

Dr. Bernard P. Zeigler and the Foundations of Agents in Artificial Intelligence

From model-based autonomy to the engineering of agentic AI in 2026

Brad M. Philipbar

Philipbar Analytics LLC

PALLC Research White Paper

September 30, 2026 | Version 1.0

A formal foundation for intelligent agents: internal models, coordinated action, precise execution, and disciplined experimentation—developed decades before today’s language-based AI.

ABSTRACT

Bernard P. Zeigler established a rigorous systems foundation for model-based intelligent agents. With his collaborators, he developed architectures in which agents use internal representations to plan actions, monitor consequences, coordinate with other agents, and recover when execution departs from expectations. The 1988 robot-organization architecture and the 1990 monograph on intelligent agents and endomorphic systems articulated these capabilities 38 and 36 years before 2026.[1, 2] His research program connects the Discrete Event System Specification (DEVS) formalism, System Entity Structure (SES), and experimental frames into complementary foundations for behavior, architecture, and validity.[12, 14, 4] These foundations address central engineering demands of contemporary artificial intelligence (AI): explicit state, controlled tool use, event timing, component interaction, and reproducible evaluation. This paper develops their application to agents built around large language models (LLMs), presents a Parallel DEVS execution architecture, and verifies a deadline-resolution contract. Zeigler’s enduring contribution is a disciplined way to turn intelligent decision capability into a specified, executable, and testable system.

Keywords: discrete-event systems; intelligent agents; model-based autonomy; endomorphic systems; hierarchical composition; experimental frames; agentic artificial intelligence.

1 A HISTORICAL CONTRIBUTION WITH PRESENT ENGINEERING FORCE

Artificial intelligence (AI) agents perform extended tasks through tools, memory, and environmental feedback. In 2026, Anthropic describes Claude Code and Claude Cowork as systems combining learned capabilities with execution harnesses, tools, and environments. OpenAI’s workspace agents, offered in research preview, combine shared workflows with permissions, monitoring, and approval gates.[5, 6] Their practical value depends on the coordinated behavior of the complete system.

For the modeling and simulation (M&S) community, this development brings Zeigler’s systems perspective into immediate focus. Model-based agency brings together a representation of the environment, an organized way to act, and a means of responding when observations depart from expectations. Zeigler and collaborators developed these capabilities in the 1980s and 1990s, linking intelligent agency to executable models and coordinated system behavior.[1, 2]

Zeigler helped establish the *formal foundations for engineering model-based intelligent agents*. The Discrete Event System Specification (DEVS) formalism and System Entity Structure (SES) connect behavioral and structural descriptions. His research unites three engineering responsibilities: specify component behavior, construct coherent systems from interacting components, and evaluate their behavior under defined experimental conditions.[12, 14, 4] The architecture developed in this paper applies that foundation to contemporary agents.

Agent research draws on several complementary traditions. Hewitt, Bishop, and Steiger’s 1973 actor formalism treated active computational entities and message passing; Shoham’s 1993 agent-oriented programming developed mental-state and communication abstractions; Wooldridge and Jennings’s 1995 account organized agency around autonomy, social ability, reactivity, and proactivity.[7, 8, 9] Zeigler’s distinctive contribution connects intelligent behavior to a mathematically organized theory of modeling, simulation, and system construction.

1.1 Intelligent agency as a systems concept

An *agent* is used here for an entity that selects actions in an environment in pursuit of an assigned or represented objective. The weaker agency properties described by Wooldridge and Jennings supply the interpretive context. An *agent-based simulation* represents interacting agents for study; a *software agent* acts in a computational environment; and a *large language model (LLM) agent* uses a learned language capability within its action-selection process. These categories can overlap, but their evaluation objectives differ.[9, 10]

2 INTELLIGENT AGENTS DECADES BEFORE 2026

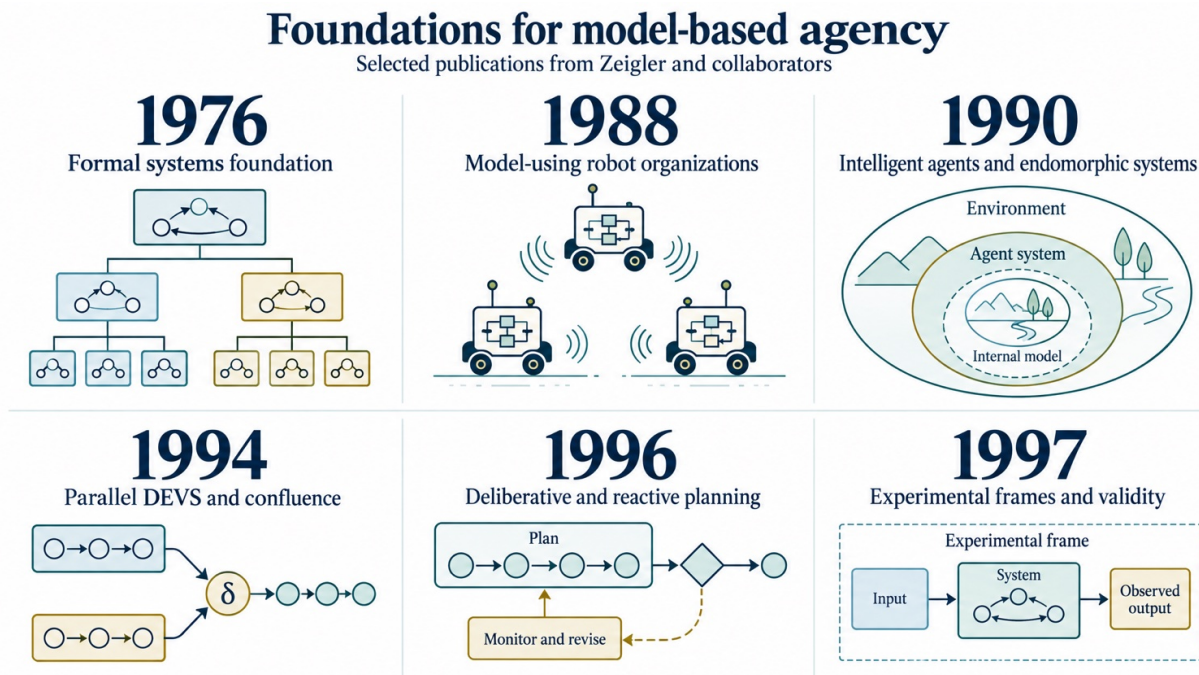


Figure 1 — Selected publications in Zeigler’s research program. The conceptual motifs summarize their engineering contributions; the dates identify the cited publications.[11, 1, 2, 3, 25, 4]

2.1 1987–1989: knowledge representation made executable

Zeigler’s 1987 article in *Applied Artificial Intelligence* linked behavioral knowledge, structural knowledge, and knowledge about model construction within a systems-theoretic representation scheme. The article connected these forms of knowledge to goal-directed construction, placing the organization of AI knowledge within the same intellectual program as simulation methodology.[23]

The 1989 DEVS-Scheme paper by Zeigler, Hu, and Rozenblit then described an implemented environment combining symbolic and numerical processing. Its layers separated the programming foundation, behavioral specification, structural/model-base organization, and system design. It associated modular construction with independent testing and described frame-and-rule mechanisms for goal-sensitive selection of structures.[24, pp. 84–89] This combination matters historically because it connected knowledge representation to an executable system-building process.

The relevant contribution was a coherent route from organized alternatives to behavior that could be generated and examined.

2.2 1988: concrete architecture for model-using robot organizations

Zeigler, François E. Cellier, and Jerzy W. Rozenblit described robot organizations for laboratory management. Their model-plan units associated anticipated process behavior with action sequences. Sensor expectations included value envelopes and timing windows; deviations could trigger diagnosis and recovery. The architecture also included requesting and offering assistance among robots. The reported tests covered three elementary assistance scenarios, while broader viability and adaptability remained research objectives.[1, pp. 299–308]

2.3 1990: intelligent agents as a sustained research program

Zeigler’s *Object-Oriented Simulation with Hierarchical, Modular Models: Intelligent Agents and Endomorphic Systems* appeared in 1990. Its organization brings internal agent models, model-plan units, event-based control, planning, diagnosis, and model-base management into a unified treatment, culminating in high autonomy. The monograph places intelligent agency at the center of a sustained systems-engineering program.[2, chs. 10–13, 15]

Endomorphy gives this conception of agency its distinctive power: an agent carries and uses representations of processes relevant to its behavior, including its environment or itself. Those internal representations support anticipation, action selection, and adaptation as objectives or constraints change.[2, 25]

2.4 1993: autonomy with explicit execution and failure behavior

Schooley, Zeigler, Cellier, and Wang’s high-autonomy control study connected task-specific abstractions and general computational engines to symbolic planning and process dynamics. Monitoring, recovery, and human intervention were integral parts of the architecture. The study reported component and prototype experience; complete-system testing remained a subsequent step.[13, pp. 29–37]

By the early 1990s, this research program joined representation, decision, physical dynamics, and exception handling in a single engineering perspective. Contemporary agent systems require the same integration: decisions must respect time, resources, and observable consequences.

2.5 1996: an explicit deliberative and reactive agent formulation

A chapter in the EUROCAST ’95 proceedings, published in 1996, developed DEVS-based endomorphic agents that use representations of themselves and their environment to construct control schemes and revise them as objectives or constraints change. Planning and control operate as complementary approaches to choosing actions over time.[25, pp. 386–387] This explicit formulation of deliberative and reactive agency predates 2026 by thirty years.

The resulting research trajectory runs from knowledge representation through executable architectures to agent deliberation and reactive control. Its organizing principle is powerful: representations guide an acting system, and formal dynamics make that system’s behavior explicit.

3 FOUR CONTRIBUTIONS THAT REMAIN TECHNICALLY CONSEQUENTIAL

3.1 Behavior as a formal specification

The DEVS formalism distinguishes the specification of dynamics from the machinery that executes it. Zeigler and Vahie’s 1993 account explicitly represents a system through sequential state and elapsed time, gives states a maximum residence time, and places output generation immediately before internal transitions. It also describes abstract simulation architectures realizable on different execution platforms.[12, pp. 573–574]

For a contemporary agent, this distinction motivates separating the *decision policy* from the *execution contract*. The decision policy may be an LLM, a planner, a controller, or a hybrid. The execution contract defines what inputs are accepted, when a pending operation expires, how results are correlated with requests, and what outputs mean. Such a contract can remain stable across changes in the learned policy. Policy behavior and execution behavior then have explicit, separately testable requirements.

3.2 Composition as a semantic construction

Chow and Zeigler’s Parallel DEVS work replaced implicit serialization of simultaneous activity with explicit treatment of input bags and collisions between internal and external transitions. Their construction preserves closure under coupling and hierarchical consistency.[3, secs. 3–5] The essential implication for agent engineering is that a composition should have a defined meaning at its boundary, including when components act at the same logical time.

A multi-agent architecture can then be assessed as a system with specified inputs and outputs. Closure under coupling establishes that the composition has a representation within the formalism. Task correctness, agreement, application requirements, and performance each receive their own acceptance criteria. Section 5 applies this separation to a contemporary agent architecture.

3.3 Architecture and knowledge organization

The SES provides a way to organize families of possible system structures and generate a selected composition. Rozenblit and Zeigler’s 1993 formulation uses canonical representations of system specifications and connects selection of an application structure with construction of component specifications and their coupling.[14, pp. 604–608]

Applied to contemporary agents, an SES can organize alternative configurations—available tools, planner variants, memory services, and permitted communication paths—before selection of an executable architecture. Explicit selection constraints and coupling rules connect configuration decisions to system behavior. The resulting design is reviewable at both the structural and behavioral levels.

3.4 Validity relative to an experimental frame

Zeigler’s 1997 National Research Council contribution distinguishes a real system, its model, a simulator, and an experimental frame. The frame identifies stimuli, observed outputs, and the context of use; both model validity and simulator adequacy are assessed in that context.[4, pp. 211–214]

Applied to agent evaluation, this yields a practical requirement: name the task population, admissible observations and actions, initialization, resource limits, and outcome criteria before claiming success. A benchmark score measures behavior under its benchmark conditions. An operational claim needs evidence that those conditions adequately represent the intended application. Section 7 develops this evaluation discipline for contemporary agents.

4 FROM INTERNAL PROCESS MODELS TO LANGUAGE-BASED AGENTS

4.1 A change in the decision mechanism

The Transformer introduced an attention-based neural architecture for sequence processing. Modern agent research subsequently demonstrated different ways of integrating language capabilities with external action: ReAct alternates generated reasoning and actions using observations from the environment; Toolformer trains a language capability to insert useful tool calls. These mechanisms expand the decision capabilities available to agent designers.[21, 17, 18]

Their engineering relationship to Zeigler’s program is best understood by distinguishing three representational levels. A learned language capability supplies a conditional distribution over possible continuations or actions. An explicit internal process model predicts behavior under stated assumptions. A system specification governs how decisions and observations change the complete agent’s state. A contemporary architecture may contain all three. Describing them with the single word “model” obscures which object has been trained, which dynamics have been specified, and which claims have been validated.

An explicit simulation model adds a distinct capability to a language-based agent: predicted consequences under stated dynamics. The combined architecture developed here assigns clear responsibilities. The LLM proposes an action or hypothesis, a bounded simulation evaluates predicted consequences, and a controller selects execution, revision, or further observation. This division makes the decision process inspectable and provides a basis for testing the contribution of each component.

4.2 Making the internal-model relationship explicit

The following formalization defines the internal-model interface used in this paper. Let z be environment state, o an observation, and $\hat{z}$ the agent’s current state estimate. Let M_w be an explicit internal process model and $\Phi_{M_w}(\hat{z}, \mathbf{a})$ its predicted trajectory under a candidate action sequence $\mathbf{a}$. A model-using agent can evaluate

$$\mathbf{a}^* \in \arg \min_{\mathbf{a} \in \mathcal{A}_{\text{adm}}} J(\Phi_{M_w}(\hat{z}, \mathbf{a}), g), \quad (1)$$

where $\mathcal{A}_{\text{adm}}$ is the set of admissible action sequences, g is the goal, and J is a stated objective function. This is an illustrative planning contract; it assumes a feasible candidate set and an attained minimum. A deployed design must define behavior for infeasibility, search exhaustion, and ambiguous optima.

In this architecture, an LLM generates candidate sequences while a simulator evaluates them. Execution produces observations for comparison with predicted observations $\hat{o}$. The monitor requires

$$d(o, \hat{o}) \leq \varepsilon, \quad t_{\text{obs}} \in [t_{\text{min}}, t_{\text{max}}], \quad (2)$$

for a declared discrepancy measure d , tolerance ϵ , and observation-time window. Violations trigger a specified revision or recovery path. The contract assigns prediction to the internal process model and assigns proposal generation to the language capability, with each evaluated against its own requirements.

The internal simulator is evaluated against the environment and task conditions it represents. Validation must address relevant dynamics, omitted constraints, and prediction errors. Task-matched experiments then measure its contribution to decision quality, robustness, and computational cost.

4.3 Engineering connections to contemporary agents

Table 1 — Engineering applications of Zeigler’s foundations to contemporary agent architectures.

Earlier contribution	Contemporary example	Engineering interpretation
Internal process representation and plan[1, 2]	ReAct updates action selection using returned observations.[17]	Connect an agent’s expectations to observations and a rule for revising action.
Hierarchical interacting components[12, 3]	AutoGen defines conversable components and interaction-driven computation.[19]	Specify message contracts, timing, ownership, and termination for the composition.
Organized structural alternatives[14]	Agent configurations combine capabilities, tools, and human participation.[19]	Make permitted configurations and their constraints explicit before execution.
Representations used within agents[2]	Generative Agents combines experience retrieval, reflection, and planning.[10]	Distinguish remembered text from a validated process representation.
Experimental context and observed behavior[4]	WebArena evaluates actions in executable web environments.[20]	Evaluate environmental outcomes under reproducible conditions.

The Generative Agents study is particularly relevant to M&S. It combines an LLM with memory, reflection, and planning to instantiate social behavior in a sandbox. Its evaluation of believability addresses a different question from predictive validity for a specified human population.[10, secs. 4, 6–7] An M&S approach can combine expressive agent behavior with explicit experimental frames, population data, and measurable criteria for the intended use.

AutoGen’s 2023 research describes agents that combine language, tool, and human capabilities through conversations.[19, sec. 2] A complete architecture specifies the value of a late result, ownership of shared state, handling of duplicate requests, and the definition of completion. The DEVS tradition supplies formal machinery for expressing these decisions as system behavior.

4.4 A continuing foundation in simulation software

Zeigler’s framework continues in simulation software. The Delft University of Technology’s Distributed Simulation Object Library (DSOL) identifies his framework as its basis and associates model, simulator, and experimental frame with software interfaces and classes. Its supported formalisms include agent modeling.[22] The result is a concrete software expression of the distinction between a system description, its execution, and its experimental context.

This separation gives contemporary agent engineering a durable architectural foundation. Learned policies and tool ecosystems can evolve while explicit interfaces, timing rules, and

experimental conditions preserve a stable basis for analysis. The following construction develops that application using Parallel DEVS.

5 A DEVS ARCHITECTURE FOR CONTEMPORARY AGENT SYSTEMS

The architecture developed here uses Parallel DEVS to define an agent’s state, event interfaces, and timing contract. It connects policy decisions to controlled execution and supplies the formal basis for the executable example in Section 6.

5.1 Atomic agent boundary and total state

Let an atomic agent model be

$$A = \langle X, Y, S, \delta_{\text{int}}, \delta_{\text{ext}}, \delta_{\text{con}}, \lambda, ta \rangle. \quad (3)$$

Here X and Y are typed input and output event sets; S is sequential state; $ta : S \rightarrow \mathbb{R}_{\geq 0} \cup \{\infty\}$ is time advance; and $\mathcal{B}(X)$ denotes finite bags of inputs. Internal, external, and confluent transitions have the normalized signatures

$$\delta_{\text{int}} : S \rightarrow S, \quad \delta_{\text{ext}} : Q \times \mathcal{B}(X) \rightarrow S, \quad (4)$$

$$\delta_{\text{con}} : S \times \mathcal{B}(X) \rightarrow S, \quad \lambda : S \rightarrow \mathcal{B}(Y), \quad (5)$$

with total state $Q = \{(s, e) : s \in S, 0 \leq e \leq ta(s)\}$. Elapsed time e is measured since the last state transition. For an input strictly before the scheduled internal transition, δ_{ext} applies; a simultaneous input and internal event uses δ_{con} . Imminent outputs are computed from the pre-transition state. These signatures express the Parallel DEVS roles in the notation used here.[3, pp. 718–721]

A possible state realization is

$$s = (p, g, m, w, r, b, v), \quad (6)$$

where p is execution phase, g the current objective, m retained memory, w an optional explicit world representation, r pending request identifiers and deadlines, b remaining budgets, and v the configuration/version identity. Each component has an engineering purpose. A response can be matched to a pending request; a deadline can expire without new input; a completed objective can move the agent into a terminal phase; and replay can identify the exact decision configuration.

For a hosted LLM or tool with variable latency, represent invocation as an output request and completion as a later input response. The simulator then handles the wait explicitly. Hiding a remote call inside an otherwise instantaneous transition confuses logical transition semantics with unbounded wall-clock execution. A chosen real-time adapter must specify how logical timestamps correspond to observations from the running system.

If the decision policy is stochastic, its uncertainty must enter the specification. One option is to treat sampled decisions as explicit inputs; another is to include a fully specified pseudorandom generator and its state. The displayed deterministic tuple applies conditionally on those inputs or states; a formulation using probability kernels is an explicit semantic extension. A seed is sufficient only for a fully controlled, deterministic implementation of that mechanism. Hosted services, changing retrieval contents, and network-dependent tool results require recorded responses and environment versions as well. Formal state completeness is a modeling decision that must be checked against the actual implementation.

5.2 Coupling and the boundary of the closure claim

For a finite component set D , a coupled model can be expressed in influence-and-translation form,

$$N = \langle X_N, Y_N, D, \{A_d\}_{d \in D}, \{I_i\}, \{Z_{i,j}\} \rangle. \quad (7)$$

The index i ranges over components and the enclosing boundary; I_i names destinations, and $Z_{i,j}$ translates source outputs into destination inputs. A planner, tool adapter, memory service, environment, and evaluator can occupy separate components. Type compatibility, request identity, and data ownership must be specified on the coupling interfaces.

For compatible components with a well-defined event evolution, the resultant can be represented within the same formalism. Its state includes component states and elapsed times; its next event is determined by the least remaining time,

$$ta_N = \min_{d \in D} (ta_d(s_d) - e_d). \quad (8)$$

This construction identifies the coupled system's next event from its component schedules. Chow and Zeigler provide the full closure construction.[3, sec. 4]

A practical agent implementation adds proof obligations: all relevant shared state must belong to a specified component or interface; simultaneous inputs must have a defined resolution; and zero-time cycles must terminate or have a justified semantics. In a distributed implementation, event ordering and timestamp assignment must be consistent with the chosen simulation protocol. Closure alone cannot correct an omitted dependency or reconcile incompatible time meanings.

Agency as an engineered system

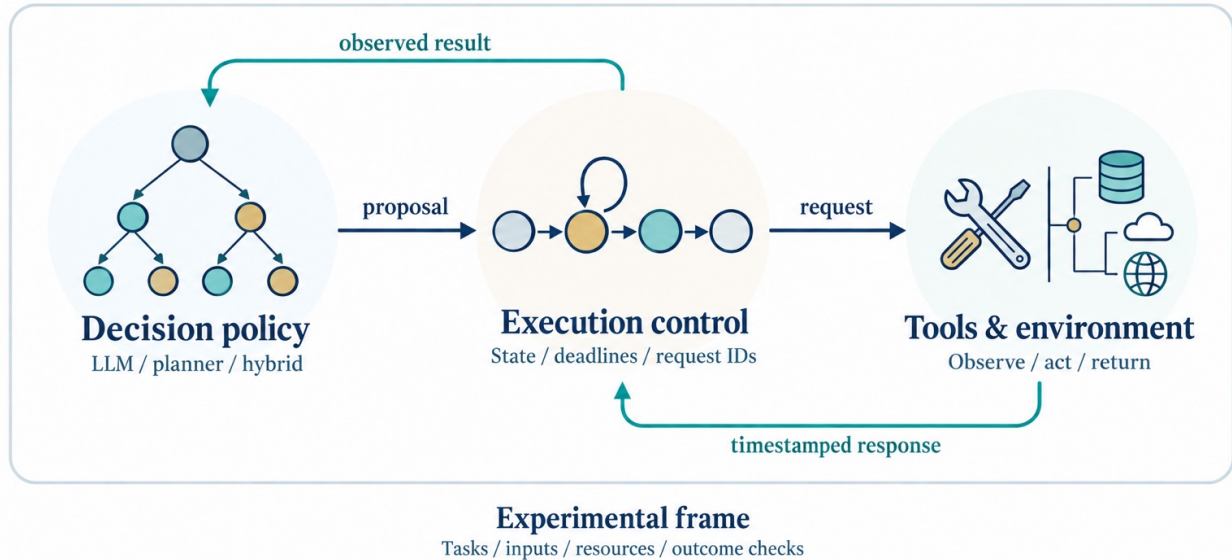


Figure 2 — Agent architecture developed in this paper. The four labeled paths carry typed events between decision, execution, and environmental interfaces; the experimental frame defines the conditions of evaluation.

6 AN EXECUTABLE EXAMPLE: A RESULT ARRIVING AT ITS DEADLINE

A small example exposes a subtle problem in agent orchestration. A controller has already issued request q at logical time zero and waits until time five. A matching result received at or before the deadline is accepted; a later result cannot reverse an emitted timeout. Identical duplicate results collapse to one outcome; different matching result values in the same complete input bag produce an explicit conflict. These rules define the illustrative contract evaluated below.

Let the phases be W (waiting), $R(o)$ (outcome ready), and D (done). While waiting, σ is the remaining time to the deadline. Define

$$ta(W, \sigma) = \sigma, \quad \lambda(W, \sigma) = \emptyset, \quad (9)$$

$$ta(R(o)) = 0, \quad \lambda(R(o)) = \{o\}, \quad (10)$$

$$ta(D) = \infty, \quad \lambda(D) = \emptyset. \quad (11)$$

The internal transition from waiting selects a timeout and moves to $R(\text{timeout})$; the next internal transition emits the selected outcome and enters D . A valid early response moves directly to $R(o)$. An unrelated response leaves the controller waiting and reduces σ by the elapsed time, thereby preserving the original deadline.

At the deadline, the confluent transition selects the valid response, a conflict, or a timeout from the complete input bag. The separate ready phase is essential: Parallel DEVS computes output *before* the confluent transition. If waiting emitted a timeout directly, the confluent transition could not retract that already generated output when a matching result arrived simultaneously. The zero-duration ready phase defers output until after resolution. Its bounded progression to D prevents an infinite zero-time loop.

Table 2 — Selected expected and verified traces. Times are exact logical units; outcomes are synthetic.

Input trace	Terminal output	Contract meaning
Matching result at $t = 4$	success at $t = 4$	Accept a result before the deadline.
Matching result at $t = 5$	success at $t = 5$	Resolve the collision before emitting an outcome.
Matching result at $t = 6$	timeout at $t = 5$	A late result cannot revise terminal output.
No result	timeout at $t = 5$	Progress does not require another external input.
Unrelated result at $t = 2$	timeout at $t = 5$	An unrelated event does not restart the deadline.
Two identical results at $t = 5$	one success at $t = 5$	Identical duplication preserves the outcome.
Different matching values at $t = 5$	conflict at $t = 5$	Bag order does not select a winner.

The companion verification script executes the contract with exact rational time. Checks cover 11 named traces, 81 result-arrival times from zero through ten in increments of one eighth, 24 permutations of a four-event deadline bag, and rejection of negative input time. All checks passed for the recorded run. Deterministic input generators, complete traces for the 11 named cases, assumptions, Python version, and script hash are retained in the repository; the sweep and permutation checks retain aggregate counts.¹

¹Reproduction: `python3 scripts/verify_agent_contract.py -report docs/verification/contract-check.json`. The script checks a deterministic semantic illustration using synthetic response events.

The example demonstrates the engineering value of explicit temporal semantics. A concurrency rule is an observable system property. Encoding it in an explicit transition structure permits inspection and repeatable checking. A conversational description such as “wait for the tool and retry if necessary” leaves several materially different behaviors possible.

7 AN M&S DISCIPLINE FOR AGENT VERIFICATION AND VALIDATION

7.1 Separate the claims being tested

The assurance structure distinguishes four complementary responsibilities. *Specification review* asks whether the state, interfaces, and timing rules express the intended behavior. *Implementation verification* asks whether the runtime follows that specification. *Empirical validation* asks whether the represented behavior adequately matches a target system for a stated purpose. *Operational acceptance* asks whether the resulting system meets its intended-use criteria. This application of experimental-frame reasoning keeps the evidence attached to the claim it can support.^[4]

For an LLM agent, scheduler verification, plan evaluation, factual assessment, and task-outcome testing have distinct acceptance criteria. Examining these layers together links local execution properties to the application’s overall requirements.

7.2 An explicit experimental frame for agent evaluation

For agent evaluation, define the operational frame

$$\mathcal{F} = \langle \mathcal{T}, \mathcal{I}, \mathcal{U}, \mathcal{O}, H, \mathcal{R}, \mathcal{C} \rangle, \quad (12)$$

where $\mathcal{T}$ is the task population or enumerated test set; $\mathcal{I}$ initial conditions; $\mathcal{U}$ admissible inputs and interventions; $\mathcal{O}$ measured outputs; H the observation horizon; $\mathcal{R}$ resource limits; and $\mathcal{C}$ acceptance criteria. This tuple is the operational evaluation record defined for the architecture in this paper.

The output record should distinguish an action request, an execution acknowledgement, and the resulting environmental state. WebArena supplies a relevant modern example by evaluating task outcomes in executable web environments, including programmatic checks of resulting state.^[20, secs. 2–3] A practical acceptance test examines the created artifact or changed application state against the specified outcome criteria.

For physical or hybrid systems, the frame also needs units, coordinate conventions, sensor uncertainty, actuator constraints, and the mapping between simulated and wall-clock time. Zeigler, Sarjoughian, and Prähofer’s work on perceiving agents formalized limited sensitivity and studied its relationship to discrete-event representation.^[16] For contemporary applications, observation filters and reduced representations are evaluated against the decision errors and timing errors that matter for the task.

7.3 Reproducibility, uncertainty, and consequential failures

A reproducible evaluation record should preserve the agent specification, implementation revision, LLM identifier and configuration, prompts, tool schemas, environment version, initial state, inputs, outputs, and event timestamps. For stochastic experiments, retain all trial results and

disclose the sampling design. Use task-matched comparisons, quantify uncertainty in the chosen performance measure, and keep logical latency separate from wall-clock latency and resource consumption.

Failure experiments should exercise the consequences of the actual interfaces: delayed and duplicate results, unavailable tools, interrupted execution, conflicting observations, and failure to satisfy a terminal condition. Authorization is part of the state and event contract whenever a tool can change an external system. A proposal that exceeds its current authority should produce a defined observable response before any external effect. Application-specific tests demonstrate conformance to these interface and authority requirements.

A comparative evaluation of this architecture should measure conversation-based orchestration, explicit execution contracts, and contracts augmented with internal simulation under the same experimental frame. Predeclared measures include task outcomes, timing violations, duplicate effects, recovery behavior, and oversight cost. This design isolates the contribution of execution semantics and internal prediction. The preceding timing example provides an executable foundation for that broader evaluation.

8 ZEIGLER’S ENDURING FOUNDATION FOR INTELLIGENT AGENTS

Dr. Bernard P. Zeigler placed model-based intelligent agency on a rigorous systems foundation decades before the emergence of today’s language-based agents. The 1988 robot-organization architecture connected internal models with action and cooperation. The 1990 monograph made intelligent agents and endomorphic systems an explicit, sustained subject of study. Subsequent work advanced autonomy, system construction, simultaneous-event semantics, and experimental evaluation.[\[1, 2, 13, 14, 3, 4\]](#)

The lasting achievement is the integration of these capabilities. Internal models support anticipation. Behavioral specifications define how agents change state. Coupling organizes interacting components into coherent systems. Experimental frames establish the conditions under which performance and validity are assessed. Together, they give agent engineering a disciplined path from an intelligent decision to an observable system outcome.

In 2026, that foundation has immediate practical value. An LLM can supply reasoning and action proposals; DEVS supplies an explicit account of the events, timing, and interactions through which those proposals become behavior. The architecture and verified deadline example developed here demonstrate how that combination makes consequential execution rules precise and testable.

For the M&S community, Zeigler’s legacy is both an intellectual foundation and an engineering opportunity. His work equips the field to shape the next generation of AI agents through formal specification, modular composition, internal simulation, and reproducible experimentation. The enduring contribution is a systems discipline for making intelligent agency executable, accountable, and fit for its intended purpose.

A TECHNICAL PROVENANCE AND REPRODUCIBILITY

The references identify original publications, proceedings papers, author-hosted copies, publisher records, and dated product documentation. Publication years refer to original editions. The companion repository preserves the claim-to-source ledger, source locators and access records, schol-

arly review, figure provenance, and the executable verification package. The research cutoff is September 30, 2026.

The Parallel DEVS agent architecture, planning interface, evaluation record, and deadline example are original applications developed for this paper. The deterministic verification package preserves its input generators, named event traces, assumptions, and implementation hash. These records support reproducible examination of the specified execution contract.

B PUBLICATION CHRONOLOGY

Table 3 — Selected milestones. Dates are original publication years; elapsed years are calendar-year differences from 2026.

Year	Years	Contribution and evidence
1976	50	<i>Theory of Modelling and Simulation</i> ; original theoretical foundation referenced in Zeigler’s later primary accounts.[11, 12]
1987	39	Knowledge representation connecting behavioral, structural, and metaknowledge.[23]
1988	38	Robot organizations with internal process representations, plans, monitoring, diagnosis, and cooperative assistance.[1]
1989	37	DEVS-Scheme integrates symbolic and numerical work with layered model construction and management.[24]
1990	36	Monograph explicitly devoted to intelligent agents and endomorphic systems, alongside hierarchical modular simulation.[2]
1993	33	Model-based high-autonomy process control; generative system specifications using SES.[13, 14]
1994	32	Parallel DEVS specifies simultaneous-event handling and confluent transitions.[3]
1996	30	Endomorphic agents use self/environment representations for deliberative and reactive planning; EUROCAST ’95 proceedings.[25]
1997	29	Explicit distinction among real system, model, simulator, and experimental frame in a National Research Council volume.[4]
2000	26	Expanded theory with Prähofer and Kim; related work formalizes limited sensitivity in perceiving agents.[15, 16]
2022–24	2–4	Modern research examples combine language-based reasoning, tools, memory, multi-agent interaction, and executable evaluation environments.[17, 18, 10, 19, 20]

References

- [1] B. P. Zeigler, F. E. Cellier, and J. W. Rozenblit, “Design of a simulation environment for laboratory management by robot organizations,” *Journal of Intelligent and Robotic Systems*, 1(3), 299–309, 1988. doi:10.1007/BF00238771. [Author-hosted original paper](#).
- [2] B. P. Zeigler, *Object-Oriented Simulation with Hierarchical, Modular Models: Intelligent Agents and Endomorphic Systems*. Academic Press, 1990. ISBN 978-0-12-778452-6. [Publisher description and contents](#).
- [3] A. C. Chow and B. P. Zeigler, “Parallel DEVS: A parallel, hierarchical, modular, modeling formalism,” *Proceedings of the Winter Simulation Conference*, 716–722, 1994. [Original proceedings paper](#).
- [4] B. Zeigler, “Components of a theory of modeling and simulation,” Appendix G, in National Research Council, *Technology for the United States Navy and Marine Corps, 2000–2035: Becoming a 21st-Century Force, Vol. 9: Modeling and Simulation*, 211–215. National Academies Press, 1997. doi:10.17226/5869. [Full appendix](#).
- [5] Anthropic, “Trustworthy agents in practice,” April 9, 2026. <https://www.anthropic.com/research/trustworthy-agents>. Accessed September 30, 2026. Primary vendor account.
- [6] OpenAI, “Workspace agents for business,” undated current product page. <https://openai.com/business/workspace-agents/>. Accessed September 30, 2026. Research-preview availability; primary vendor account.
- [7] C. Hewitt, P. Bishop, and R. Steiger, “A universal modular ACTOR formalism for artificial intelligence,” *Proceedings of IJCAI*, 1973. <https://www.ijcai.org/Proceedings/73/Papers/027B.pdf>.
- [8] Y. Shoham, “Agent-oriented programming,” *Artificial Intelligence*, 60(1), 51–92, 1993. doi:10.1016/0004-3702(93)90034-9. [Author bibliography](#).
- [9] M. Wooldridge and N. R. Jennings, “Intelligent agents: Theory and practice,” *The Knowledge Engineering Review*, 10(2), 115–152, 1995. doi:10.1017/S0269888900008122. [Author-hosted paper](#).
- [10] J. S. Park, J. C. O’Brien, C. J. Cai, M. R. Morris, P. Liang, and M. S. Bernstein, “Generative agents: Interactive simulacra of human behavior,” 2023. doi:10.48550/arXiv.2304.03442. [version 2](#), August 6, 2023.
- [11] B. P. Zeigler, *Theory of Modelling and Simulation*. Wiley-Interscience, 1976. [Publisher record](#); see also Ref. [12].
- [12] B. P. Zeigler and S. Vahie, “DEVS formalism and methodology: Unity of conception/diversity of application,” *Proceedings of the Winter Simulation Conference*, 573–579, 1993. [Original proceedings paper](#).
- [13] L. C. Schooley, B. P. Zeigler, F. E. Cellier, and F.-Y. Wang, “High-autonomy control of space resource processing plants,” *IEEE Control Systems*, 13(3), 29–39, June 1993. doi:10.1109/37.214942. [Author-hosted original paper](#).
- [14] J. W. Rozenblit and B. P. Zeigler, “Representing and constructing system specifications using the system entity structure concepts,” *Proceedings of the Winter Simulation Conference*, 604–611, 1993. doi:10.1145/256563.256742. [Original proceedings paper](#).
- [15] B. P. Zeigler, H. Prähofer, and T. G. Kim, *Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems*, 2nd ed. Academic Press, 2000. [Publisher description and contents](#).
- [16] B. P. Zeigler, H. S. Sarjoughian, and H. Prähofer, “Theory of quantized systems: DEVS simulation of perceiving agents,” *Cybernetics and Systems*, 31(6), 611–647, 2000. doi:10.1080/01969720050143175. [Institutional publication record and abstract](#).

- [17] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, “ReAct: Synergizing reasoning and acting in language models,” *ICLR*, 2023; first preprint 2022. [doi:10.48550/arXiv.2210.03629](https://doi.org/10.48550/arXiv.2210.03629). [version 3](#), March 10, 2023.
- [18] T. Schick, J. Dwivedi-Yu, R. Dessì, R. Raileanu, M. Lomeli, L. Zettlemoyer, N. Cancedda, and T. Scialom, “Toolformer: Language models can teach themselves to use tools,” 2023. [doi:10.48550/arXiv.2302.04761](https://doi.org/10.48550/arXiv.2302.04761). [version 1](#), February 9, 2023.
- [19] Q. Wu et al., “AutoGen: Enabling next-gen LLM applications via multi-agent conversation,” 2023. [doi:10.48550/arXiv.2308.08155](https://doi.org/10.48550/arXiv.2308.08155). [version 2](#), October 3, 2023.
- [20] S. Zhou et al., “WebArena: A realistic web environment for building autonomous agents,” *ICLR*, 2024; first preprint 2023. [doi:10.48550/arXiv.2307.13854](https://doi.org/10.48550/arXiv.2307.13854). [version 4](#), April 16, 2024.
- [21] A. Vaswani et al., “Attention is all you need,” *Advances in Neural Information Processing Systems*, 30, 2017. [doi:10.48550/arXiv.1706.03762](https://doi.org/10.48550/arXiv.1706.03762). [Original research record](#).
- [22] Delft University of Technology, “DSOL—Distributed Simulation Object Library,” DSOL demo documentation, version 4.3.2. <https://simulation.tudelft.nl/dsol/docs/latest/dsol-demo/index.html>. Accessed September 30, 2026; sections “What is DSOL?” and “Implemented formalisms.”
- [23] B. P. Zeigler, “Knowledge representation from Newton to Minsky and beyond,” *Applied Artificial Intelligence*, 1(1), 87–107, 1987. [doi:10.1080/08839518708927962](https://doi.org/10.1080/08839518708927962). [Publisher abstract and publication record](#).
- [24] B. P. Zeigler, J. Hu, and J. W. Rozenblit, “Hierarchical, modular modelling in DEVS-Scheme,” *Proceedings of the Winter Simulation Conference*, 84–89, 1989. [doi:10.1145/76738.76749](https://doi.org/10.1145/76738.76749). [Original proceedings paper](#).
- [25] Hamaidi Lyes, P. Bourseau, G. Muratet, J. M. Couretas, and B. P. Zeigler, “DEVS-based endomorphic agents: Control through deliberative and reactive planning,” in *Computer Aided Systems Theory—EUROCAST ’95*, Lecture Notes in Computer Science, 1030, 386–402, 1996. [doi:10.1007/BFb0034775](https://doi.org/10.1007/BFb0034775). [Original preview, pp. 386–387](#).