced passages, fifty-seven AI-derived passages, two approved assumptions, thirty-two passages marked as user corrections, three non-blocking limitations, and zero unresolved blocking matters. The thirty-two user-correction passages reflect propagation of a smaller number of domain corrections through every affected part of the specification; they do not represent thirty-two independently user-designed model elements. Passage status is distinct from evidential basis: material technical content can be grounded in retrieved or project evidence, an explicit user-supplied domain fact, or a bounded assumption. Figure 7 captures v35 after material edits had made the earlier Generation-gate approval stale. The retained session subsequently records renewed user authorisation and completion of implementation, verification, calibration and validation, and experimentation. Table 17 shows a representative extract of the Agentic AI-authored specification, including passages affected by user corrections.

Table 17. Representative extract from the AI-authored neighbourhood-energy specification.

Specification Section	Representative AI-Authored Structured-English Statement	Basis and Status
Purpose, boundary, and intended use	The simulation supports structural inspection of self-sufficiency and shared-battery sizing for approximately 150 all-electric homes behind one low-voltage feeder.	AI-authored from the domain brief; the user subsequently corrected the feeder capacity and clarified the shared-EV population.
Purpose and intended use	It must not be used to certify grid-code compliance or to size protection equipment.	AI-authored limitation, user-corrected to state the excluded uses explicitly.
Simulation questions	The model examines whether the district stays mostly self-sufficient across a winter reporting period and how self-sufficiency changes with shared-battery capacity relative to a no-storage baseline.	Derived from the user's brief; interpretation confirmed.
Agents and environment	The model contains household-demand, rooftop-photovoltaic, wind-turbine, shared-battery, EV-charger, and grid-connection agents.	AI-derived modelling decision; no agent selection requested from the user.
Behaviours and decision rules	Demand, photovoltaic, and wind report first. EV arrivals occupy free ports at 7.4 kW in the evening. The battery then charges or discharges against the residual within its limits, and the grid balances the remainder.	AI-authored mechanism; consequences presented for approval.

Table 17. Cont.

Specification Section	Representative AI-Authored Structured-English Statement	Basis and Status
Time and execution order	The simulation runs at one-hour discrete steps over a representative winter month with an exclusive end.	AI-derived execution semantics; checked against AgentSim timing.
Outputs and observations	Required outputs are aggregate household demand, shared generation, battery state of charge, EV charging load, grid import and export, and self-sufficiency.	AI-defined from the questions and available illustrative inputs.
Evidence and validity	Synthetic winter profiles support qualitative structural inspection only.	AI-derived; a non-blocking limitation on the captured version.
Readiness judgement	The implementation-critical components, behaviours, ordering, parameters, outputs, and checks are specified; the specification check reports zero blocking matters.	Agentic AI readiness judgement; progression remained subject to the user-controlled Generation gate and renewed approval after material edits.

8.4. Bounded AgentSim Implementation and Verification

The accepted specification was implemented through AgentSim’s configuration-driven path: an application creates the scenario from typed configuration; typed records and an agent mapper instantiate the domain agents; the simulation builder advances the reporting horizon at one-hour resolution; published accessors expose the required series; and service registration makes the package discoverable by the runner. The Agentic AI recorded fifteen specification-to-code mappings and, through the bounded coding runtime, generated the model and dashboard, repaired implementation defects when the accepted specification determined the meaning-preserving solution, built the package, and ran structural and semantic checks. A potential repair that would alter EV-arrival semantics, charger, battery or feeder limits, ordering, or permitted claims required specification-level treatment rather than an unrecorded coding change. Table 18 states the implemented correspondence and verification targets.

Table 18. Mapping from neighbourhood-energy specification elements to AgentSim constructs and verification targets.

Specification Element	AgentSim Implementation	Verification Target
Configurable application	Application entry point with a simulation configuration and platform context	Runner entry path, required configuration, and application discovery
Simulation settings	Simulation builder with start, end, step, mode, and threading settings	One-hour discrete-time progression over the monthly horizon and exclusive end
Asset configuration	Agent specifications, typed configuration records, and an agent mapper	Known-type mapping, parameter conversion, range checks, and rejection of unknown asset types
Scheduled reporters	Household-demand, rooftop-photovoltaic, and wind-turbine agents with recurring hourly events	One report per expected participant and correct profile evaluation

Table 18. Cont.

Specification Element	AgentSim Implementation	Verification Target
EV charging	EV-charger-bank agent with an evening arrival profile and eight 7.4 kW ports	Port occupancy, per-port power bound, contention when arrivals exceed free ports
Storage	Reactive shared-battery agent with persistent state	Charge and discharge power, energy capacity, efficiency, state bounds, and state carry-over
Upstream connection	Grid-connection service with an import capacity bound and uncapped export in the current implementation	Import sign and cap, export sign and uncapped-export behaviour, and residual and curtailment calculations
Observability	Public published accessors on the domain agents	Required time-series presence and values emitted after the accepted hourly sequence
Packaging	Shaded application JAR and service-provider entry	Referenced-class integrity and remote-runner discovery

8.5. Controlled Returns and Domain-Authority Corrections

The case shows why an AI-owned specification with user-held domain authority is necessary. The user clarified three consequential aspects of meaning: the feeder is limited to approximately 400 kW rather than being an unlimited external supply; approximately forty resident EVs rely on the eight shared 7.4 kW charging ports; and the model must not be used for protection-equipment sizing, grid-code certification, or comparable operational claims. These corrections were recorded in the specification rather than silently absorbed. None required the user to select agents, edit code, or diagnose the implementation. The controller retained semantic decisions, including EV-arrival ordering, port contention, battery balancing, and feeder settlement, in the specification and verification record. Table 19 records the principal decisions and corrections.

Table 19. Case-specific semantic decisions, user corrections, and responsible stages.

Issue or Decision	Responsible Stage	Treatment	Retained Record or Boundary
Feeder and shared-EV scope	Stage 4 (user authority)	The user corrected the external connection to an approximately 400 kW import-limited feeder and clarified that approximately forty resident EVs rely on the eight shared ports.	Recorded as user corrections in the boundary, parameter, interaction, and acceptance passages.
Permitted and excluded uses	Stage 4 (user authority)	The user added that the model must not certify grid-code compliance or size protection equipment.	Recorded as a user correction; a stated non-goal of the artefact.
Whether EV arrivals precede or follow battery balancing	Stage 4	Treat the coordination order as a domain-semantic decision, not a coding default.	EV arrivals occupy free ports first; the battery then absorbs or supplies the residual.
Port contention at the evening peak	Stages 3 and 4	State the eight-port limit and the 7.4 kW per-port bound explicitly.	Arrivals beyond free ports wait; charger power is bounded.

Table 19. Cont.

Issue or Decision	Responsible Stage	Treatment	Retained Record or Boundary
Battery as an hourly reporter	Stage 4	Separate scheduled contributors from the reactive balancing participant.	The battery is excluded from the reporting count and acts only after coordination.
Self-sufficiency accounting	Stage 4	Define on-site-met demand versus grid import so the reported ratio is reproducible.	Self-sufficiency is derived from published series, not asserted.

The question of whether EV arrivals precede battery balancing, recorded in Table 19, is a simulation-design decision assigned to the Agentic AI, not a choice that the domain user is expected to make. The domain-level information supplied by the user was that approximately forty EVs share eight chargers and that evening charging coincides with the household demand peak. The Agentic AI is responsible for translating those conditions into a coherent execution order and presenting the real-world consequences of any consequential assumption for approval. The technically informed developer intervention during correction of the subsequent timing defect is reported separately in Sections 4.4 and 8.6.

8.6. The Agentic AI-Authored Simulation Dashboard

The Simulation UI (Figure 10) renders the neighbourhood dashboard that the platform serves from the workspace, reading the runs recorded by the generated model through the platform's Data API. Its header states the configuration, “~150 homes · rooftop PV · 250 kW wind turbine · shared battery · 8 shared EV chargers · winter month, hourly”, and it exposes the observations required by the specification: KPI tiles for self-sufficiency, peak grid import against the feeder limit, the share of EV charging served locally, and battery full cycles; a neighbourhood panel with the power balance, EV charging allocation, battery state of charge, and the eight charger ports; a battery-capacity sweep across seeds; and Grid import, Net load, and Battery series tabs that compare the no-battery, 1000 kWh, and 3000 kWh runs. The captured frame shows the busiest hour at 18:00 on 2 February. The panel reports “Net load 375 kW · grid import 223 kW of 400 kW · EV load 52 kW · 8/8 chargers in use”. This frame exposes the domain tension identified in the brief: all eight shared ports are occupied during the household evening peak, while the 1000 kWh battery supplies approximately 150 kW and is empty by the end of the hour.

The dashboard demonstrates that the required observation surface exists and matches the specified outputs, and the displayed numbers are recorded simulation results. In workspace ws-3db7ac98, the Agentic AI carried the accepted specification through implementation, verification, calibration and validation, and experimentation while invoking the bounded coding runtime for authorised repository changes. The corrected model represents 150 all-electric homes with heat pumps and approximately 4 kWp of rooftop photovoltaics each, a 250 kW wind turbine, eight 7.4 kW ports shared by approximately forty EVs, a shared battery with a 90% usable window, 0.5 C power limit, and 90% one-way efficiency, and a 400 kW feeder. The model is balanced hourly over a 744-h winter reporting window following a two-week warm-up with seeded synthetic weather and demand.

Inspection of the Simulation UI revealed that the shared EV chargers were not being used during the intended evening peak, while most charging occurred after midnight. The first author identified this discrepancy by comparing the observed results with the intended behaviour of residents returning home in the early evening, connecting their vehicles, and charging until their energy needs were met or they departed the following morning. He

suggested that the simulated charging cycle might have been aligned with calendar-day boundaries rather than the period between arrival and departure. He therefore requested that the Agentic AI examine the charging sequence, introduce an explicit check for the expected evening charging pattern, and repeat the experiments. The Agentic AI subsequently inspected and revised the implementation, introduced the targeted verification check, corrected timestamp handling, and reran the experiments. Runs 1–23 were retained as superseded development records, while corrected runs 24–47 provide the reported results. The episode illustrates the complementary roles of automated verification and domain-informed inspection: although the initial implementation passed its automated checks, comparison with the intended real-world behaviour revealed a semantic discrepancy that those checks had not detected.

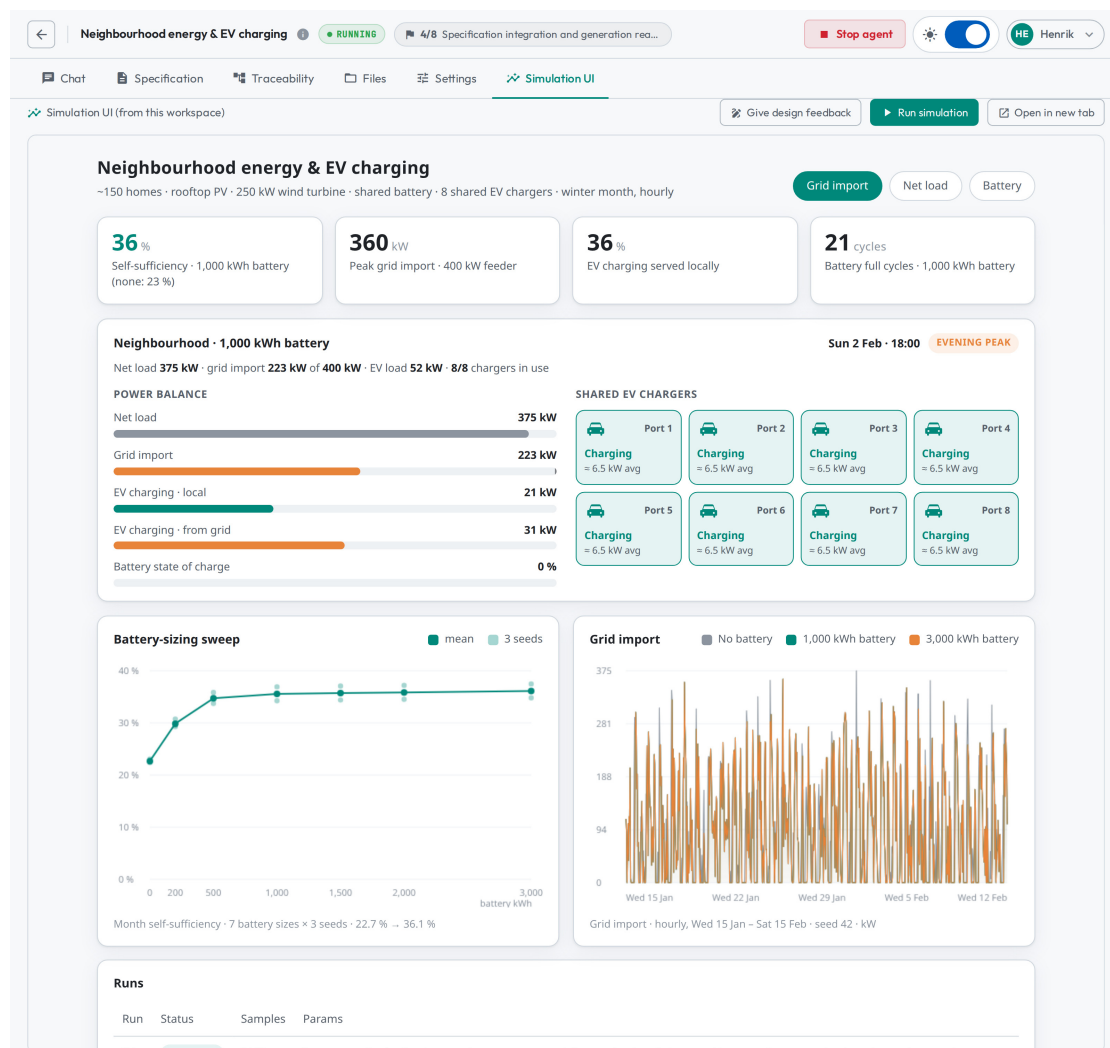


Figure 10. The Simulation UI for the neighbourhood-energy & EV-charging case, showing results recorded from the AgentSim model generated in a live platform session: KPI tiles (self-sufficiency, peak grid import, EV charging served locally, battery cycles), the power balance and the eight shared 7.4 kW charging ports at the month’s busiest hour (18:00, 2 February, 1000 kWh battery), the battery-sizing sweep over seven capacities and three seeds, and the grid-import series for three battery sizes. Genuine platform capture.

Corrected runs 24–47 comprise seven battery capacities across the three base-sweep seeds 42, 7, and 99, two additional stormier-wind runs, and a 300 kW feeder sensitivity. Shared charging begins at the 18:00 arrival peak. Peak grid import is 349–375 kW, so the 400 kW feeder does not bind, whereas the 300 kW sensitivity binds for 23 h and records 446 kWh

of curtailed demand. Mean self-sufficiency rises from 22.7% without storage to 35.6% at 1000 kWh and 36.1% at 3000 kWh. Under the stormier-wind profile, it reaches 36.6% without storage and 56.0% at 3000 kWh. No tested base configuration reaches the user-supplied 80% stretch target, and incremental gains diminish sharply beyond approximately 1000 kWh under the stated synthetic profiles and dispatch logic. These values support structural inspection rather than prediction. Exact run-level timestamps, timezone, and sensitivity parameterisations remain fixed in the supplied manifests and recorded results rather than being reconstructed from prose. Table 20 summarises the retained evidence.

Table 20. Retained evidence from the domain brief through the completed workspace workflow.

Evidence Item	Observed Result	Interpretation
Domain-level entry	The brief and domain clarifications contained the self-sufficiency and battery-capacity purpose, the neighbourhood scenario, domain context and conditions, the evening EV coincidence, and the 80% stretch target, but no ABM design.	The workflow received domain meaning and objectives through user interaction without receiving an agent-based model design.
AI-authored specification	All fourteen sections are written across ninety-six passages, with the three domain-agent types and their associated passive components, hourly sequence, outputs, and readiness judgement.	The Agentic AI converted the brief into an implementable model description.
Provenance tally	Two sourced, fifty-seven AI-derived, two approved assumptions, thirty-two user-correction passages, three non-blocking limitations, and zero unresolved blocking matters.	The thirty-two corrected passages propagate a smaller set of domain corrections through affected sections; they do not represent thirty-two user-authored modelling decisions.
Domain-authority corrections	The user corrected the feeder limit, clarified the approximately forty-EV shared-charging population, and strengthened the prohibited-use statement.	Domain authority remained with the user and materially constrained the model's meaning and permitted claims.
Generation gate	Figure 7 records v35 with a stale approval after material edits; the retained workflow records renewed user authorisation before downstream completion.	Generation remained user-authorised, and version invalidation prevented material edits from silently inheriting an earlier approval.
Observation surface	The Simulation UI exposes demand, generation, storage, EV load, grid exchange, and self-sufficiency.	The specified observation surface is realised in the dashboard.
Numeric outputs	The dashboard series are results recorded from 24 corrected runs: seven battery sizes from 0 to 3000 kWh across three seeds, a stormier-weather pair, and a 300 kW feeder sensitivity.	All reported numerical values use corrected runs 24–47; runs 1–23 remain only as superseded development evidence, and the synthetic inputs bound the interpretation to structural inspection.
Workflow recovery	Dashboard inspection exposed the missing evening EV coincidence; the first author suggested an arrival-to-departure interpretation, after which the Agentic AI investigated, revised and checked the charging cycle, rebuilt, and reran the experiment.	A domain-semantic mismatch prompted a controlled return and AI-implemented verification and revision following domain-informed user guidance.

8.7. Demonstration Against the Interaction Criteria

The retained workspace records provide functional evidence against the same ten specified dimensions stated in Table 4. The user supplied the domain brief, scenario context and conditions, corrections, and gate authority without supplying an agent-based model design. The Agentic AI framed the questions, derived the model, acquired and recorded technical evidence, authored and maintained the specification, invoked bounded coding for implementation and repair, evaluated verification evidence, bounded validity, authored the Simulation UI, and executed the corrected experiment under a controller-governed state. The EV timing episode provides additional evidence of the return mechanism: domain-informed inspection identified a discrepancy, and the Agentic AI subsequently investigated, revised and verified the model implementation before corrected execution. Table 21 evaluates each dimension against the retained evidence.

Table 21. Functional assessment of the domain-to-simulation interaction.

Evaluation Dimension	Case Evidence	Result
Domain-level entry	The session began with an ordinary-language problem, intended use, and scenario context rather than a simulation plan or development procedure.	Functionally supported in the captured session
No ABM-specific user input	Recorded user input supplied purpose, scenario context and conditions, corrections, and authority but did not define agents, state, behaviour, scheduling, tests, or AgentSim constructs.	Functionally supported in the captured session
Autonomous specification construction	The Agentic AI authored and maintained all fourteen specification sections across ninety-six passages and declared generation readiness without delegating model design to the user.	Functionally supported through the completed specification and readiness stage
Technical knowledge acquisition	Technical gaps were handled through skills, AgentSim and project documentation, data, and recorded sources or assumptions, while user-supplied scenario context remained explicitly separated from AI-derived modelling.	Functionally supported by the retained source, domain-context, evidence, and assumption records
Human domain authority	The user supplied scenario context, corrected the feeder and shared-EV interpretation, restricted permitted uses, initiated correction of the observed EV-timing mismatch, accepted the validity boundary, and retained the Generation and Package gate authority.	Functionally supported through recorded corrections and approvals
Procedural control	Stage state, gates, returns, and the rule that only the workflow controller changes the active stage remained controller-governed throughout the retained session.	Functionally supported in the captured controller and WorkflowState history
Semantic consistency	Specification and implementation checks covered ordering, energy accounting, capacity and state bounds, output coverage, EV timing, and feeder behaviour; corrected execution closed the identified EV timing defect.	Functionally supported after correction by the retained validators, checks, and traceability records
Bounded implementation	Specification-to-code mappings authorised repository work through the coding runtime; meaning-preserving implementation defects were repaired without silently redefining accepted model semantics.	Functionally supported in the retained mappings, ACP events, diffs, builds, and verification record

Table 21. *Cont.*

Evaluation Dimension	Case Evidence	Result
Workflow recovery	User inspection exposed the missing evening charging pattern; a domain-informed arrival-to-departure interpretation prompted the Agentic AI to investigate, revise and check the cycle, rebuild, and rerun the experiment without bypassing model or authority gates.	Functionally supported by the recorded return, repair, and corrected-run lineage
Executable completion	The repaired package built, passed discovery and verification checks, executed remotely, produced corrected runs 24–47, and exposed the mandatory observations through the Simulation UI.	Functionally demonstrated for the reported synthetic case

The demonstration provides functional evidence across the completed workflow for domain-level entry, absence of ABM-specific user instructions, autonomous specification construction, technical knowledge acquisition, retained human domain authority, controller-governed procedure, semantic checking, bounded implementation, controlled recovery, and executable completion. It does not establish that the approach is faster, easier, or more effective than experienced modeller-led development. Nor does it establish empirical validity for a real feeder, operational safety, protection settings, grid-code compliance, financial bankability, statistical robustness beyond the recorded seeds and scenarios, or usability for representative domain experts. Those outcomes require separate empirical and comparative studies.

9. Discussion

9.1. Answers to the Research Questions

RQ1 asked how an inverted-initiative interaction can support participation through domain-level input and authority rather than agent-based modelling instructions. The implemented workflow begins from an ordinary-language domain brief and constrains the user role to purpose, intended use, scope, questions, domain context, correction, assumptions, limitations, and gates. The Agentic AI normally initiates requests for such input as the procedure progresses, while the user remains able to initiate a domain-level correction when inspection reveals a mismatch in meaning. In the reported session, the user interaction contained no instructions defining agents, state variables, behaviours, schedules, tests, or AgentSim constructs. The session therefore provides functional evidence that the platform can allocate modelling work to the Agentic AI while retaining human domain authority.

RQ2 asked how the Agentic AI can construct and assess a natural-language specification. Simulation literature supplies the coverage obligations, while the workflow turns them into a living structured English artefact. The Agentic AI progressively authors its sections, evaluates gaps and contradictions after every change, traces evidence and assumptions, and declares the specification complete enough when all implementation-critical content is coherent, supported or explicitly bounded, testable, and approved at the generation gate.

RQ3 asked how autonomous technical work can coexist with human authority. The Agentic AI uses skills, AgentSim knowledge, project sources, data, documentation, and authorised online search to resolve modelling and implementation gaps. It makes supported technical decisions and proposes bounded assumptions. The Agentic AI initiates user contact when domain knowledge or accountable approval is indispensable, while the user may independently correct observable domain meaning without taking over the

technical procedure. The controller, rather than the language model, applies progression, and retrieved content or coding tools cannot redefine accepted meaning.

RQ4 asked how the interaction can be operationalised and functionally demonstrated. The platform combines eight development stages, specification and source versioning, readiness gates, search and coding permissions, semantic validators, controlled returns, traceability, packaging, and remote execution. In the reported session, the lineage progressed from a short problem statement to a parameterised AgentSim IScenario implementation and comparison views without a user-authored development plan or model specification. The EV timing defect provides direct evidence of the return mechanism: user inspection exposed a mismatch in observable domain behaviour, after which the Agentic AI diagnosed the implementation cause, returned to the responsible activity, repaired and rechecked the cycle, and completed corrected runs 24–47.

9.2. Inversion of Initiative as the Principal Contribution

The principal contribution is not greater autonomy inside a coding task. It is an implemented responsibility allocation for transforming a domain investigation into a specified, implemented, and evaluated agent-based simulation. The Agentic AI contains the procedure and performs the modelling work. It determines what must be represented, finds the technical information required to represent it, authors the simulation specification, and normally initiates human interaction where the user possesses indispensable domain knowledge or authority. Human authority remains active throughout inspection, so the user may also initiate a correction when observable artefacts or behaviour conflict with intended domain meaning. The demonstration provides case-specific evidence of this responsibility allocation rather than establishing its general effectiveness across users or simulation domains.

The distinction between IoI and existing mixed-initiative approaches is therefore not that the AI can initiate questions, plan activities, or execute tools. Rather, IoI defines a specific allocation of responsibility in which the Agentic AI determines and carries out the simulation-development procedure, constructs the agent-based model, and evaluates the evolving simulation specification, while the human retains authority over domain meaning, consequential assumptions, and progression approvals. This allocation can be examined through the recorded user inputs, AI-authored specification, workflow transitions, and approval records, rather than inferred solely from conversational behaviour.

The structured natural-language specification makes this stronger inversion operational. It translates specialised modelling obligations into a persistent artefact the Agentic AI can evaluate and the user can read. The user does not fill the template. The template allows the system to recognise what it still needs to determine. This distinction separates the reported platform from conversational assistants that make modelling easier while leaving the user responsible for specifying the model.

The platform mechanisms jointly realise this contribution. The controller preserves procedure. Skills encode modelling and assurance work. Technical retrieval extends the knowledge available for component design. The specification carries meaning. Gates prevent premature generation. AgentSim constrains implementation, and assurance and execution services provide evidence. These are parts of one interaction paradigm rather than separate contributions.

9.3. Implications for Simulation Development

For domain users, the interaction model is designed to reduce the requirement to act as simulation developers before investigating a problem. A user can describe what needs to be understood and inspect the Agentic AI's interpretation in ordinary language. In the

reported session, the platform assumed responsibility for agentification, state, behaviour, interaction, time, parameters, implementation, and testing without receiving this content through the user interaction. Whether representative domain experts can understand, correct, and confidently approve the resulting artefacts remains an empirical question for further study.

Human participation remains essential, but changes form. The user corrects misunderstandings of the real system, supplies inaccessible local facts, and accepts or rejects consequential assumptions and limitations. The Agentic AI must explain these matters in terms of their domain effect. Approval should not require the user to reason about classes, events, schedulers, or validation methods.

9.4. Implications for Agentic Software Engineering

The architecture illustrates a broader form of procedure-aware agentic software engineering in which the user states an outcome at the level of their own expertise while the system owns a specialised development procedure. Inversion of initiative therefore describes a class of Agentic AI applications in which the AI owns a specialised process and initiates the interaction, while the human contributes domain knowledge and accountable authority. The name parallels inversion of control in software frameworks, where the framework rather than the application owns the flow of execution. Here the Agentic AI rather than the user owns the initiative of the development process. Recent software-engineering work catalogues architectural patterns for foundation-model agents, covering goal seeking, plan generation, tool use, and accountability [41]. The platform instantiates the orchestrator and subagent arrangement with a deterministic orchestrator, and inversion of initiative names a complementary pattern at the level of human and AI responsibility rather than agent architecture. Tool use, retrieval, and repository autonomy are insufficient unless a host system also represents the evolving specification, completion conditions, authority boundaries, and recovery paths.

The distinction between semantic returns and local implementation repairs is particularly important. Compiler failures and test results are useful feedback, but the permitted response depends on the cause. A coding agent may repair an import or address syntax. It may not alter execution order or behavioural meaning merely to make tests pass. The controller-governed return mechanism converts that distinction into executable policy.

9.5. Integrated Role of AgentSim and the Platform Mechanisms

AgentSim is essential as the bounded implementation target, but it is not the contribution in isolation. Its stable abstractions and execution services allow the Agentic AI to map natural-language statements to known constructs and validate ordering, messaging, observability, randomness, persistence, and packaging. A different target framework would require different implementation skills and validators while leaving the domain-level entry and AI-authored specification concept intact.

Likewise, the neighbourhood energy-balancing simulation artefact is not a separate domain contribution. It serves as a demanding object through which the interaction can be inspected. Its coordinated scheduled and reactive agents, capacity limits, stateful battery, and observable scenario through a dashboard required the Agentic AI to exercise the complete workflow. The domain outputs serve as evidence of operational completion for the demonstrated configuration but do not define the novelty of the article.

9.6. Human Authority and Bounded Autonomy

Inverting the initiative does not imply unrestricted autonomy. The Agentic AI controls modelling and procedure but does not possess accountable authority over the user's purpose or the consequences of using the model. The user confirms intended use, corrects

material interpretations, supplies facts unavailable from authorised sources, accepts or rejects consequential assumptions, and approves the validity boundary and release gates.

The EV timing episode illustrates how domain-informed inspection can complement automated verification. Although the initial implementation passed its automated checks, the first author recognised that the observed charging pattern conflicted with the intended behaviour of residents arriving home and charging during the evening. His proposed arrival-to-departure interpretation helped identify the discrepancy, which the Agentic AI subsequently investigated, corrected, and verified. This selective human intervention is consistent with the proposed responsibility allocation, although the demonstration does not establish that automated assurance or users without simulation-development expertise would necessarily detect comparable semantic errors.

This boundary also constrains online search. Retrieved sources can support technical facts and modelling choices, but they cannot decide that a model is suitable for a consequential use or that a weak assumption is acceptable. The platform contributes by locating, synthesising, exposing, and tracing evidence while keeping source content, AI reasoning, human approval, and controller authority distinct.

9.7. Limitations and Threats to Validity

The demonstration is based on one session, one illustrative domain, one user role, and one target framework. It provides evidence of functional realisation under these conditions. It does not establish that every domain question can be transformed reliably, that users without modelling expertise will detect an incorrect but plausible specification, or that different language models, search systems, and coding runtimes will make equivalent decisions. The skills, template, and validators are designed to carry the required simulation-development doctrine explicitly rather than to assume it from model training, although the sufficiency of this encoding across language models has not been tested.

The demonstration user was the first author, who also developed the platform. Familiarity with the platform and with software engineering may have shaped how the domain brief was expressed and how questions and approval requests were understood, so the session cannot show how representative domain experts without this background experience the interaction. The energy-domain review by the last author addresses the plausibility of the produced model rather than the usability of the interaction for independent users.

The evaluation is functional rather than comparative. It does not measure development time, cognitive effort, defect rates, or cost relative to direct prompting, user-directed agentic development, or expert modeller-led work. The ten functional criteria are research-team-defined artefact-conformance criteria used to assess whether the implemented session realises the intended responsibility allocation; they are not an independent effectiveness benchmark or a preregistered comparative outcome set. Within the reported session, the user interaction did not contain agent-based model design content. The first implementation nevertheless passed automated checks despite incorrect EV timing, and the mismatch was exposed by user inspection of the dashboard. The study therefore does not establish that automated assurance alone can detect plausible semantic defects, that the explanations and approval requests are understandable to representative novice users, or that independent users can reliably recognise materially incorrect AI interpretations.

Online search expands technical reach but introduces source-quality, coverage, versioning, and prompt-injection risks. Retrieval cannot create independent empirical evidence, and plausible synthesis may still be wrong. The illustrative model therefore uses synthetic profiles and supports only structural inspection. Its scenario views do not substitute for calibration, independent validation, uncertainty analysis, or repeated-run reproducibility.

Containerisation improves build consistency but is not a complete security boundary for generated code or untrusted dependencies.

9.8. Implications for Further Evaluation

Further studies should compare the inverted initiative with prompt-to-code interaction, user-directed agentic development, and expert modeller-led workflows. Relevant outcomes include the modelling knowledge demanded from users, correctness and completeness of the AI-authored specification, quality and provenance of retrieved evidence, number of consequential assumptions exposed before implementation, semantic defects, time to verified execution, and user comprehension of approval requests.

Multi-session studies should involve users with strong domain knowledge but different levels of modelling expertise, additional simulation domains, and alternative language models, search systems, and coding runtimes. Particular attention should be given to cases in which online sources conflict, domain mechanisms are poorly documented, or an untrained user must recognise a materially incorrect AI interpretation.

10. Conclusions

This article has presented an inverted-initiative model for agent-based simulation development. Instead of requiring a user to understand simulation methodology and express it through prompts, a procedure-aware Agentic AI accepts a domain-level investigation, performs the modelling work, constructs and evaluates a structured natural-language specification, obtains technical information, invokes bounded implementation and assurance tools, and controls progression and return.

The user's role is limited to the knowledge and authority that cannot legitimately be automated: explaining the problem and intended use, clarifying the relevant real system, correcting interpretation, supplying inaccessible local facts, approving consequential assumptions and limitations, and authorising gates. A deterministic controller preserves workflow authority, while versioned artefacts, technical sources, skills, traceability, semantic validators, bounded coding, and remote execution connect the conversation to the complete lifecycle.

The neighbourhood energy-balancing simulation provides an end-to-end functional demonstration from a short domain brief to an AI-authored simulation specification and a remotely executed AgentSim artefact. The neighbourhood energy-balancing simulation remains an illustrative product rather than a domain contribution. The principal contribution is a procedure-aware platform in which the Agentic AI owns both the simulation-development procedure and the agent-based modelling work. The reported session supports the feasibility of the interaction model in this particular case; its effectiveness, usability, and generalisability require subsequent independent evaluation.

Supplementary Materials: The following supporting information can be downloaded at: <https://www.mdpi.com/article/10.3390/a19110920/s1>, The accompanying supplementary material provides the inspectable evidence behind the demonstration. Section S1 defines the supplement scope and provenance. Section S2 reports the complete fourteen-section, ninety-six-passage structured specification and its passage-level status totals. Section S3 documents the inversion-of-initiative interaction, generation and verification evidence, and the Agentic AI-authored Simulation UI. Section S4 provides implementation-aligned model diagrams. Section S5 reports the corrected simulation evidence from runs 24–47. Section S6 documents discovery and correction of the EV daily-cycle timing defect. Section S7 identifies the reproducibility files, generated source, recorded runs, runnable model, dashboard, diagrams, and integrity manifest supplied with the evidence package. These materials support inspection of the platform realisation without changing the article's focus on the inverted-initiative interaction model.

Author Contributions: Conceptualisation, H.S., Z.G.M. and B.N.J.; methodology, H.S., Z.G.M. and B.N.J.; software, H.S.; validation, H.S., Z.G.M. and B.N.J.; investigation, H.S.; writing, original draft, H.S. and B.N.J.; writing, review and editing, all authors; supervision, project administration, and funding acquisition Z.G.M. and B.N.J. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by Innovation Fund Denmark, grant number 5235-00008B; and the Energy Technology Development and Demonstration Programme (EUDP), Danish Energy Agency, grant number 134251-549133, and 134243-533239.

Institutional Review Board Statement: Not applicable. The demonstration was conducted by an author as part of software development and did not involve external human participants, personal data, or animal subjects.

Informed Consent Statement: Not applicable.

Data Availability Statement: The supplementary evidence package described in Section S7 contains the fourteen-section simulation specification, session interaction and tool records, workflow state, generated simulation source, build configuration, verification records, dashboard artefacts, and results from all forty-seven recorded runs. Corrected runs 24–47 support the reported numerical results, while runs 1–23 are retained as superseded development evidence. The supplied artefacts support independent inspection of the recorded development process and simulation results. Re-execution of the generated model requires Java 25, Maven, and the relevant AgentSim dependencies. The underlying AgentSim framework and production platform are not distributed as part of the supplementary package and remain available from the corresponding author subject to University of Southern Denmark access and security conditions. Consequently, the supplied evidence supports examination of the reported demonstration but does not provide an independently deployable reproduction of the complete Agentic AI platform.

Acknowledgments: Generative AI is an integral component of the research artefact. The authors designed the workflow controller, development skills, natural-language specification structure, progression gates, semantic validators, authority boundaries, and mappings. Language-model reasoning was used for domain-brief interpretation, model synthesis, technical retrieval, specification construction, question and approval generation, repository modification, code generation, diagnostic interpretation, and repair within controller-defined permissions. Material domain assumptions, validity statements, and manuscript content were reviewed and approved by the authors. Generative AI tools were also used under author supervision for language editing and manuscript preparation.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Ringler, P.; Keles, D.; Fichtner, W. Agent-based modelling and simulation of smart electricity grids and markets—A literature review. *Renew. Sustain. Energy Rev.* **2016**, *57*, 205–215. [\[CrossRef\]](#)
2. Chassin, D.P.; Fuller, J.C.; Djilali, N. GridLAB-D: An Agent-Based Simulation Framework for Smart Grids. *J. Appl. Math.* **2014**, *2014*, 492320. [\[CrossRef\]](#)
3. Værbak, M.; Ma, Z.; Demazeau, Y.; Jørgensen, B.N. A generic agent-based framework for modeling business ecosystems: A case study of electric vehicle home charging. *Energy Inform.* **2021**, *4*, 28. [\[CrossRef\]](#)
4. Jimenez, C.E.; Yang, J.; Wettig, A.; Yao, S.; Pei, K.; Press, O.; Narasimhan, K.R. SWE-bench: Can language models resolve real-world GitHub issues? In Proceedings of the Twelfth International Conference on Learning Representations, Vienna, Austria, 7–11 May 2024.
5. Wang, X.; Li, B.; Song, Y.; Xu, F.F.; Tang, X.; Zhuge, M.; Pan, J.; Song, Y.; Li, B.; Singh, J.; et al. OpenHands: An open platform for AI software developers as generalist agents. In Proceedings of the Thirteenth International Conference on Learning Representations, Singapore, 24–28 April 2025.
6. Yang, J.; Jimenez, C.E.; Wettig, A.; Lieret, K.; Yao, S.; Narasimhan, K.; Press, O. SWE-agent: Agent-computer interfaces enable automated software engineering. *Adv. Neural Inf. Process. Syst.* **2024**, *37*, 50528–50652. [\[CrossRef\]](#)
7. Frydenlund, E.; Martínez, J.; Padilla, J.J.; Palacio, K.; Shuttleworth, D. Modeler in a box: How can large language models aid in the simulation modeling process? *Simulation* **2024**, *100*, 727–749. [\[CrossRef\]](#)

8. Siebers, P.O. Exploring the potential of conversational AI support for agent-based social simulation model design. *J. Artif. Soc. Soc. Simul.* **2025**, *28*, 2. [\[CrossRef\]](#)
9. Khatami, S.; Frantz, C.K. From concepts to model: Automating feature extraction of agent-based models using large language models. *J. Artif. Soc. Soc. Simul.* **2025**, *28*, 9. [\[CrossRef\]](#)
10. Chen, J.; Lu, X.; Du, Y.; Rejtig, M.; Bagley, R.; Horn, M.S.; Wilensky, U.J. Learning agent-based modeling with LLM companions: Experiences of novices and experts using ChatGPT & NetLogo Chat. In Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems, Honolulu, HI, USA, 11–16 May 2024.
11. Robinson, S. Conceptual modelling for simulation part I: Definition and requirements. *J. Oper. Res. Soc.* **2008**, *59*, 278–290. [\[CrossRef\]](#)
12. Robinson, S. Conceptual modelling for simulation part II: A framework for conceptual modelling. *J. Oper. Res. Soc.* **2008**, *59*, 291–304. [\[CrossRef\]](#)
13. Borshchev, A. *The Big Book of Simulation Modeling: Multimethod Modeling with AnyLogic 6*; AnyLogic North America: Chicago, IL, USA, 2013.
14. Cetinkaya, D.; Verbraeck, A.; Seck, M.D. MDD4MS: A model driven development framework for modeling and simulation. In Proceedings of the 2011 Summer Computer Simulation Conference, Hague, The Netherlands, 27–30 June 2011; pp. 113–121.
15. Ma, Z.; Schultz, M.J.; Christensen, K.; Værbak, M.; Demazeau, Y.; Jørgensen, B.N. The Application of Ontologies in Multi-Agent Systems in the Energy Sector: A Scoping Review. *Energies* **2019**, *12*, 3200. [\[CrossRef\]](#)
16. Grimm, V.; Berger, U.; DeAngelis, D.L.; Polhill, J.G.; Giske, J.; Railsback, S.F. The ODD protocol: A review and first update. *Ecol. Model.* **2010**, *221*, 2760–2768. [\[CrossRef\]](#)
17. Grimm, V.; Railsback, S.F.; Vincenot, C.E.; Berger, U.; Gallagher, C.; DeAngelis, D.L.; Edmonds, B.; Ge, J.; Giske, J.; Groeneveld, J.; et al. The ODD protocol for describing agent-based and other simulation models: A second update to improve clarity, replication, and structural realism. *J. Artif. Soc. Soc. Simul.* **2020**, *23*, 7. [\[CrossRef\]](#)
18. Müller, B.; Bohn, F.; Dreßler, G.; Groeneveld, J.; Klassert, C.; Martin, R.; Schlüter, M.; Schulze, J.; Weise, H.; Schwarz, N. Describing human decisions in agent-based models: ODD + D, an extension of the ODD protocol. *Environ. Model. Softw.* **2013**, *48*, 37–48. [\[CrossRef\]](#)
19. Grimm, V.; Revilla, E.; Berger, U.; Jeltsch, F.; Mooij, W.M.; Railsback, S.F.; Thulke, H.H.; Weiner, J.; Wiegand, T.; DeAngelis, D.L. Pattern-oriented modeling of agent-based complex systems: Lessons from ecology. *Science* **2005**, *310*, 987–991. [\[CrossRef\]](#) [\[PubMed\]](#)
20. Grimm, V.; Augusiak, J.; Focks, A.; Frank, B.M.; Gabsi, F.; Johnston, A.S.A.; Liu, C.; Martin, B.T.; Meli, M.; Radchuk, V.; et al. Towards better modelling and decision support: Documenting model development, testing, and analysis using TRACE. *Ecol. Model.* **2014**, *280*, 129–139. [\[CrossRef\]](#)
21. 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. *J. Simul.* **2019**, *13*, 55–67. [\[CrossRef\]](#)
22. Sargent, R.G. Verification and validation of simulation models. *J. Simul.* **2013**, *7*, 12–24. [\[CrossRef\]](#)
23. Zeigler, B.P.; Muzy, A.; Kofman, E. *Theory of Modeling and Simulation: Discrete Event and Iterative System Computational Foundations*, 3rd ed.; Academic Press: San Diego, CA, USA, 2019.
24. Railsback, S.F.; Grimm, V. *Agent-Based and Individual-Based Modeling: A Practical Introduction*, 2nd ed.; Princeton University Press: Princeton, NJ, USA, 2019.
25. Galán, J.M.; Izquierdo, L.R.; Izquierdo, S.S.; Santos, J.I.; del Olmo, R.; López-Paredes, A.; Edmonds, B. Errors and artefacts in agent-based modelling. *J. Artif. Soc. Soc. Simul.* **2009**, *12*, 1.
26. Hörner, L.F.; Möller, M.; Reichert, M. Automated Business Process Model Generation with Large Language Models: A Systematic Literature Review. *IEEE Access* **2026**, *14*, 125550–125568. [\[CrossRef\]](#)
27. White, J.; Fu, Q.; Hays, S.; Sandborn, M.; Olea, C.; Gilbert, H.; Elnashar, A.; Spencer-Smith, J.; Schmidt, D.C. A prompt pattern catalog to enhance prompt engineering with ChatGPT. *arXiv* **2023**, arXiv:2302.11382. [\[CrossRef\]](#)
28. Horvitz, E. Principles of mixed-initiative user interfaces. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, Pittsburgh, PA, USA, 15–20 May 1999; pp. 159–166.
29. Siebers, P.O.; Klügl, F. What software engineering has to offer to agent-based social simulation. In *Simulating Social Complexity: A Handbook*, 2nd ed.; Edmonds, B., Meyer, R., Eds.; Springer: Cham, Switzerland, 2017; pp. 81–117.
30. Stray, V.; Barbala, A.; Wivestad, V.T. Human-AI Collaboration in Software Development: A Mixed-Methods Study of Developers’ Use of GitHub Copilot and ChatGPT. In Proceedings of the FSE Companion ’25: Proceedings of the 33rd ACM International Conference on the Foundations of Software Engineering, Trondheim, Norway, 23 June 2025; pp. 1325–1332.
31. Hemmat, A.; Sharbaf, M.; Kolahdouz-Rahimi, S.; Lano, K.; Tehrani, S.Y. Research directions for using LLM in software requirement engineering: A systematic review. *Front. Comput. Sci.* **2025**, *7*, 1519437. [\[CrossRef\]](#)
32. Madaan, A.; Tandon, N.; Gupta, P.; Hallinan, S.; Gao, L.; Wiegrefe, S.; Alon, U.; Dziri, N.; Prabhunoy, S.; Yang, Y.; et al. Self-Refine: Iterative refinement with self-feedback. *Adv. Neural Inf. Process. Syst.* **2023**, *36*, 46534–46594. [\[CrossRef\]](#)

33. Boettiger, C. An introduction to Docker for reproducible research. *ACM SIGOPS Oper. Syst. Rev.* **2015**, *49*, 71–79. [[CrossRef](#)]
34. Merkel, D. Docker: Lightweight Linux containers for consistent development and deployment. *Linux J.* **2014**, *2014*, 2.
35. Project, J.; Bussonnier, M.; Forde, J.; Freeman, J.; Granger, B.; Head, T.; Holdgraf, C.; Kelley, K.; Nalvarte, G.; Osheroff, A.; et al. Binder 2.0: Reproducible, interactive, sharable environments for science at scale. In Proceedings of the 17th Python in Science Conference, Austin, TX, USA, 9–15 July 2018; pp. 113–120.
36. Brinckman, A.; Chard, K.; Gaffney, N.; Hategan, M.; Jones, M.B.; Kowalik, K.; Kulasekaran, S.; Ludäscher, B.; Mecum, B.D.; Nabrzyski, J.; et al. Computing environments for reproducibility: Capturing the “Whole Tale”. *Future Gener. Comput. Syst.* **2019**, *94*, 854–867. [[CrossRef](#)]
37. Agent Client Protocol. Agent Client Protocol Specification. 2026. Available online: <https://agentclientprotocol.com> (accessed on 21 October 2026).
38. Hevner, A.R.; March, S.T.; Park, J.; Ram, S. Design science in information systems research. *MIS Q.* **2004**, *28*, 75–105. [[CrossRef](#)]
39. Peffers, K.; Tuunanen, T.; Rothenberger, M.A.; Chatterjee, S. A design science research methodology for information systems research. *J. Manag. Inf. Syst.* **2007**, *24*, 45–77. [[CrossRef](#)]
40. Agache, A.; Brooker, M.; Florescu, A.; Iordache, A.; Liguori, A.; Neugebauer, R.; Piwonka, P.; Popa, D.M. Firecracker: Lightweight virtualization for serverless applications. In Proceedings of the 17th USENIX Symposium on Networked Systems Design and Implementation, Santa Clara, CA, USA, 25–27 February 2020; pp. 419–434.
41. Liu, Y.; Lo, S.K.; Lu, Q.; Zhu, L.; Zhao, D.; Xu, X.; Harrer, S.; Whittle, J. Agent design pattern catalogue: A collection of architectural patterns for foundation model based agents. *J. Syst. Softw.* **2025**, *220*, 112278. [[CrossRef](#)]

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.