Bridging Cognitive Robotics and Generative AI Protocols

Matthias Scheutz 1, and Evan Krause 1

1Department of Computer Science, Tufts University, Medford, MA 02155, USA

Abstract To overcome limitations of individual generative AI models, recent multi-agent systems methods have been introduced for how generative AI agents can invoke external functions, so-called “tool calls”, via the Model Context Protocol (MCP), and communicate with other agents, e.g., via Agent2Agent (A2A). We analyze the relationship between these paradigms and existing methods for developing distributed cognitive architectures as illustrated by the DIARC cognitive robotic architecture. We show how exposing DIARC actions as MCP tools and using DIARC's TRADE middleware as a substrate for A2A-style communication enables integration of cognitive robotic architectures into agentic frameworks. This serves a dual purpose: it allows AI agents to utilize existing cognitive functions and architectures while providing cognitive architectures with generative AI modules in a systematic way. We further argue that semantic action representations and reflective middleware monitoring provide key ingredients for safer and more introspectable multi-agent systems, especially in mixed-initiative and embodied domains, because they allow cognitive systems to check, mediate, and supervise the interactions among agentic components rather than merely relaying them.

Keywords Cognitive Architectures, TRADE middleware, MCP tool calls, A2A protocol

1 Introduction

Generative AI (genAI) systems have made rapid progress in open-domain language understanding, synthesis, and problem decomposition (Yao et al. 2023; Schick et al. 2023). Yet, stand-alone models remain fundamentally limited: They lack perceptual grounding in the physical world, have no direct access to evolving

world state, execute multi-step procedures unreliably, and provide little transparency into how a decision pipeline arrived at a particular action. These limitations have driven two complementary developments in the genAI ecosystem. The first is the introduction of structured *tool calls* that allow language models to invoke external capabilities such as retrieval, database access, web services, or actuation (OpenAI 2023; Schick et al. 2023). The second is the emergence of explicit *multi-agent coordination* mechanisms that support task decomposition, delegation, and parallel specialization across multiple model instances or heterogeneous agents (Anthropic 2024; Google 2025).

Both developments have recently begun to crystallize into open protocols. The Model Context Protocol (MCP) has emerged as a common pattern for tool discovery and invocation through typed schemas and structured request/response interfaces (Anthropic 2024), while the Agent2Agent (A2A) protocol aims to standardize agent-level capability discovery, task negotiation, and long-running structured exchanges among autonomous agents (Google 2025). These protocols substantially improve interoperability among genAI components. What they do not provide, however, is any guarantee about *action semantics*: An MCP tool schema specifies the syntactic shape of an invocation, but says nothing about the conditions under which the underlying operation is applicable, what effects it will have on the world, or how its execution should be monitored and, if necessary, interrupted. In embodied settings, where tool calls move physical actuators near humans, this gap is not just a technicality, but a safety-critical deficiency.

Cognitive robotic architectures, by contrast, have addressed exactly these questions for decades. They provide explicit representations of action structure, world-state constraints, and monitored execution loops that make it possible to reason about actions before, during, and after their execution. In the DIARC architecture (Scheutz et al. 2019), actions are not mere API endpoints; rather, they encode symbolic roles, preconditions, effects, and execution bindings that support planning, failure diagnosis, and explanation. DIARC's reflective middleware TRADE (Scheutz 2025) complements these representations with typed service discovery, distributed invocation, and service-level monitoring hooks, in line with long-standing middleware traditions in robotics (Macenski et al. 2022).

The central claim of this paper is that the new genAI protocols and the established cognitive robotics infrastructure are not competitors but natural complements, and that they can be connected through two concrete bridges. The first, which we call the *MCP bridge*, exposes DIARC actions as semantically aligned MCP tools, giving genAI agents access to grounded, precondition-checked, monitored robot capabilities. The second, the *A2A bridge*, realizes A2A-style capability exchange and task coordination on top of TRADE's reflective service substrate, so that inter-agent communication itself becomes observable and governable by the cognitive architecture. The resulting integration serves a dual purpose: genAI agents gain access to robust cognitive and robotic competencies, while cognitive architectures gain systematic access to generative AI modules, all under explicit monitoring and policy control. We argue that this combination of

protocol interoperability, semantic transparency, and reflective supervision is a key ingredient for safer multi-agent systems in mixed-initiative and embodied domains.

The remainder of the paper is organized as follows. We first review tool-augmented language model systems, multi-agent coordination protocols, and cognitive architectures with reflective middleware. We then describe DIARC's action representation and TRADE's service model in detail, before presenting the MCP bridge, including a working implementation and the referential grounding problem it exposes. We subsequently develop the A2A bridge over TRADE, illustrate both bridges with extended multi-step scenarios, propose evaluation dimensions, and discuss the implications for safety, introspection, and governance of multi-agent systems.

2 Background and Related Work

2.1 Tool-augmented LLM systems

Modern LLM deployments increasingly rely on explicit tool interfaces for operations that exceed purely textual reasoning. Early approaches demonstrated that models can be prompted to interleave reasoning and acting (Yao et al. 2023) or can even teach themselves when and how to call external APIs (Schick et al. 2023). Commercial systems subsequently formalized "function calling" interfaces in which models emit structured invocation requests that are validated against declared schemas (OpenAI 2023). Across these approaches, the common lesson is that structured tool invocation improves reliability and observability compared to unconstrained free-text actions, particularly when inputs are validated by schemas and outputs are normalized before being fed back to the model.

MCP consolidates this trend into a language-agnostic protocol for tool discovery and invocation (Anthropic 2024). An MCP server advertises a set of tools, each with a name, a natural-language description, and a JSON schema for its inputs; clients—typically LLM-hosting applications—discover these tools at runtime and route model-generated invocation requests to the appropriate server. MCP's strength lies in its broad interoperability and transport flexibility. Its limitation in robotics contexts, however, is that tool schemas alone do not capture full symbolic action semantics as there is no standard place to express formal preconditions and effects, resource conflicts, expected durations, or action-level policy contracts. A tool description may *state* in prose that an object "must be visible and reachable", but nothing in the protocol enforces this condition or allows a planner to reason over it.

2.2 Multi-agent coordination protocols

Where MCP addresses the model-tool boundary, A2A-style designs address the agent-agent boundary. A2A provides mechanisms for capability advertisement (via "agent cards"), task negotiation, long-running exchanges with intermediate status updates, and structured final responses among autonomous agents (Google 2025). The two protocol families are complementary: Tools provide capabilities, while A2A provides coordination patterns among the entities that use those capabilities. Yet, most current agent systems treat both tools and peer agents as opaque functions, black boxes that accept structured input and return structured output. This opacity forecloses two opportunities that cognitive architectures have long exploited: Robust planning-time reasoning over what a capability actually does, and execution-time safety mediation over whether a particular invocation should be allowed to proceed.

2.3 Cognitive architectures and reflective middleware

Cognitive architectures in robotics emphasize explicit task semantics, deliberative control, and explainable execution traces, a tradition that reaches back to general cognitive architectures such as Soar (Laird 2012) and ACT-R (Anderson 2007). DIARC (Scheutz et al. 2019) instantiates this tradition for embodied human-robot interaction, provideing symbolic action representations integrated with planning, natural-language understanding, perception, and affect processing. It has been deployed on a wide range of robotic platforms (Scheutz and others 2025) and is implemented in the TRADE middleware (Scheutz 2025) which contributes the distributed systems substrate, i.e., a reflective service middleware supporting typed discovery and invocation across processes, machines, and languages, comparable in spirit to, but architecturally distinct from, robotics middleware such as ROS 2 (Macenski et al. 2022).

The key observation motivating this paper is that these two research lineages solve disjoint parts of the same problem. MCP and A2A provide broad ecosystem interoperability but shallow semantics; DIARC and TRADE provide deep semantics and runtime introspection but have historically lived inside closed architectural boundaries. Bridging them yields systems that are simultaneously *open,* i.e., able to interoperate with the rapidly growing agentic ecosystem, and *governable*, i.e., subject to the checking, monitoring, and supervision that cognitive architectures make possible. The safety-engineering literature has long argued that such systemic control structures, rather than component reliability alone, are what make complex socio-technical systems safe (Leveson 2011). The bridges we

propose can be viewed as instantiating precisely such control structures for multi-agent genAI systems.

3 DIARC Actions and TRADE Middleware

3.1 Semantically rich action representations

Actions in DIARC represent both physical robot behaviors (grasping, navigating, handing over) and internal information-processing operations (memorizing a fact, adopting a goal, resolving a referent). Every action carries a name and type that identify it within the architecture's action database, together with a set of typed *roles* such as`?actor:agent` or `?object:physobj` that constrain what entities may fill the action's argument positions. Crucially, each action also specifies logical *preconditions* that must hold for the action to be applicable, and *postconditions* that describe its expected outcomes, separately for success, failure, and unconditional effects. An action's *event specification* binds the symbolic description to an executable implementation, typically a TRADE service call, and additional *metadata* annotate the action with cost, expected duration, timeout, benefit, and policy information that planners and executives can exploit.

This representational richness is what distinguishes a DIARC action from a simple remote procedure call. Because preconditions and effects are explicit and symbolic, the architecture can plan over actions, verify their applicability at execution time, detect and diagnose failures by comparing expected against observed effects, and generate natural-language explanations of what it did and why (Scheutz et al. 2019). None of these capabilities are available when a capability is exposed only as an opaque function signature.

3.2 TRADE: a reflective distributed service substrate

TRADE provides the distributed execution substrate underneath DIARC (Scheutz 2025). Components register their capabilities as typed *services*, and consumers locate services through constraint-based discovery over service names, argument and return types, and group memberships—so that, for example, a component can request "any service named `moveTo`taking a `Pose` and provided by a component in the group `robot1`." Invocation is type-safe and location-transparent: TRADE routes calls across process and machine boundaries via sockets and Java

reflection, so callers need not know where a service is physically hosted. Wrappers and language bridges further allow TRADE to interoperate with external middleware ecosystems such as ROS (Macenski et al. 2022).

Two properties of TRADE are central to this paper's argument. First, TRADE is *reflective*, i.e., the set of currently registered services, their types, and their group structure are themselves queryable at runtime, so that the architecture can reason about which components are active and which capabilities are currently available and can notice when that set changes. Second, TRADE supports *monitoring* via before- and after-call hooks that can be attached to any service. These hooks turn every service interaction into an observable, and potentially interceptable, event. Together, reflection and monitoring enable architecture-level introspection that is directly useful for fault isolation, dynamic reconfiguration, and, as we develop below, safety policy enforcement over agentic interactions (Leveson 2011).

4 Exposing DIARC Actions as MCP Tools

4.1 The structural mapping

The correspondence between DIARC actions and MCP tools (Anthropic 2024) is direct and, we argue, semantically well-founded. An action's name becomes the tool name; its typed roles induce the tool's JSON input schema; its preconditions and effects are rendered into the tool description as guidance and constraints for the calling model; and its event specification becomes the MCP handler implementation that submits the corresponding goal to DIARC's goal manager. Consider the DIARC`pick-up`action:

```
(:action pick-up
  :roles (?actor:agent ?object:physobj)
  :precondition (and (visible ?object)
                     (reachable ?actor ?object))
  :effect (holding ?actor ?object))
```

An MCP-compatible tool representation renders the same capability as:

```
{
  "name": "pick_up",
  "description": "Pick up an object if visible and reachable.",
  "input_schema": {
    "type": "object",
    "properties": {
      "object": {"type": "string"}
    },
    "required": ["object"]
```

```
  }
}
```

and the handler translates an invocation into a DIARC goal:

```
public ActionResult pickUp(String object) {
  return diarcGoalManager.submitGoal(
    "pick-up ?actor robot ?object " + object);
}
```

The critical point is what happens *after* the handler fires. Because the invocation is realized as a DIARC goal rather than a bare function call, the full machinery of the architecture applies: The action's preconditions are checked against the current world model before execution begins, execution is monitored against the declared effects, failures are classified and reported with symbolic reasons, and the goal can be interrupted or superseded by higher-priority goals. The genAI agent thus receives not merely a return value but a *goal status* grounded in the architecture's execution semantics. Figure 1 shows the actual tool definition in our implementation, which uses the Spring AI MCP framework. Note how the handler submits the goal via TRADE service calls (`submitGoal`,`joinOnGoal`) and returns the resulting `GoalStatus`, so that success and failure are communicated in architecture-level terms rather than as opaque exceptions.

4.2 Grounding descriptive language to executable symbols

Implementing this bridge immediately surfaces a challenge that is easily overlooked in disembodied tool use: *referential grounding*. DIARC's `pickUp`expects a unique object identifier such as `physobj_3`, because the architecture's perception and manipulation pipelines operate over grounded object tokens. A user prompt like “pick up the bottle,” however, provides only an open-vocabulary description. No amount of prompt engineering can conjure the correct identifier, because the mapping from descriptions to identifiers depends on the robot's current perceptual state.

Our solution is to expose a companion *grounding tool*, `findObject` (Figure 2), that maps open-vocabulary descriptions to grounded unique identifiers through DIARC's perception and reference-formation machinery. Internally, the tool posits a reference for the description, submits a `findObject` goal that drives visual search, and returns the unique identifier only if the goal succeeds. The LLM can then chain the two tools: `findObject("bottle")`→`pickUp(uniqueID)`. In our experiments, models discover and execute this chaining pattern readily, guided only by the tool descriptions, which state that `pickUp` requires a unique ID and that `findObject` returns one. This two-step pattern generalizes across embodied tasks (finding locations before navigating, identifying persons before addressing them)

and motivates a general design principle for embodied MCP servers: *toolsets should expose grounding tools alongside terminal actuation tools*, mirroring the separation between reference resolution and action execution inside the cognitive architecture itself.

4.3 Implementation

MCP servers and tool calls are implemented in DIARC using the Spring AI MCP framework, with tool handlers dispatching into the architecture via TRADE service calls as shown in Figure 1 and Figure 2. For experimentation, we ran a self-hosted `qwen3:8b` model through Ollama and connected it to the DIARC MCP server via the open-source Ollmcp bridge using streamable HTTP transport. This entirely self-hosted configuration demonstrates that the bridge does not depend on any proprietary model or hosted service, which matters for robotics deployments with connectivity, latency, or data-governance constraints. Qualitatively, even a modest 8B-parameter model was able to resolve user requests into correct `findObject`/`pickUp` tool chains, with the architecture's precondition checking catching the cases where the model's choices were unexecutable.

```
@McpTool(name = "pickUp", description = "Pick up the object specified
by the unique ID. The input to this call must be the object's unique
ID. The object must be visible and reachable. ")
public Map<String,Object> pickUp(
      @McpToolParam(description = "Unique ID of the object to pick up")
String uniqueID
) {
       Predicate    goal    =    new    Predicate("pickUp",    actor,
Factory.createSymbol(uniqueID));
  GoalStatus goalStatus = GoalStatus.UNKNOWN;
  try {
    long goalId = submitGoal.call(Long.class, goal);
    goalStatus = joinOnGoal.call(GoalStatus.class, goalId);
  } catch (TRADEException e) {
    log.error("Error submitting goal", e);
  }
  return Map.of("status", goalStatus);
}
```

PickUp MCP tool definition in DIARC using the Spring AI framework.

```
@McpTool(name = "findObject", description = "Find an object matching
the provided description in the current FOV. Returns the unique ID of
the visible object.")
public Map<String,Object> findObject(
      @McpToolParam(description = "Description of object to look for")
String description
) {
  Symbol uniqueId = positReference(description);
  Predicate goal = new Predicate("findObject", actor, uniqueId);

  GoalStatus goalStatus = GoalStatus.UNKNOWN;
```

```
  try {
    long goalId = submitGoal.call(Long.class, goal);
    goalStatus = joinOnGoal.call(GoalStatus.class, goalId);
  } catch (TRADEException e) {
    log.error("Error submitting goal", e);
  }

  String uniqueIdStr = "";
  if (goalStatus == GoalStatus.SUCCEEDED) {
    uniqueIdStr = uniqueId.getName();
  }

  return Map.of("status", goalStatus, "uniqueID", uniqueIdStr);
}
```

FindObject MCP tool definition in DIARC using the Spring AI framework.

5 A2A-Style Communication over TRADE

While the MCP bridge connects genAI models to individual capabilities, the A2A bridge addresses coordination among agents. A2A is designed for interoperable exchanges among autonomous agents: capability discovery, task negotiation, and structured long-running responses (Google 2025). Our observation is that TRADE already provides the primitives from which these patterns can be built (Scheutz 2025). Inter-agent tasks map naturally onto typed TRADE services: an agent that can perform deliveries simply registers a `deliver` service with appropriately typed arguments. Capability advertisements, which A2A realizes as agent cards, map onto TRADE's `TRADEServiceInformation` structures, which describe available services together with their type signatures and group memberships and are queryable at runtime through TRADE's reflective discovery machinery. Negotiation and state-sharing patterns, i.e., proposing a task, accepting or declining it, reporting intermediate progress, can be realized as dedicated meta-services layered over the same substrate, with long-running tasks naturally represented as DIARC goals whose status can be polled or joined on.

This mapping yields a bridge with benefits in both directions. GenAI agents speaking A2A can interact with TRADE-native cognitive agents as peers, discovering and invoking their capabilities without bespoke integration code. Conversely, DIARC/TRADE systems, which historically communicated only over their internal socket-based protocols, can participate in web-facing agent ecosystems.

The most significant benefit, however, is *unification of oversight*. Because A2A-style exchanges are realized as TRADE service calls, the same before/after monitoring hooks that observe classical cognitive-service interactions also observe genAI-mediated interactions. There is no separate, unmonitored channel through which external agents influence the system. A policy component can therefore implement cross-cutting safety and accountability logic such as logging every inter-

agent request, checking proposed tasks against normative constraints, vetoing or modifying calls before they reach actuation, etc., and in uniform way across both worlds. In safety-engineering terms, the middleware becomes the locus of the control structure that constrains the behavior of the overall system (Leveson 2011): the cognitive architecture does not merely *participate* in the multi-agent system, it can *supervise* it.

6 Illustrative Multi-Step Scenarios

We now illustrate how the two bridges operate together in mixed-initiative embodied settings. Both scenarios are grounded in the implemented tool definitions of Figures~1 and~2.

Scenario 1: Assistive manipulation with recovery

Consider a user who asks a robot, “Please hand me the blue bottle.” The request first reaches the genAI agent, which decomposes it into a grounding step and a manipulation step. The agent calls `findObject("blue bottle")`; inside DIARC, this posits a reference for the description and drives a visual search over the current field of view, returning a grounded identifier such as `physobj_7` together with a success status. The agent then calls`pickUp("physobj_7")`. Before any motor command is issued, DIARC validates the action's preconditions that the object must be visible and reachable by the acting manipulator. Suppose the bottle has in the meantime become occluded by another object. The precondition check fails, and unlike a bare API that would return an uninformative error the architecture can respond in two ways: It can report the symbolic failure reason (`not(visible(physobj_7))`) back through the MCP result, allowing the LLM to re-invoke `findObject`or ask the user for help; or the DIARC planner can autonomously insert recovery actions such as reobserving the scene, repositioning the arm or base, and retrying the grasp, with the genAI agent receiving only the final consolidated goal status.

The key point is that the tool chain is constrained by explicit action semantics and monitored execution at every step, rather than by ad-hoc prompt logic (Scheutz et al. 2019; Yao et al. 2023). The LLM proposes *what* to do; the architecture decides *whether* it can be done, verifies *that* it can be done and *that* it was done, and handles *when* it goes wrong. Failure handling is thereby moved from the least reliable component of the system (the language model) to the most reliable one (the symbolic execution engine).

Scenario 2: Multi-agent delegation with safety interlock

The second scenario exercises the A2A bridge. Suppose a genAI task-planning agent, coordinating a warehouse workflow, needs a package moved from a loading dock to a laboratory. Through A2A-style discovery over TRADE, it locates a mobile robot agent advertising a `deliver` capability and delegates the task. In a conventional agent framework, the delegation message would travel directly from planner to robot, and any safety checking would depend on both agents' internal diligence. In our architecture, the delegation is a TRADE service call, and a third component, a policy-monitoring agent, has registered before-call hooks on all actuation-relevant services.

When the delivery request arrives, the monitor inspects the proposed route against a normative constraint: the corridor past the biosafety area is a restricted zone during operating hours. Because the robot's planned trajectory would traverse that zone, the monitor vetoes execution *before actuation begins* and returns a structured rejection identifying the violated constraint. The planning agent, receiving this symbolic feedback, requests replanning with the restricted zone as an added constraint; the robot proposes an alternate route; the monitor approves; and execution proceeds under continued after-call monitoring, with intermediate progress reported back to the delegating agent as long-running task updates.

Three properties of this interaction deserve emphasis. First, the safety interlock required no cooperation from the genAI agents since it is enforced at the middleware layer that all interactions must traverse. Second, the veto is *informative*: The symbolic constraint violation supports replanning rather than mere refusal. Third, the entire episode, including the vetoed attempt, is captured in TRADE's monitoring stream, yielding an audit trail of who requested what, what was blocked and why, and what ultimately executed. This is precisely the kind of controllable, accountable mixed-initiative behavior under uncertainty that safety engineering demands of autonomous systems (Leveson 2011).

7 Evaluation Dimensions

Evaluating an integration of this kind requires metrics that go beyond raw task performance to capture controllability and robustness. We propose six dimensions. *End-to-end task success rate* measures the fraction of natural-language tasks completed successfully through the full pipeline, from utterance to embodied outcome. *Grounding success* isolates the referential component by tracking the rate at which descriptive references are resolved to executable unique identifiers, which our experience suggests is a dominant failure mode in embodied tool use. *Tool-chain efficiency* captures the average number of tool calls per successful task and

the distribution of failure stages, distinguishing model errors (wrong tool, wrong arguments) from architectural failures (precondition violations, execution faults). *Latency decomposition* apportions end-to-end time among LLM inference, middleware invocation, and embodied execution, identifying where responsiveness is lost. *Safety interventions* quantify the governance layer, e.g., the frequency of policy vetoes, the proportion of genuinely unsafe requests blocked, and the rate at which vetoed tasks are successfully replanned. Finally, *recoverability* measures task success under injected faults such as missing capabilities, stale capability advertisements, and intermittent service failures, probing the value of TRADE's reflective discovery and monitoring.

Together, these dimensions quantify not only what the integrated system can do, but how well it can be trusted to do it, a key property required for real-world deployment.

8 Discussion: Safety, Introspection, and Governance

The proposed bridges induce a layered assurance model. At the lowest layer, *protocol structure* (MCP/A2A) improves interoperability and eliminates the ambiguity of free-form interfaces as every capability invocation is typed, validated, and attributable (Anthropic 2024; Google 2025). At the middle layer, *action semantics* (DIARC) make the assumptions behind each capability explicit through typed roles, preconditions, and effects, so that applicability can be checked before execution and outcomes verified after it (Scheutz et al. 2019). At the top layer, *reflective middleware* (TRADE) provides the runtime control structure, enabling observation of all service interactions, policy mediation with veto power, and complete auditability (Scheutz 2025).

The layers are mutually reinforcing. Consider the life of a single genAI-originated request in the integrated system. The request arrives as a structured MCP invocation in layer one, is transformed into a DIARC action whose preconditions gate execution in layer two, and is realized as TRADE service calls that policy hooks can inspect, log, veto, or modify before and after execution in layer three. At every stage there is a well-defined interception point with access to symbolic information about what is being attempted. Such control points are extremely difficult to retrofit when tool use is handled only inside model prompts, where the "policy" is a natural-language instruction the model may or may not honor and the "audit trail" is only a chat transcript (OpenAI 2023; Yao et al. 2023).

This is, we submit, the deeper significance of the integration for the multi-agent systems community: Cognitive architectures should not be viewed merely as another kind of agent to be wrapped in an agentic protocol. Rather, their semantic representations and reflective infrastructure position them to serve as *supervisory substrates* for agentic ecosystems: checking interactions among genAI agents,

mediating their access to the physical world, and rendering the collective behavior of the system introspectable and accountable (Leveson 2011).

9 Limitations and Future Work

Several challenges remain. The most fundamental is *ontology alignment*: mapping open-vocabulary language onto closed typed role systems can still be brittle; and while grounding tools such as `findObject` mitigate the problem for perceptual reference, more abstract mismatches between a model's conceptualization of a task and the architecture's action vocabulary require further work. Second, the protocols themselves are moving targets: MCP and A2A ecosystem details continue to evolve (Anthropic 2024; Google 2025), so the bridges must be designed for adaptability rather than fixed to a single protocol revision. Third, *trust and provenance* become pressing once external agents can advertise capabilities into the system: Capability advertisements from third parties require validation mechanisms well beyond type checking. Relatedly, *security hardening* of exposed services demands authentication, authorization, and misuse-resistant policy controls, since an MCP endpoint on a robot is an attack surface with physical consequences. Finally, *reproducibility* remains difficult as non-deterministic LLM behavior produces variable tool-selection trajectories, complicating both debugging and systematic evaluation.

These limitations chart the future agenda. We plan to develop machine-readable exports of DIARC's precondition/effect models into richer tool metadata, so that calling agents (and their planners) can reason over action semantics rather than prose descriptions; formal policy languages for expressing safety contracts at middleware boundaries, replacing hand-coded monitoring hooks with declarative, verifiable specifications; and benchmark suites for embodied MCP/A2A interoperability that operationalize the evaluation dimensions proposed above.

10 Conclusion

We presented a practical integration path between cognitive robotics and modern genAI agent protocols. By exposing DIARC actions as MCP tools together with the grounding tools that embodied reference demands and by realizing A2A-style capability exchange over the reflective TRADE middleware, we obtain systems that are interoperable, semantically grounded, and operationally introspectable. The bridge is bidirectional: genAI agents gain access to robust cognitive and robotic competencies with precondition checking, monitored execution, and principled

failure handling, while cognitive architectures gain systematic access to generative AI modules under explicit monitoring and policy control. Most importantly, because all interactions traverse a reflective middleware substrate, the cognitive architecture can check, mediate, and supervise the interactions among agentic components, providing the layered control structure that safe multi-agent systems in mixed-initiative and embodied settings will require.

11 References

Anderson, John R.. 2007. How Can the Human Mind Occur in the Physical Universe?. Oxford University Press.

Anthropic. 2024. Model Context Protocol Specification. https://modelcontextprotocol.io/specification/2024-11-05/. Last accessed: 2026-07-26.

Google. 2025. Agent2Agent (A2A) Protocol. https://google.github.io/A2A/. Last accessed: 2026-07-26.

Laird, John E.. 2012. The Soar Cognitive Architecture. MIT Press.

Leveson, Nancy. 2011. Engineering a Safer World: Systems Thinking Applied to Safety. MIT Press.

Macenski, Steven, Foote, Tully, Gerkey, Brian, Lalancette, Christian, Woodall, William. 2022. Robot Operating System 2: Design, architecture, and uses in the wild. Science Robotics 7(66).

OpenAI. 2023. Function calling and other API updates. https://openai.com/index/function-calling-and-other-api-updates/. Last accessed: 2026-07-26.

Scheutz, Matthias. 2025. TRADE: A Middleware Framework for Distributed Cognitive Robotic Architectures. In Proceedings of the AAAI Spring Symposium Series.

Scheutz, Matthias, others. 2025. DIARC: Distributed Integrated Affect Reflection Cognition Architecture. https://github.com/mscheutz/diarc. Last accessed: 2026-07-26.

Scheutz, Matthias, Williams, Tom, Krause, Evan, Oosterveld, Bradley, Sarathy, Vasanth, Frasca, Tyler. 2019. An Overview of the Distributed Integrated Affect Reflection Cognition Architecture. In Cognitive Architectures, edited by Aldinhas Ferreira, M., Silva Sequeira, J., Ventura, R., 165–193. Springer.

Schick, Timo, Dwivedi-Yu, Jane, Dessi, Roberto, Raileanu, Roberta, Lomeli, Maria, Hambro, Eric, Zettlemoyer, Luke, Cancedda, Nicola, Scialom, Thomas. 2023. Toolformer: Language Models Can Teach Themselves to Use Tools. In Advances in Neural Information Processing Systems (NeurIPS).

Yao, Shunyu, Zhao, Jeffrey, Yu, Dian, Du, Nan, Shafran, Izhak, Narasimhan, Karthik, Cao, Yuan. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. In The Eleventh International Conference on Learning Representations (ICLR).