

DALI2: Evolving a Logic-Based Agent Metainterpreter Towards Neuro-Symbolic Multi-Agent Systems

Lorenzo De Lauretis¹, Giovanni De Gasperis^{1,*} and Stefania Costantini¹

¹Department of Information Engineering, Computer Science and Mathematics (DISIM), University of L'Aquila, Via Vetoio, 67100 L'Aquila, Italy

Abstract

DALI is an agent-oriented logic programming language, built as a Prolog metainterpreter, that has served multi-agent systems research for over two decades. Its design provides formal constructs for reactive and proactive behaviour, temporal event memory, communication filtering, and goal-directed reasoning within a BDI-inspired framework. In this paper we present DALI2, a complete reimplementing of the DALI metainterpreter on SWI-Prolog that preserves the full language semantics while restructuring the interpreter around an explicit, formally characterisable agent reasoning cycle. Beyond the migration from commercial SICStus Prolog to the open-source SWI-Prolog runtime and the consolidation of over 20 source files into 8 modular components, DALI2 introduces new declarative constructs — periodic tasks, condition monitors, learning rules, and a simplified ontology system — that extend the expressiveness of the agent language. A central scientific contribution is the integration of a *neuro-symbolic oracle* that allows DALI2 agents to query large language models (LLMs) from within their symbolic reasoning cycle. The oracle interface, governed by the existing tell/told filtering mechanism, enables controlled information flow between the symbolic agent layer and the sub-symbolic LLM, opening new research directions in grounded neuro-symbolic agent architectures, LLM-augmented planning, and trustworthy hybrid reasoning.

Keywords

agent-oriented logic programming, metainterpreter, neuro-symbolic AI, large language models, multi-agent systems, DALI, BDI agents

1. Introduction

Agent-oriented programming (AOP) [1] provides high-level abstractions — beliefs, goals, plans, reactive rules — for the design of autonomous, distributed systems [2]. Among AOP languages grounded in logic programming, the *DALI* language [3, 4] occupies a distinctive position: it is realised as a Prolog *metainterpreter* [5] that extends standard Prolog with dedicated operators and suffix conventions for external events, internal events, condition-action rules, past-event memory, tell/told communication filtering, FIPA messaging, ontologies, and goal-directed behaviour. Since its introduction, DALI has been applied to cognitive robotics [6], complex event processing [7], industrial fault detection [8], flooding response [9], and formal studies of trustworthy agent behaviour [10].

The metainterpreter design is central to DALI's identity: rather than compiling agent programs into an external runtime, the DALI interpreter *interprets agent rules at the meta-level*, maintaining full access to the Prolog inference engine while adding domain-specific agent reasoning. This architecture enables features difficult to achieve in non-reflective systems — runtime introspection of beliefs and past events, ontology-aware unification, and dynamic rule evaluation within a cyclic sense-reason-act loop.

However, the original implementation has accumulated significant technical debt: dependence on commercial SICStus Prolog [11], a custom character-level tokeniser with file-based intermediate

WOA 2026: 27th Workshop "From Objects to Agents", June 15–17, 2026, Salerno, Italy

*Corresponding author.

✉ lorenzo.delauritis@univaq.it (L. De Lauretis); giovanni.degasperis@univaq.it (G. De Gasperis);

stefania.costantini@univaq.it (S. Costantini)

🌐 <https://www.disim.univaq.it/GiovanniDeGasperis> (G. De Gasperis); <https://www.disim.univaq.it/StefaniaCostantini> (S. Costantini)

🆔 0000-0002-8796-3672 (L. De Lauretis); 0000-0001-9521-4711 (G. De Gasperis); 0000-0002-5686-6124 (S. Costantini)



© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

processing, over 20 tightly coupled source files, and TCP-only Linda communication. These factors have hindered adoption, reproducibility, and extensibility.

In this paper we present **DALI2**¹ [12], a ground-up reimplementaion that addresses these limitations while preserving identical DALI syntax and semantics. Beyond the engineering modernisation, DALI2 makes two scientific contributions:

1. It restructures the agent reasoning cycle into an **explicit, formally characterisable 13-phase loop** (Section 3), making the operational semantics of DALI agents transparent and amenable to formal analysis.
2. It introduces a **neuro-symbolic oracle interface** (Section 5) that allows agents to query large language models from within their symbolic reasoning cycle, governed by DALI’s tell/told mechanism, opening new research directions in grounded neuro-symbolic agent architectures [13].

Section 2 reviews the DALI language and its theoretical foundations. Section 3 presents the DALI2 agent reasoning cycle. Section 4 details language evolution and new constructs. Section 5 discusses the neuro-symbolic oracle and its implications. Section 6 summarises key implementation aspects. Section 7 provides a systematic comparison. Section 8 discusses related work. Section 9 concludes.

2. Background: DALI as a Logic-Based Agent Metainterpreter

2.1. Theoretical Foundations

DALI (“Dynamic Agent, Logic-based, with an Inferential capability”) was introduced by Costantini and Tocchio [3, 4, 14] as a logic programming language for BDI-style agents [15]. Its theoretical underpinnings draw on the tradition of Prolog metainterpreters: a DALI agent program is a set of extended Prolog clauses that are *interpreted* by the DALI meta-level engine rather than executed by the standard Prolog resolution mechanism alone.

The key insight is that different *categories* of agent behaviour — reactivity, proactivity, deliberation, communication — can each be captured by a distinct syntactic class of clauses, differentiated through custom operators and suffix conventions. The metainterpreter orchestrates these classes within a cyclic execution model, providing a principled mapping from the BDI conceptual model to executable logic programs. The declarative semantics of this decomposition has been studied in [16].

Formally, a DALI agent program $\mathcal{A}$ can be decomposed as:

$$\mathcal{A} = \langle \mathcal{R}, \mathcal{I}, \mathcal{C}, \mathcal{P}, \mathcal{M}, \mathcal{X}, \mathcal{K}, \mathcal{G}, \mathcal{B}, \mathcal{T}, \mathcal{O} \rangle$$

where $\mathcal{R}$ denotes reactive rules, $\mathcal{I}$ internal events, $\mathcal{C}$ condition–action rules, $\mathcal{P}$ present-event rules, $\mathcal{M}$ multi-event rules, $\mathcal{X}$ export-past rules, $\mathcal{K}$ constraints, $\mathcal{G}$ goal rules, $\mathcal{B}$ initial beliefs, $\mathcal{T}$ tell/told filtering rules, and $\mathcal{O}$ ontological declarations.

2.2. DALI Language Constructs

The language extends Prolog with the following syntactic categories:

- **Reactive rules** ($:>$, E suffix): `eventE(X) :> Body`. fires when external event `event(X)` is perceived.
- **Internal events** (I suffix + `internal_event/5`): proactive rules with configurable firing conditions.
- **Condition–action rules** ($:<$): *edge-triggered* — fire once when a condition transitions from false to true.
- **Present events** (N suffix): *level-triggered* environment observations.

¹<https://github.com/AAAI-DISIM-UnivAQ/DALI2>

- **Export-past rules** ($\sim/$, $</$, $?/$): rules that *consume* past events.
- **Constraints** ($\sim$): invariant conditions checked every cycle.
- **Goals**: `obt_goal` (persistent) and `test_goal` (one-shot).
- **Tell/told filtering**: declarative rules controlling message acceptance and sending, with priority-based ordering.
- **FIPA performatives** [17]: `confirm`, `disconfirm`, `propose`, `query_ref`, with extensible handling for additional performatives.
- **Ontology**: semantic matching via OWL files and `meta/3` predicates.

All of these constructs are **fully preserved** in DALI2 with identical syntax.

2.3. Limitations of the Original Implementation

The original SICStus-based implementation presents several obstacles to modern research use: (i) dependence on a commercial runtime, precluding open-source reproducibility; (ii) a monolithic codebase (>20 files, 86 KB main file); (iii) a custom tokeniser performing character-level variable manipulation via temporary files on disk; (iv) TCP-only communication through a Linda tuple space; (v) no integrated monitoring or introspection tools. These factors impede both adoption and the use of DALI as a platform for new agent-theoretic research.

3. The DALI2 Agent Reasoning Cycle

A fundamental contribution of DALI2 is the restructuring of the agent execution model into an **explicit 13-phase reasoning cycle**. In the original DALI, the execution order was implicitly encoded within the control flow of the monolithic interpreter. In DALI2, each phase is a distinct, named step of the metainterpreter, making the operational semantics transparent and amenable to formal characterisation.

3.1. Formal Characterisation

Let the state of a DALI2 agent at cycle n be described by:

$$\sigma_n = \langle \mathcal{B}_n, \mathcal{H}_n, \mathcal{L}_n, \mathcal{G}_n, \mathcal{Q}_n \rangle$$

where $\mathcal{B}_n$ is the belief base, $\mathcal{H}_n$ the past-event history (with timestamps), $\mathcal{L}_n$ the learned associations, $\mathcal{G}_n$ the active goals (including residue goals), and $\mathcal{Q}_n$ the incoming message queue. The transition $\sigma_n \rightarrow \sigma_{n+1}$ is effected by 13 phases in strict sequence:

1. **Message processing.** Dequeue from $\mathcal{Q}_n$, filtered and sorted by `told` rules. Evaluate matching reactive rules; record in $\mathcal{H}$; trigger learning.
2. **Injected event processing.** Process externally injected events via the same reactive-rule mechanism.
3. **Internal event processing.** For each $e_I \in \mathcal{I}$ whose firing conditions are met (interval, guard, time window), evaluate the handler and record in $\mathcal{H}$.
4. **Periodic tasks.** Fire each `every(N, Goal)` whose interval has elapsed.
5. **Condition monitoring.** Evaluate `when(Cond)` rules; fire for every satisfied condition (*level-triggered*).
6. **Condition-action evaluation.** Track truth values across cycles; fire on `false`→`true` transition (*edge-triggered*).
7. **Present-event evaluation.** Evaluate present-event rules against observable state.
8. **Multi-event evaluation.** Check conjunctive patterns in $\mathcal{M}$ against $\mathcal{H}$.
9. **Export-past evaluation.** Match and consume past events per $\mathcal{X}$.
10. **Constraint checking.** Verify $\mathcal{K}$; execute handler if violated.

11. **Goal processing.** Attempt achieve-goals and test-goals; re-attempt residue goals.
12. **Past-event maintenance.** Expire events per `past_event / 2`; promote to remember tier; enforce limits.
13. **Cycle sleep.** Wait for cycle duration before $n+1$.

This ordering ensures: (a) external stimuli are processed before internal deliberation; (b) beliefs updated by reactive rules are visible to subsequent checks; (c) constraints verify invariants after all event-processing, monitoring, and condition-evaluation phases, before goal-directed deliberation proceeds; (d) temporal reasoning remains consistent throughout the cycle.

3.1.1. Phase Transition Rules

We sketch transition rules for selected phases to illustrate the formal structure. Let $\sigma = \langle \mathcal{B}, \mathcal{H}, \mathcal{L}, \mathcal{G}, \mathcal{Q} \rangle$ and $\mathcal{T}_{told}$ denote the agent's told-filtering rules.

Message processing (ϕ_1). For each message $m = (e, s) \in \mathcal{Q}$ from sender s with content e :

$$\frac{m \in \mathcal{Q} \quad \exists \text{told}(_, P, p) \in \mathcal{T}_{told}. P \sqsupseteq e \quad r = (P' :> B) \in \mathcal{R}, P' \sqsupseteq e}{\mathcal{Q}' = \mathcal{Q} \setminus \{m\}, \quad \mathcal{H}' = \mathcal{H} \cup \{(received(e, s), t)\}, \quad eval(B\theta)}$$

where $P \sqsupseteq e$ denotes that pattern P subsumes e , and θ is the resulting substitution. Messages not matching any told rule are rejected (closed-world filtering).

Condition-action (ϕ_6). Let $st_n(c)$ record the truth value of condition c at cycle n :

$$\frac{c \in \mathcal{C} \quad st_{n-1}(c) = false \quad \sigma \models c}{st_n(c) := true, \quad eval(action(c))} \quad \frac{\sigma \not\models c}{st_n(c) := false}$$

This captures the edge-triggered semantics: the action fires only on the *false* $\rightarrow$ *true* transition.

Constraint checking (ϕ_{10}).

$$\frac{k = (C : \sim H) \in \mathcal{K} \quad \sigma \not\models C}{eval(H)}$$

If $H = true$ (no handler), the violation is logged but does not halt the agent. Constraints thus function as declarative invariant monitors with optional recovery actions.

3.2. Temporal Reasoning via Past-Event Memory

A distinguishing feature of DALI is its structured past-event memory $\mathcal{H}$, which supports *time-bounded temporal reasoning* within a logic programming framework. Each event e that occurs at time t is recorded as a pair $(wrap(e), t)$ where the wrapper encodes the event's provenance: *received*($e, sender$), *injected*(e), *internal*(e), *did*(e) (action performed), *confirmed*(e) (FIPA confirmation), or *goal_achieved*(e). This provenance-in-term encoding allows pattern matching on event history to discriminate by source without additional indexing.

DALI2 preserves the two-tier temporal memory model:

- **Past tier:** events persist for a configurable duration (`past_event(Pattern, Duration)`).
- **Remember tier:** expired past events are promoted here (`remember_event(Pattern, Duration)`), with optional cardinality limits (`remember_event_mod`).

This mechanism allows agents to reason about *what has happened* within configurable time horizons — a capability absent from most BDI platforms. Export-past rules ($\sim /, < /, ? /$) further enable *event consumption*, modelling one-shot reactions to historical patterns.

3.3. Edge-Triggered vs. Level-Triggered Semantics

DALI2 provides a clear formal distinction between:

- **Level-triggered** constructs (when, present events): fire *every cycle* while the condition holds. Suited for continuous monitoring.
- **Edge-triggered** constructs (condition-action :<): fire *once* on the false→true transition. Suited for state-change reactions.

This distinction, often conflated in other agent platforms, is syntactically explicit in DALI2 and formally grounded in the cycle semantics.

4. Language Evolution: New Constructs

DALI2 preserves *all* original DALI constructs with identical syntax while introducing new declarative features that extend the language's expressiveness.

4.1. Single-File Multi-Agent Definitions

The original DALI required separate type files and instance files assembled by shell scripts. DALI2 replaces this with a single .pl file where :- agent(Name, Options). directives demarcate agent boundaries:

Listing 1: Multi-agent single-file format

```
:- agent(sensor, [cycle(1)]).
readingE(V) :>
    log("Sensor: ~w", [V]),
    send(analyzer, data(V)).

:- agent(analyzer, [cycle(2)]).
dataE(V) :>
    ( V > 80 -> send(alert, overheating(V))
    ; assert_belief(last_reading(V)) ).
```

The loader parses this file, associates each rule with its nearest preceding agent directive, and stores typed definitions that the engine retrieves at runtime.

4.2. Periodic Tasks and Condition Monitors

Two new constructs address common agent patterns that previously required workarounds:

Listing 2: Periodic tasks and condition monitors

```
%% Fire every 30 seconds (no DALI equivalent)
every(30) :- send(dashboard, health_check).

%% Level-triggered: fire every cycle while true
when(believes(temperature(T)), T > 30) :-
    send(ac_controller, cool_down).
```

every/2 provides fixed-interval scheduling independent of the agent cycle. when/1-2 complements condition-action rules (:<) by providing level-triggered monitoring.

4.3. Declarative Learning Rules

The original DALI learning module performed character-level file manipulation to record learned associations. DALI2 replaces this with declarative learn_from/2 rules:

Listing 3: Declarative learning

```
learn_from(read_temp(T), overheating) :- T > 80.  
learn_from(read_temp(T), normal) :- T =< 80.
```

When an event matching the pattern occurs, the agent records the association in $\mathcal{L}$, queryable via `learned(Pattern, Outcome)`. This approach is both more principled (associations are defined declaratively alongside other agent rules) and more maintainable.

4.4. Simplified Ontology System

The original meta/3 ontology mechanism involved nested OWL lookups with repository URLs. DALI2 replaces this with lightweight Prolog-native declarations:

Listing 4: Ontology declarations

```
ontology(same_as(hot, warm)).  
ontology(eq_property(temperature, temp)).  
ontology(symmetric(connected_to)).
```

These are applied transparently during event matching and belief checking via `onto_match/2`, preserving the semantic power while eliminating the OWL dependency.

4.5. Enhanced Internal Events

DALI2 extends internal event configuration with composable options:

- `forever / times(N)`: repetition control (unbounded or bounded).
- `interval(N)`: minimum seconds between consecutive firings.
- `trigger(Condition)`: guard predicate that must hold for the event to fire.
- `until(Condition)`: stop condition — the event ceases when the condition becomes true.
- `change([FactList])`: resets the firing counter when any fact in the list changes, enabling re-triggering upon belief updates.
- `between(time(H1,M1), time(H2,M2))`: restricts firing to a daily time window.
- `between(date(Y1,M1,D1), date(Y2,M2,D2))`: restricts firing to a calendar date range.

All options are combinable in lists, and a single internal event can simultaneously carry, e.g., an interval constraint, a guard, and a time window.

4.6. Shared Blackboard Predicates

DALI2 exposes the Redis-backed shared blackboard directly to agent rules via three predicates: `bb_write(Tuple)` to publish a tuple, `bb_read(Pattern)` to query by unification, and `bb_remove(Pattern)` to retract a matching tuple. These replace the original Linda-style `out/in/rd` operations with a simpler interface backed by a Redis SET, enabling cross-agent coordination through a persistent shared memory visible to all agents in the cluster.

4.7. Helpers and Simplified Predicates

Utility predicates (`helper(Head) :- Body.`) provide a clean namespace for auxiliary computations. Additionally, DALI2 introduces concise aliases: `send(D, E)` for `messageA(D, send_message(E, Me))`, `has_past(E)` for `evp(E)`, `achieve(G)` for `tenta_residuo(G)`, and `believes(F)` for belief checking. Both original and new styles are interchangeable, ensuring full backward compatibility.

Notably, the optional agent prefix syntax (e.g., `sensor:eventE(X) :-> Body.`) is preserved for explicit agent qualification, while the prefix-free style using `:- agent(Name).` directives to set the current agent context is recommended for readability.

5. Neuro-Symbolic Agent Reasoning via LLM Integration

The most scientifically significant new feature in DALI2 is its *AI oracle*: a built-in interface that allows agents to consult large language models (LLMs) from within their symbolic reasoning cycle. This creates a **neuro-symbolic agent architecture** where symbolic Prolog-based reasoning and sub-symbolic neural inference coexist within a single agent, mediated by the existing communication-filtering mechanism.

5.1. The Neuro-Symbolic Gap in Agent Systems

The dichotomy between symbolic and sub-symbolic AI — what Minsky termed the “neat vs. scruffy” divide [18] — has been a persistent tension in AI research. Symbolic systems excel at formal reasoning, explainability, and compositional generalisation, while neural systems excel at pattern recognition, natural language understanding, and generalisation from data.

Recent work on neuro-symbolic AI [13] argues that the next generation of AI systems must integrate both paradigms. In the agent domain, this challenge is particularly acute: LLM-based agents [19, 20] have demonstrated impressive natural-language capabilities but lack the formal reasoning guarantees of logic-based systems, while traditional BDI agents offer formal properties but limited ability to handle unstructured knowledge.

DALI2’s oracle provides a principled bridge between these paradigms *within the framework of an established agent-oriented logic programming language*.

5.2. The Oracle Interface

The AI oracle is accessed via `ask_ai/2-3`, callable from any rule body:

Listing 5: Oracle usage within symbolic reasoning

```
:- agent(crop_advisor, [cycle(2)]).

soil_reportE(Moisture, PH, Field) :-
    %% Symbolic reasoning first
    ( Moisture < 30 ->
        Action = irrigate
    ; Moisture > 80 ->
        Action = reduce_water
    ;
        %% Consult LLM for ambiguous cases
        ( ai_available ->
            ask_ai(
                soil_analysis(m(Moisture), ph(PH)),
                Advice),
            Action = Advice
        ; Action = monitor )
    ),
    send(controller, action(Action, Field)).
```

The key design decisions are:

1. **Prolog-term grounding.** The LLM is prompted to return a single valid Prolog fact. The oracle module parses the response into a Prolog term, *grounding* the sub-symbolic output into the symbolic domain. If parsing fails, a fallback term is returned, ensuring the agent’s reasoning is never blocked by malformed LLM output.
2. **Context as Prolog terms.** The query context is a Prolog term (e.g., `soil_analysis(m(25), ph(6.5))`), not free-form text. This ensures the symbolic structure of the agent’s knowledge is preserved across the neuro-symbolic boundary.

3. **Tell/told filtering.** Crucially, the oracle interface is subject to the same tell/told filtering as inter-agent communication. Tell rules control what context the agent may send to the oracle; told rules control what responses the agent is willing to accept. This enables **information-flow governance** over the neuro-symbolic boundary.

5.3. Tell/Told as Neuro-Symbolic Trust Mechanism

The reuse of tell/told filtering for oracle communication is a non-trivial design choice with significant theoretical implications. In the original DALI, tell/told rules were designed to enforce *social trust* between agents [10] — an agent could declare which information sources it trusts (told) and what information it is willing to share (tell).

In DALI2, this same mechanism governs the agent’s relationship with an LLM:

Listing 6: Tell/told filtering applied to oracle

```
:- agent(medical_advisor, [cycle(2)]).

%% Only send anonymised patient data to oracle
tell(_, _, patient_summary(_)) :- true.

%% Only accept drug_interaction or dosage advice
told(_, suggestion(drug_interaction(_)), 100) :- true.
told(_, suggestion(dosage(_,_)), 80) :- true.

%% Reject any other oracle response
analyzeE(PatientData) :-
    ask_ai(patient_summary(PatientData), Response),
    %% Response is filtered by told rules:
    %% accepted suggestions are processed,
    %% unmatched responses are rejected
    log("Oracle filtered response: ~w", [Response]).
```

This gives the agent programmer *declarative control* over:

- What information flows to the LLM (preventing sensitive data leakage).
- What types of LLM outputs the agent is willing to incorporate into its reasoning (preventing hallucination injection).
- The *priority* of oracle-derived information relative to other sources.

5.4. Implications for Agent-Theoretic Research

The integration of an LLM oracle into a formal agent metainterpreter opens several research directions:

Hybrid planning. Kambhampati et al. [21] argue that LLMs “can’t plan, but can help planning.” DALI2 provides a concrete realisation of the LLM-Modulo framework: the agent’s goal-directed reasoning (via `obt_goal/test_goal`) remains in the symbolic layer, while the LLM can be consulted for *sub-goal suggestions* or *domain heuristics* that inform plan selection. The told filtering ensures that only well-formed, type-compatible suggestions enter the agent’s deliberation.

Grounded neuro-symbolic reasoning. Unlike pure LLM-based agents [19, 22], DALI2 agents maintain *formally grounded* beliefs, past events, and goal states. The LLM oracle is a *consulted resource*, not the reasoning engine. This preserves properties like belief consistency, constraint satisfaction, and temporal-memory integrity that are lost in purely neural architectures.

Explainability. Since the agent’s reasoning cycle is symbolic and the oracle’s influence is mediated by tell/told filtering, the agent’s decision process remains *inspectable*: one can trace which oracle responses were accepted, which were rejected, and how they influenced the agent’s beliefs and actions.

Multi-agent neuro-symbolic coordination. In a MAS, different agents can have different oracle configurations (different LLM models, different filtering rules). This enables heterogeneous neuro-symbolic societies where agents with varying levels of AI augmentation interact through FIPA protocols, with trust mediated by the existing communication-filtering infrastructure.

Experiential learning from oracle interaction. Combined with DALI2’s `learn_from` mechanism, agents can learn to *predict* oracle responses over time, gradually building symbolic approximations of the LLM’s behaviour:

Listing 7: Learning from oracle interactions

```
learn_from(oracle_response(Context, Suggestion),
           reliable) :-
    has_past(outcome_verified(Context, positive)).

learn_from(oracle_response(Context, Suggestion),
           unreliable) :-
    has_past(outcome_verified(Context, negative)).
```

This creates a feedback loop where the symbolic agent develops its own model of when to trust the sub-symbolic oracle — a form of *meta-learning about neural capabilities*.

6. Implementation

DALI2 is implemented as 8 SWI-Prolog [23] modules: `server.pl` (HTTP server and REST API), `engine.pl` (core metainterpreter), `agent_process.pl` (agent-side execution in child processes), `loader.pl` (DALI syntax parser), `communication.pl` (message-passing abstraction), `redis_comm.pl` (Redis [24] pub/sub and blackboard), `blackboard.pl` (thread-safe local tuple space), and `ai_oracle.pl` (LLM integration via OpenRouter [25]).

6.1. Key Design Decisions

Open-source runtime. The migration from SICStus Prolog [11] to SWI-Prolog eliminates the commercial-licence barrier. SWI-Prolog’s built-in libraries for HTTP, JSON, Redis, and process management replace external dependencies.

Elimination of the custom tokeniser. The original DALI’s three-file tokeniser (`tokefun.pl`, `togli_var.pl`, `metti_var.pl`) performed character-level variable manipulation via temporary files. DALI2 eliminates this entirely: DALI operators (`:>`, `:<`, `~/`, `</`, `?/`, `:~`) are defined as standard Prolog operators, and SWI-Prolog’s native `read_term/3` parses agent files directly.

Redis star topology. Inter-agent communication uses Redis with two pub/sub channels — LINDA (messages encoded as `TO:CONTENT:FROM`) and LOGS (monitoring) — plus a Redis SET (BB) serving as a shared blackboard that replaces the original Linda tuple space. This naturally supports single-machine, multi-terminal, and multi-machine deployment without code changes.

Process-per-agent model. Each agent runs as a separate OS process, providing strong isolation and individual lifecycle control.

6.2. Codebase Comparison

6.3. Deployment and Tooling

DALI2 includes a built-in web dashboard (agent inspection, log viewer, event injection, source editor, AI oracle panel), a comprehensive REST API (20+ endpoints), and Docker support for single-command

Table 1

Codebase complexity comparison

Metric	DALI	DALI2
Source files	>20	8
Largest file	86 KB	55 KB
Custom tokeniser	Yes (3 files)	None
Temporary file I/O	Yes	None
External UI/scripts	Yes	Built-in

Table 2

Feature comparison: DALI vs. DALI2

Aspect	DALI (SICStus)	DALI2 (SWI-Prolog)
<i>Theoretical / Language</i>		
Agent reasoning cycle	Implicit in control flow	Explicit 13-phase formal cycle
Event syntax	<code>eventE(X) :- Body.</code>	Identical
Temporal memory	Past + remember tiers	Identical
Edge/level-triggered	Both (implicit distinction)	Both (explicit, documented)
Learning	File-based module	Declarative <code>learn_from/2</code>
Ontology	OWL + meta/3	<code>ontology/1</code> (Prolog-native)
LLM integration	—	<code>ask_ai/2-3</code> (neuro-symbolic oracle)
Periodic tasks	—	<code>every(N, Goal)</code>
Condition monitors	—	<code>when(Cond) :- Body</code>
Helpers	—	<code>helper(Head) :- Body</code>
Shared blackboard	Linda <code>out/in/rd</code>	<code>bb_write/bb_read/bb_remove</code> (Redis SET)
<i>Implementation</i>		
Runtime	SICStus (commercial)	SWI-Prolog (open-source)
Source files	>20	8 modules
Agent definition	Types + instances + scripts	Single <code>.pl</code> file
Communication	TCP/Linda	Redis pub/sub + SET (star topology)
Tokeniser	Custom (3 files)	None (native parsing)
Monitoring	External (Python)	Built-in web UI + REST API

deployment. These replace the original DALI’s shell-script assembly, separate Python UI, and manual process management.

7. Systematic Comparison

Table 2 provides a feature comparison highlighting both engineering and scientific dimensions.

8. Related Work

Agent-oriented logic programming. Jason [26] implements AgentSpeak(L) on Java, providing BDI reasoning with plan libraries. Unlike DALI, Jason does not offer past-event memory, export-past rules, or tell/told filtering. The DALI metainterpreter approach [3, 4, 16] provides richer temporal reasoning and communication control.

LLM-based agents. Recent surveys [19, 20] document the rise of LLM-based autonomous agents. These systems typically use LLMs as the primary reasoning engine, augmented with tool-use and memory modules. Sumers et al. [22] propose cognitive architectures for language agents. However, these systems lack the formal guarantees of logic-based approaches: belief consistency, constraint satisfaction, and temporal-memory integrity are not formally ensured.

Neuro-symbolic AI. Garcez and Lamb [13] characterise neuro-symbolic AI as the “third wave” combining neural pattern recognition with symbolic reasoning. DALI2’s oracle realises this vision within an agent framework: the symbolic layer (Prolog metainterpreter) maintains formal properties while the neural layer (LLM) provides domain knowledge and natural-language capabilities.

LLM-Modulo frameworks. Kambhampati et al. [21] argue that LLMs should serve as *idea generators* within a symbolic verification loop. DALI2’s architecture naturally implements this pattern: the oracle suggests actions, the agent’s constraint-checking and goal-processing phases verify them symbolically. Hao et al. [27] explore reasoning with language models as planning with world models, a perspective compatible with DALI2’s integration of LLM suggestions into the goal-processing phase.

Trustworthy agents. Costantini [10] established the theoretical basis for trustworthy agent behaviour in DALI. DALI2 extends this to the neuro-symbolic setting: tell/told rules provide a declarative, verifiable mechanism for governing the agent–LLM trust boundary.

9. Conclusion and Future Work

We have presented DALI2, a reimplementaion of the DALI agent-oriented logic programming language as a modular SWI-Prolog metainterpreter. The contributions span three dimensions:

1. **Formal.** An explicit 13-phase agent reasoning cycle that makes operational semantics transparent, with clear distinctions between edge-triggered and level-triggered constructs, and a formally characterised temporal memory model.
2. **Language.** New declarative constructs (periodic tasks, condition monitors, learning rules, simplified ontologies, helper predicates, shared blackboard predicates) that extend DALI’s expressiveness while maintaining full backward compatibility.
3. **Neuro-symbolic.** An LLM oracle interface governed by tell/told filtering, enabling controlled information flow between symbolic agent reasoning and sub-symbolic neural inference — a concrete realisation of neuro-symbolic agent architecture.

The engineering improvements (open-source runtime, 8-module codebase, Redis communication, web dashboard) lower the barrier to adoption and reproducibility.

Future work includes several directions:

- **Formal verification** of the semantic equivalence between DALI and DALI2 execution models.
- **LLM-augmented plan synthesis:** using the oracle within `obt_goal` to generate candidate plans that are then verified symbolically.
- **Belief revision via oracle:** allowing the LLM to suggest belief updates subject to constraint checking and consistency verification.
- **Multi-agent argumentation** [28]: exploring how oracle-augmented agents can engage in structured argumentation, with LLM suggestions serving as defeasible arguments.
- **Empirical evaluation:** benchmarking oracle-augmented vs. purely symbolic agents in complex domains (emergency response, precision agriculture) to quantify the benefits of neuro-symbolic integration.

DALI2 is available as open-source software under the Apache License 2.0.

Acknowledgments

The authors acknowledge the foundational work on DALI by Stefania Costantini and Arianna Tocchio, which provided the language design and semantics that DALI2 preserves and extends.

Declaration on Generative AI

During the preparation of this work, the authors used AI-assisted tools for code development support. The authors reviewed and edited all content and take full responsibility for the publication's content.

References

- [1] Y. Shoham, Agent-oriented programming, *Artificial Intelligence* 60 (1993) 51–92. doi:10.1016/0004-3702(93)90034-9.
- [2] M. Wooldridge, N. R. Jennings, Intelligent agents: Theory and practice, *The Knowledge Engineering Review* 10 (1995) 115–152. doi:10.1017/S0269888900008122.
- [3] S. Costantini, A. Tocchio, A logic programming language for multi-agent systems, in: *Logics in Artificial Intelligence (JELIA 2002)*, volume 2424 of *Lecture Notes in Computer Science*, Springer, 2002, pp. 1–13. doi:10.1007/3-540-45757-7_1.
- [4] S. Costantini, A. Tocchio, The DALI logic programming agent-oriented language, in: *Logics in Artificial Intelligence (JELIA 2004)*, volume 3229 of *Lecture Notes in Computer Science*, Springer, 2004, pp. 685–688. doi:10.1007/978-3-540-30227-8_57.
- [5] L. Sterling, E. Y. Shapiro, *The Art of Prolog: Advanced Programming Techniques*, 2nd ed., MIT Press, 1994.
- [6] S. Costantini, G. De Gasperis, G. Nazzicone, DALI for cognitive robotics: Principles and prototype implementation, in: *International Symposium on Practical Aspects of Declarative Languages (PADL 2017)*, volume 10137 of *Lecture Notes in Computer Science*, Springer, 2017, pp. 152–162. doi:10.1007/978-3-319-51676-9_10.
- [7] S. Costantini, G. De Gasperis, V. Pitoni, A. Salutari, DALI: A multi agent system framework for the web, cognitive robotic and complex event processing, in: *Proceedings of the 32nd Italian Conference on Computational Logic (CILC 2017)*, volume 1949 of *CEUR Workshop Proceedings*, 2017, pp. 286–300. URL: <http://ceur-ws.org/Vol-1949/CILCpaper05.pdf>.
- [8] V. Bevar, S. Costantini, A. Tocchio, G. De Gasperis, A multi-agent system for industrial fault detection and repair, in: *Advances on Practical Applications of Agents and Multi-Agent Systems (PAAMS 2012)*, volume 7209 of *Lecture Notes in Computer Science*, Springer, 2012, pp. 47–55. doi:10.1007/978-3-642-28786-2_5.
- [9] A. Rafanelli, S. Costantini, G. De Gasperis, A multi-agent-system framework for flooding events, in: *WOA 2022: 23rd Workshop From Objects to Agents*, volume 3261 of *CEUR Workshop Proceedings*, 2022, pp. 142–151. URL: <https://ceur-ws.org/Vol-3261/paper11.pdf>.
- [10] S. Costantini, Ensuring trustworthy and ethical behaviour in intelligent logical agents, *Journal of Logic and Computation* 32 (2022) 443–478. doi:10.1093/logcom/exab091.
- [11] Swedish Institute of Computer Science, SICStus Prolog User's Manual, SICS, 2020. URL: <https://sicstus.sics.se/>.
- [12] AAI-DISIM-UnivAQ, AAI-DISIM-UnivAQ/DALI2, 2026. URL: <https://doi.org/10.5281/zenodo.19858728>. doi:10.5281/zenodo.19858728.
- [13] A. d. Garcez, L. C. Lamb, Neurosymbolic AI: The 3rd wave, *Artificial Intelligence Review* 56 (2023) 12387–12406. doi:10.1007/s10462-023-10448-w.
- [14] S. Costantini, A. Tocchio, DALI: An architecture for intelligent logical agents, in: *AAAI Spring Symposium: Emotion, Personality, and Social Behavior*, 2008, pp. 13–18.
- [15] A. S. Rao, M. P. Georgeff, BDI agents: From theory to practice, in: *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS 1995)*, 1995, pp. 312–319.
- [16] S. Costantini, A. Tocchio, About declarative semantics of logic-based agent languages, in: *Declarative Agent Languages and Technologies III*, volume 3904 of *Lecture Notes in Computer Science*, Springer, 2006, pp. 106–123. doi:10.1007/11691792_7.
- [17] Foundation for Intelligent Physical Agents, FIPA ACL message structure specification, 2002. URL: <http://www.fipa.org/specs/fipa00061/SC00061G.html>.

- [18] M. Minsky, Logical versus analogical or symbolic versus connectionist or neat versus scruffy, *AI Magazine* 12 (1991) 34–51.
- [19] Z. Xi, W. Chen, X. Guo, W. He, Y. Ding, B. Hong, M. Zhang, J. Wang, S. Jin, E. Zhou, R. Zheng, X. Fan, X. Wang, L. Xiong, Y. Zhou, W. Wang, C. Jiang, Y. Zou, X. Liu, Z. Yin, S. Dou, R. Weng, W. Qin, Y. Zheng, X. Qiu, X. Huang, Q. Zhang, T. Gui, The rise and potential of large language model based agents: A survey, *Science China Information Sciences* 68 (2023). doi:10.1007/s11432-024-4222-0.
- [20] L. Wang, C. Ma, X. Feng, Z. Zhang, H. Yang, J. Zhang, Z. Chen, J. Tang, X. Chen, Y. Lin, W. X. Zhao, Z. Wei, J. Wen, A survey on large language model based autonomous agents, *Frontiers of Computer Science* 18 (2024). doi:10.1007/s11704-024-40231-1.
- [21] S. Kambhampati, K. Valmeekam, L. Guan, K. Stechly, M. Verma, S. Munos, R. Kapoor, L. Marquez, Llms can't plan, but can help planning in LLM-modulo frameworks, *Forty-first International Conference on Machine Learning* (2024).
- [22] T. R. Sumers, S. Yao, K. Narasimhan, T. L. Griffiths, Cognitive architectures for language agents, *Transactions on Machine Learning Research* (2024).
- [23] J. Wielemaker, T. Schrijvers, M. Triska, T. Lager, *SWI-Prolog, Theory and Practice of Logic Programming* 12 (2012) 67–96. doi:10.1017/S1471068411000494.
- [24] Redis Ltd., Redis – the real-time data platform, 2024. URL: <https://redis.io/>.
- [25] OpenRouter, OpenRouter – unified API for LLMs, 2024. URL: <https://openrouter.ai/>.
- [26] R. H. Bordini, J. F. Hübner, M. Wooldridge, *Programming Multi-Agent Systems in AgentSpeak using Jason*, John Wiley & Sons, 2007.
- [27] S. Hao, Y. Gu, H. Ma, J. Hong, Z. Wang, D. Wang, Z. Hu, Reasoning with language model is planning with world model, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing* (2023) 8154–8173. doi:10.18653/v1/2023.emnlp-main.507.
- [28] P. M. Dung, On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games, *Artificial Intelligence* 77 (1995) 321–357. doi:10.1016/0004-3702(94)00041-X.