Probabilistic Programming with Programmable Context Control

McCoy Reynolds Becker
MIT
Cambridge, Massachusetts, USA
mccoyb@mit.edu

Vikash K. Mansinghka
MIT
Cambridge, Massachusetts, USA
vkm@mit.edu

Abstract

Programming with autonomous agents based on large language model is an emerging discipline: tools like Claude Code and Codex have popularized the usage of agents to write and execute code for tasks of increasing complexity.

Existing tools adopt the UI model of chat assistants, and typically force users into *single agent chat loops* – a single stream of interaction. Expert users move beyond single agent interactions, orchestrating multiple assistants simultaneously in many streams of interaction in what we refer to as a *multi-agent conversation*.

Under the probabilistic viewpoint, each interaction stream within a larger conversation corresponds to a sequence of samples from next-token distributions. Coordinating these streams (including passing information *between* streams) to perform useful work demands *context control* of the conversation: facilities for manipulation, including *reactive control*, what information each agent consumes to condition their next-token distribution on, and *time travel*, the ability to move forward and backward in the states of multi-agent conversations, which supports forking of conversations in new directions, or trying them multiple times.

We introduce Duet, a new language and system which addresses the requirement of context control: Duet extends the *syndicated actor model*, a programming model for entities which communicate across a shared communication channel, with *reversibility*, the ability to operationally

Keywords: contextual control, conversational concurrency, multi-agent systems, reversible runtimes, AI tooling

1 Introduction

Contemporary "agentic" IDE experiments increasingly chain multiple assistants into collaborative workflows. A developer might task one agent with exploring design patterns or API trade-offs, capture its rationale as a structured document, and hand that artefact to a second agent that writes and tests code. Each agent inhabits its own conversational state, conditioned on past turns and delimited by the tools it may invoke. Orchestrators must therefore keep several probabilistic contexts aligned: every message is a conditional query on the agent's amortised distribution, and letting contexts bleed together risks lost focus, capability misuse, or inconsistent provenance.

We argue that managing these multi-agent distributions requires language support for contextual control. Practitioners need constructs that partition conversations into scoped facets, govern authority over external tools, and make the resulting inference trail reversible. Ad-hoc scripts and chat interfaces do not provide these guarantees: they interleave unrelated contexts, lack precise capability boundaries, and rarely produce audit-ready logs.

Duet addresses this gap by combining a reversible Rust runtime with a Python DSL tailored to contextual control. Facets carve out independent conversational regions whose assertions persist across turns. Capability attenuation constrains which tools an agent may invoke inside its facet. Every turn is logged so that users can replay or rewind entire agent pipelines, enabling systematic experimentation across multiple distributions.

This paper contributes:

- A conceptual case that conversational concurrency—the syndicated actor model of Garnock-Jones—provides the right linguistic substrate for contextual control of multi-agent distributions, reifying each agent stream as a facet with explicit assertions and credit.
- A reversible execution design that turns those facets into navigable artefacts: users can rewind and branch conditional query streams thanks to stateless entities, turn logging, and capability-aware tooling.
- A language of contextual policies, expressed through structural reactions and capability caveats, which governs how evidence flows between facets in practice; our early experience highlights open questions in probabilistic semantics and governance.

2 Motivating Scenario

Consider a development assistant that must explore a repository, suggest edits, run `ripgrep` queries, and coordinate with human operators. The assistant should only run approved commands, must leave an auditable trail, and should allow the operator to rewind if a change looks suspicious. Traditional shell scripts or chat-based bots treat each command independently and cannot maintain shared assertions about the codebase or the conversation state.

Duet models this scenario as a set of facets hosted by a Rust actor. Each facet manages capability-scoped tool access via the Codebase broker, emits assertions describing work

in progress, and reacts to structured facts describing user requests or tool results. When the operator rewinds a turn, the runtime automatically retracts associated assertions and requeues the triggering event, giving the assistant a chance to explore an alternative plan without side effects leaking through the system. This structure lets practitioners iterate on LLM-driven workflows while keeping their repositories and tool invocations under explicit control.

3 Background and Goals

Duet builds on conversational concurrency [1], which extends actors with assertions and facets to express multi-party dialogues. Runtimes such as `syndicate-rs` inspired the project, but their OCaml and Rust prototypes lacked ergonomic Python bindings, persistence suited to tooling, and a modern capability story. The design goals for Duet were:

1. **Deterministic reversibility**: every processed turn should be replayable or rewindable without undefined behaviour.
2. **Stateless entities**: durable facts live in assertions so that persisting turn logs suffices to reconstruct state.
3. **Scoped authority**: capabilities can be attenuated, combined, and revoked to match the principle of least privilege.
4. **Scripting ergonomics**: orchestrators should write high-level logic in Python while relying on Rust for determinism and performance.

4 Conversational Concurrency for Contextual Control

Conversational concurrency gives us the vocabulary to describe multi-agent workflows. Following Garnock-Jones, an actor owns a tree of facets; each facet encapsulates the assertions, observers, and credit that define an agent's role. A facet is the handle for a conditional query stream: spawning a facet creates a new conversational context, while retracting its assertions retires that slice of the distribution. This framing lets orchestrators name roles (research, implementation, review) and reason about them independently even when they interleave during execution.

5 Reversible Orchestration Policies

We extend this substrate with two ideas. First, reversibility: Duet records every turn (event cause, assertion delta, entity lifecycle, facet creation) and keeps entities stateless, so we can replay or rewind any agent stream without losing contextual separation. Second, contextual policies: structural reactions compile into observe patterns that filter which evidence reaches a facet, while capability caveats attenuate access to external tools. Presets for Git, Jujutsu, and `ripgrep` exemplify how tool authority follows the facet boundary. Together, reversibility and policies form a language for steering how evidence flows between agents while preserving determinism.

6 Experience and Outlook

A pilot workflow tasks one facet with surveying implementation strategies and another with synthesizing code from that survey. Reversibility let users branch on alternative research artefacts, replay the coding facet, and compare outcomes without contaminating parallel roles. Similar benefits appeared in code-review scenarios, where policy-scoped facets surfaced capability misuse immediately. These experiences raise questions we hope to explore with the LAFI community: How should turn logs expose probabilistic annotations? Can we statically analyse capability policies for safety? And how might conversational concurrency interact with differentiable inference engines? Additional examples appear in the appendix.

7 Conclusion

Duet aligns with LAFI's agenda by demonstrating how language abstractions can structure AI-facing orchestration. A reversible turn-based core, stateless entities, and capability-aware tooling give practitioners a principled substrate for building assistants that remain auditable and correctable. Ongoing work will extend Duet's persistence, pattern compilation, and policy support, bringing conversational concurrency into mainstream AI toolchains.

References

[1] Tony Garnock-Jones. 2013. *Effigy: Language Support for Communicating Managers.* Ph.D. Dissertation. Northeastern University.