

ACTIONGRAPH + AI

How a Process-Oriented DSL Enables
AI-Assisted Industrial Automation

Overview for Technical Decision-Makers

THE VISION

An AI assistant that can **understand** a robotics task, **generate** the control program, **simulate** it against a digital twin, and **iterate** until all tests pass — autonomously.



Not a future concept — ActionGraph is designed from the ground up to make this possible.

WHY ACTIONGRAPH IS UNIQUELY AI-FRIENDLY

DECLARATIVE, NOT IMPERATIVE

Describe *what* should happen, not *how*. LLMs naturally generate declarative specifications.

YAML — A KNOWN FORMAT

LLMs have extensive training data on YAML. No custom parser or syntax guessing needed.

CONSTRAINED LANGUAGE

Limited feature set = fewer ways to go wrong. LLMs produce more reliable output in constrained domains.

PROCESS-ORIENTED THINKING

Task decomposition into processes maps directly to how LLMs reason about workflows.

General-purpose languages have infinite ways to solve a problem. ActionGraph has *one right way* — making AI generation reliable.

THREE PILLARS OF AI INTEGRATION

1. DOCUMENTATION AS API

- Docs written for humans *and* machines
- Structured, prescriptive, unambiguous
- Single source of truth
- LLM agents read the same docs developers read

2. TOOL ACCESS VIA MCP

- Compiler, simulator, debugger as MCP servers
- Standardized wire protocol
- Vendor-neutral (Claude, OpenAI, etc.)
- Agent calls tools in agentic loops

3. VERIFIABLE OUTPUT

- Generated code runs in simulator
- Deterministic digital twin testing
- Automated safety assertions
- No "trust and deploy"

PILLAR 1: DOCUMENTATION AS API

AG documentation follows the **Diátaxis framework**, structured for both human scanning and LLM extraction:

Principle	How It Helps LLM Agents
Rules as bullet-point lists	Agents extract constraints reliably
Tables for specifications	Structured data for parameter lookup
Prescriptive language (MUST / MUST NOT)	Eliminates ambiguity in code generation
Complete, copy-paste-ready examples	Pattern matching → correct output
Self-contained sections	No cross-reference chains to follow

Key decision: No separate "LLM prompt engineering" artifacts. One docs set serves developers, AI agents, and onboarding — all at once.

HOW AN LLM AGENT WRITES AG CODE

The agent follows a deterministic protocol — not free-form generation:

1. 1 Read `docs/index.md` — understand the documentation map
2. 2 Read `task_decomposition.md` — learn the process-oriented programming model
3. 3 Read relevant reference docs — syntax, node types, hardware modules
4. 4 Look up node specs in YAML source — the single source of truth for built-in types
5. 5 Study existing applications — real working examples to pattern-match against

The agent reads first, writes second. No guessing AG syntax — it follows the spec exactly.

PILLAR 2: MCP — THE TOOL PROTOCOL

Model Context Protocol — the emerging standard for LLM ↔ tool integration:

WHAT IS MCP?

- Open protocol by Anthropic (2024)
- Adopted by OpenAI, Microsoft, Google
- Standardized "USB-C" for AI tool access
- Thousands of public MCP servers

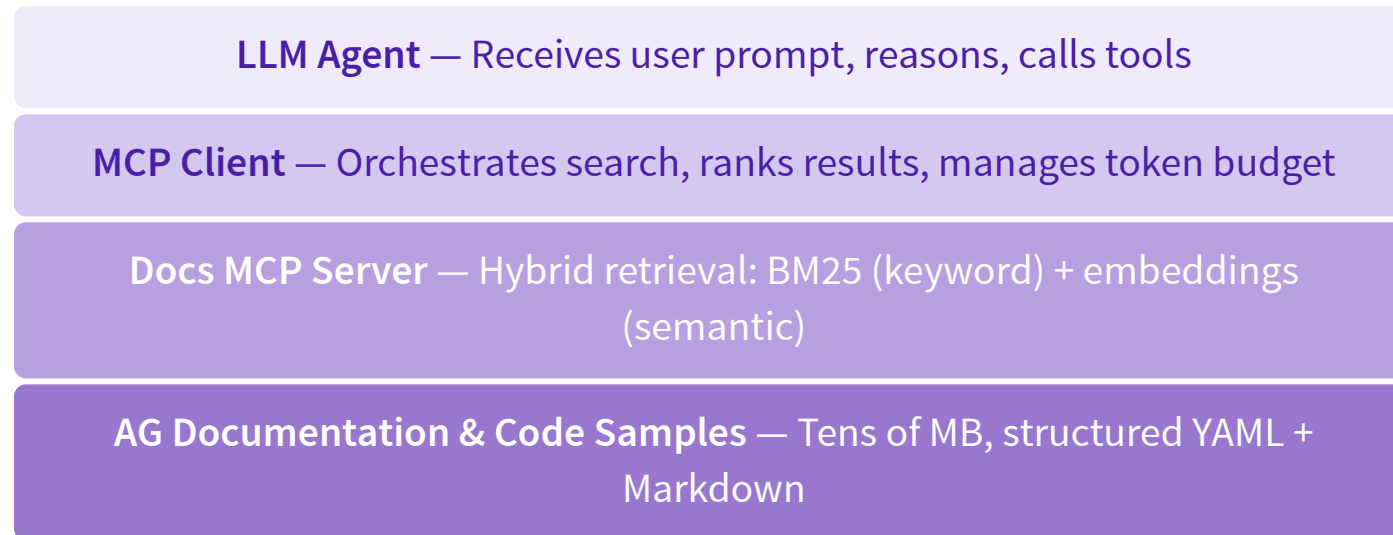
AG TOOLS AS MCP SERVERS

- **Compiler** Validate & compile YAML
- **Simulator** Run against digital twin
- **Debugger** Inspect runtime state
- **Docs** Search & retrieve docs (RAG)

Vendor neutral: Same MCP servers work with Claude, ChatGPT, Copilot, or any MCP-compatible client.
No lock-in.

KNOWLEDGE RETRIEVAL: RAG + MCP

The agent retrieves AG documentation via a hybrid RAG approach:



Exposed endpoints:

- `search(query)` → ranked hits
- `read(uri)` → full content
- `related(uri)` → linked docs

Why hybrid retrieval?

- BM25: exact matches on node names, params
- Embeddings: conceptual queries ("how to sync axes")
- Combined: best of both for DSL docs

PILLAR 3: VERIFIABLE OUTPUT

Every AI-generated program is verified before deployment:



COMPILE-TIME CHECKS

Syntax validation, parameter resolution, type checking — catches structural errors immediately.

SIMULATION TESTING

Python simulator replaces hardware. Deterministic execution with fixed seeds, reproducible results.

SAFETY ASSERTIONS

Automated checks for forbidden behaviors: no motion if safety low, timing budgets respected.

ITERATIVE FIX LOOP

Agent reads errors, adjusts code, re-compiles, re-tests. ReAct-style loop until all gates pass.

THE AI AGENT WORKFLOW

- 1 Understand** — Parse user prompt, identify required devices/APIs, infer implicit requirements
- 2 Decompose** — Break task into concurrent AG processes with clear responsibilities
- 3 Retrieve** — Search docs for relevant node types, patterns, and hardware modules
- 4 Generate** — Produce AG YAML + Python simulator + test suite
- 5 Compile** — Call compiler MCP server, fix any errors
- 6 Simulate** — Run against digital twin, measure KPIs
- 7 Iterate** — Fix failures, re-test until timing, safety, and coverage gates pass
- 8 Deliver** — Emit code, deployment docs, commissioning playbook

EXAMPLE: CONVEYOR PICK-AND-PLACE

USER PROMPT

"A camera detects defect parts on a conveyor.
The robot picks them off.
Test with various conveyor speeds.
Add e-stop button support."

WHAT THE AI AGENT DOES

1. Identifies: camera API, conveyor API, robot API
2. Infers: coordinate frames, pick windows, timing
3. Generates AG program with concurrent processes
4. Creates simulator with speed sweeps
5. Adds safety interlocks for e-stop
6. Runs tests, iterates to green

GENERATED AG STRUCTURE

```
Parallel:
- Type: ConveyorTracker
  # Monitors belt speed, part positions

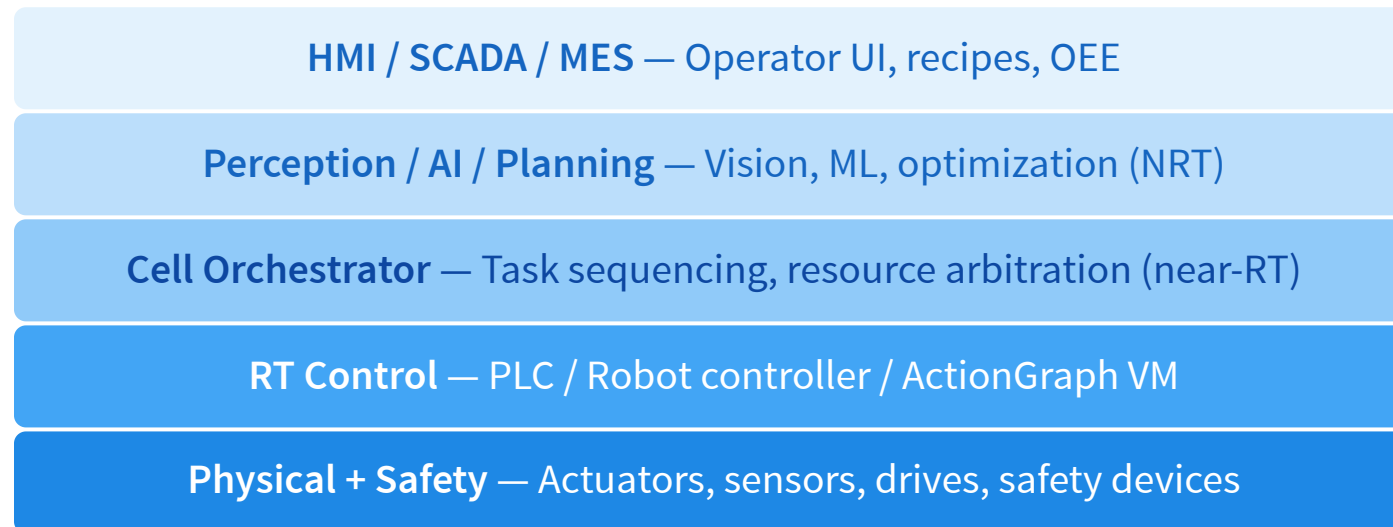
- Type: PickController
  # Coordinates picks based on tracking

- Type: SafetyMonitor
  # E-stop handling, safe retract

Sequential: # Per pick cycle
- MOVEMENT: {target: pick_pos}
- DIGITAL_SIGNAL_VALUE_SETTER:
  signal: gripper, value: 1
- MOVEMENT: {target: place_pos}
- DIGITAL_SIGNAL_VALUE_SETTER:
  signal: gripper, value: 0
```

BEYOND AG: THE GENERIC AI ASSISTANT BLUEPRINT

The same architecture generalizes to any industrial RT/NRT hybrid system:



The AI assistant operates through **contracts** between layers — not by generating monolithic code.

CONTRACT-BASED ARCHITECTURE

The assistant generates and enforces **machine-readable contracts** at every boundary:

INTERFACE CONTRACT

Schema, units, rate, QoS, error semantics per device/API. Agent auto-generates stubs and mocks.

TIMING CONTRACT

End-to-end latency/jitter budgets, per-hop deadlines, watchdog periods. Verified in simulation.

SAFETY CONTRACT

Safe states, interlocks, forbidden behaviors. Encoded as non-bypassable assertions in tests.

DEPLOYMENT CONTRACT

Where code runs, resource requirements, start/stop order, rollback procedures.

```
# Example: Timing Contract (machine-readable)
task: "actuate_on_event"
deadline_ms: 25
budget_ms:
  sensing: 2      # Camera → inference
  transport: 3    # Network hop
  nrt_processing: 8 # ML / planning
  rt_logic: 6     # AG VM execution
  actuation: 4    # Motor response
watchdog_ms: 40
```

AI-DRIVEN TEST GENERATION

The assistant doesn't just generate code — it generates comprehensive tests:

FUNCTIONAL

Does each process do what it should? Correct picks, correct sequence, correct outputs.

TIMING / PROPERTY

Are deadlines met? Gripper close within Δ ms of contact? Sweep speed & rate ranges.

STRESS

Burst arrivals, max part density, variable belt speeds, camera frame drops.

FAULT INJECTION

Camera timeout, conveyor stop mid-pick, network jitter, encoder failure.

SAFETY ASSERTIONS

No motion while e-stop active. No command into muted zone. Energy bounded.

COVERAGE GOALS

Pose distribution, state/transition coverage, speed range sweep.

The assistant **blocks deployment** unless timing, safety, and coverage gates all pass.

SIMULATION LADDER

Progressive verification from pure software to real hardware:

MIL — Model-in-the-Loop Fast kinematic model, broad coverage

SIL — Software-in-the-Loop AG VM + Python simulator, full logic test

PLC-SIL — PLC Sim-in-the-Loop Real PLC scan cycle, simulated I/O

HIL — Hardware-in-the-Loop Real robot controller I/O, physical actuators

AG's built-in simulator provides SIL out of the box. The same AG program runs unmodified at every level — only the HAL bindings change.

IMPLEMENTATION ARCHITECTURE

BROWSER UI

- Chat interface
- Code editor with AG syntax
- 3D visualization
- Test dashboard

AI ORCHESTRATOR

- LLM calls (Claude / GPT)
- ReAct-style tool loops
- Token budget management
- RAG retrieval

MCP TOOL SERVERS

- AG Compiler
- Simulator / Digital Twin
- Debugger & Logger
- Docs search (hybrid RAG)

Local web service. LLMs via paid API. Tools run on-premise. Data never leaves the network.

KEY CHALLENGES & MITIGATIONS

Challenge	Mitigation
LLMs are non-deterministic PLC/robot code must be deterministic	Strict RT/NRT partitioning. LLM generates; simulation verifies. Human sign-off gate.
Vendor heterogeneity Dozens of PLC/robot/camera SDKs	Adapter pattern from Interface Contracts. Normalize to canonical AG hardware modules.
Safety-critical outputs AI-generated code controls physical actuators	Safety contracts as non-bypassable assertions. Simulation gates. No deploy without green.
Hallucination risk LLM invents non-existent AG nodes	Agent must read docs first. Compiler catches invalid syntax. Constrained DSL limits scope.
Change control Who approves auto-generated code?	Human sign-off gates in pipeline. Traceability: requirement ↔ test ↔ code.

WHY THIS WORKS WITH ACTIONGRAPH

AG PROPERTIES THAT ENABLE AI

- **Declarative YAML** — LLMs generate structured data reliably
- **Constrained DSL** — fewer valid programs = higher accuracy
- **Process model** — natural mapping to LLM task decomposition
- **Built-in simulator** — immediate verification of generated code
- **No dynamic memory** — no runtime surprises

WHAT WOULD BE HARDER WITHOUT AG

- General-purpose C/C++ — infinite code space, hard to verify
- Proprietary PLC IDEs — no API for AI tool integration
- Ladder logic — visual, not text-generatable
- Imperative languages — state management errors
- No simulator — can't test without hardware

CURRENT STATE & ROADMAP

AVAILABLE TODAY

- LLM agent rules ([ag_llm_rules.md](#))
- LLM-optimized documentation set
- Agent workflow: YAML verification
- Claude Code integration (settings, permissions)
- AI-generated AG code examples
- Python simulator for SIL testing
- AG compiler with structured error output

PLANNED

- MCP servers for compiler & simulator
- Docs MCP server with hybrid RAG
- IDE-integrated chat assistant
- Automated test generation
- Contract-based system spec
- SIL → HIL simulation ladder
- Multi-vendor adapter generation

KEY TAKEAWAYS

1. ActionGraph is designed for AI from the ground up

Declarative YAML, constrained DSL, process-oriented model, structured docs

2. Documentation = the AI integration layer

No separate prompt engineering. Same docs serve humans and agents.

3. MCP makes it vendor-neutral

Tools exposed as standard servers. Works with Claude, GPT, Copilot, or any future LLM.

4. Verification closes the safety gap

AI generates code; simulation proves it correct. No trust-and-deploy.

5. The pattern generalizes beyond AG

Contract-based architecture applies to any industrial RT/NRT hybrid system.

ACTIONGRAPH + AI

Process-Oriented. Verifiable. AI-Ready.

Questions?