AgentDSL: Compile-Time Validated Orchestration for LLM Agent Systems

Simon Massey*

August 28, 2025

Abstract

We propose AgentDSL, a novel approach to LLM agent orchestration that leverages compile-time type validation to ensure structural correctness of agent workflows. By utilizing Java 25's reactive stream abstractions and dynamic compilation capabilities, AgentDSL enables LLMs to generate executable workflow specifications that are validated before execution. This approach addresses fundamental reliability challenges in current agent systems while maintaining the flexibility required for dynamic task decomposition.

1 Introduction

Current LLM agent frameworks rely heavily on runtime validation and prompt engineering to coordinate complex workflows. This approach leads to several challenges: late discovery of structural errors, context pollution from unnecessary tool loading, and difficulty in expressing complex execution patterns. AgentDSL addresses these limitations by introducing a compile-time validated domain-specific language for agent orchestration.

2 Motivation

2.1 Limitations of Current Approaches

Contemporary agent systems typically implement orchestration through:

- Template-based prompt construction with runtime tool discovery
- Linear conversation histories that inadequately represent parallel execution
- Global tool contexts that introduce unnecessary complexity

*Corresponding author: simbo1905@github.com

- Runtime validation that discovers structural errors only during execution

2.2 The Promise of Type-Safe Orchestration

Type systems have long provided guarantees about program structure. By applying these principles to agent orchestration, we can:

- Detect invalid tool invocations before execution
- Express complex workflow patterns explicitly
- Minimize context size through selective tool loading
- Enable LLMs to reason about workflow structure programmatically

3 Design Principles

3.1 Compile-Time Validation

AgentDSL leverages the Java compiler to validate workflow structure. If generated code compiles successfully, the workflow is guaranteed to reference only valid tools with correct parameter types.

3.2 Reactive Stream Abstraction

Using Project Loom-compatible reactive streams (Eclipse Mutiny, Spring WebFlux), AgentDSL represents workflows as composable asynchronous operations with built-in retry, timeout, and error handling semantics.

3.3 Dynamic Compilation

Java 25's enhanced compilation APIs enable runtime generation and validation of workflow specifications without filesystem operations, supporting ephemeral, task-specific orchestrations.

3.4 Immutable Event DAG

Rather than linear conversation histories, AgentDSL models interactions as directed acyclic graphs of immutable events, enabling parallel execution branches and selective history pruning.

4 Architecture

4.1 Core Components

Listing 1: Core Type Definitions

```
sealed interface AgentOp<T> permits
    Execute<T>, Fork<T>, Validate<T>, Checkpoint<T> {}

interface ToolRegistry {
    @CompileTime
    Uni<T> invokeTool(String name, JsonNode params);
}

class WorkflowCompiler {
    Class<?> compile(String sourceCode)
        throws CompilationException;
}
```

4.2 Workflow Generation Process

1. LLM generates Java source code representing workflow
2. WorkflowCompiler validates and compiles specification
3. Successful compilation guarantees structural validity
4. Runtime executes validated workflow with reactive stream semantics

5 Implementation Strategy

5.1 Phase 1: Core Infrastructure

Development of compilation framework, type-safe tool registry, and basic reactive stream operators.

5.2 Phase 2: LLM Integration

Training data generation for workflow specification, prompt engineering for reliable code generation, and error message interpretation.

5.3 Phase 3: Runtime Optimization

Context caching strategies, parallel execution optimization, and checkpoint/recovery mechanisms.

6 Evaluation Metrics

6.1 Reliability

- Reduction in runtime failures due to structural errors

- Time to error discovery (compile-time vs runtime)
- Recovery success rate from transient failures

6.2 Efficiency

- Token usage reduction through selective context loading
- Parallel execution speedup for independent tasks
- Context cache hit rates

6.3 Expressiveness

- Complexity of workflows expressible in DSL
- LLM success rate in generating valid specifications
- Developer productivity metrics

7 Related Work

While several projects explore agent orchestration (LangGraph, AutoGen, CrewAI), none leverage compile-time validation for structural correctness. The closest analogues exist in distributed systems (Apache Beam, Flink) and reactive programming frameworks, which inspire our approach but target different problem domains.

8 Conclusion

AgentDSL represents a paradigm shift in LLM agent orchestration, moving from runtime discovery to compile-time validation. By enabling LLMs to generate type-safe workflow specifications, we achieve both reliability and flexibility in agent system design. Initial prototypes demonstrate feasibility, and we seek collaboration to develop this approach into production-ready tooling.

9 Acknowledgments

This work builds upon insights from the reactive programming community and benefits from discussions within the OpenHands and Aider.chat communities.