Architecture Notes

Overview

This document outlines the architecture of the Roo Code extension and the plan to transform it into an AI-Native IDE with Intent-Code Traceability.

Current Architecture

1. Tool Loop

The tool execution loop is primarily handled in `src/core/task/Task.ts` and `src/core/assistant-message/presentAssistantMessage.ts`.

- **Main Loop**: `Task.recursivelyMakeClineRequests` (src/core/task/Task.ts lines ~2511-3326) handles the conversational loop:
    1. Constructs prompt.
    2. Calls API.
    3. Parses response (chunks).
    4. Calls `presentAssistantMessage` to handle tool calls.
- **Tool Execution**: `presentAssistantMessage` (src/core/assistant-message/presentAssistantMessage.ts) parses the assistant's message, identifies tool calls, and executes them via a `switch` statement that delegates to specific tool classes (e.g., `writeToFileTool.handle`).

2. Prompt Builder

The system prompt is constructed in `src/core/prompts/system.ts`.

- **Function**: `generatePrompt` (lines 41-110) builds the prompt by concatenating various sections (role definition, capabilities, rules, system info, objective, custom instructions).
- **Sections**: Prompt sections are imported from `./sections`.

3. Tool Definitions

- **Interfaces**: Defined in `src/shared/tools.ts`.
- **Implementations**: Located in `src/core/tools/`.
- **Registration**: Tools seem to be individually imported and used in `presentAssistantMessage.ts`.

Proposed Architecture for AI-Native IDE

1. Hook Engine (Middleware)

We will introduce a `HookEngine` in `src/hooks/HookEngine.ts` to intercept tool executions.

- **Integration Point**: `presentAssistantMessage.ts`.

- **Pre-Hook**: Before `writeToFileTool.handle(...)`, call `HookEngine.preToolExecution("write_to_file", params, task)`.
- **Post-Hook**: After tool execution, call `HookEngine.postToolExecution("write_to_file", params, result, task)`.

2. Data Model (.orchestration/)

We will implement the following data models in `src/hooks/models/`:

- `ActiveIntent`: Represents entries in `.orchestration/active_intents.yaml`.
- `AgentTrace`: Represents entries in `.orchestration/agent_trace.jsonl`.
- `IntentMap`: Represents `.orchestration/intent_map.md`.

3. New Tool: `select_active_intent`

- **Definition**: Add to `src/shared/tools.ts`.
- **Implementation**: Create `src/core/tools/SelectActiveIntentTool.ts`.
- **Behavior**: Reads `active_intents.yaml`, loads context, and returns it to the agent.
- **Hook Integration**: The `HookEngine` will enforce that this tool is called first.

4. Intent-Driven Prompt

- **Modification**: Update `src/core/prompts/system.ts` (or a specific section) to enforce the "Reasoning Loop" protocol: "You are an Intent-Driven Architect... Your first action MUST be to call select_active_intent...".

5. Semantic Git Layer

- **Traceability**: `postToolExecution` for `write_to_file` will calculate content hashes and append to `agent_trace.jsonl`.
- **Concurrency**: `preToolExecution` for `write_to_file` will check for stale file hashes (optimistic locking).

Implementation Plan

1. **Phase 0**: Architecture exploration (Complete).
2. **Phase 1**: Handshake (Reasoning Loop) - Implement `select_active_intent` and prompt updates.
3. **Phase 2**: Hook Middleware - Implement `HookEngine` and integrate into `presentAssistantMessage.ts`.
4. **Phase 3**: AI-Native Git - Implement tracing and hashing.
5. **Phase 4**: Parallel Orchestration - Implement concurrency control.