

ACTIONGRAPH

A Process-Oriented Language for Realtime Automation

Key Concepts Overview

THE PROBLEM

Controlling robots requires two conflicting things:

Hard Realtime

Microsecond-level timing for motion control, I/O, and safety

High-Level Flexibility

Easy to describe tasks, change behavior, and coordinate processes

Traditional approaches force you to choose one or the other.

WHY NOT EXISTING SOLUTIONS?

Approach	Strength	Limitation for Automation
G-Code	Industry standard, precise toolpaths	Too low-level for concurrent processes, state machines, error handling
Embedded Scripting (Lua, MicroPython)	High-level abstraction, flexible	Dynamic memory, unpredictable timing, cognitive overhead for simple tasks
PLC Languages (Ladder, ST)	Proven in industry	Poor composability, hard to express concurrent workflows declaratively

THE CORE INSIGHT

Robot control is about **processes**, not **computation**.

Most robot programs are naturally described as:

- "First do A, *then* do B"
- "Do X and Y *at the same time*"
- "*When* sensor triggers, start the sequence"
- "*If* error occurs, abort and go to safe position"

This is the language of processes — not function calls and variable assignments.

WHAT IS ACTIONGRAPH?

A domain-specific language + runtime for realtime automation.

EVENT-DRIVEN GRAPHS

Programs are graphs of nodes connected by events — inspired by Petri Nets.

PROCESS-ORIENTED

Describe *what* should happen, not *how*. The runtime handles execution.

REALTIME GUARANTEED

No dynamic memory, deterministic tick-based execution, predictable timing.

HYBRID ARCHITECTURE

Runs standalone on MCUs *or* bridges with Python on a host PC.

EXECUTION MODEL

Every node is a state machine with well-defined transitions:



- `.started` — began execution
 - `.stopped` — completed normally
- Events (outputs):
- `.erred` — encountered error

- `.start` — begin execution
- `.stop` — graceful stop

- Slots (inputs):
- `.cancel` — immediate cancel

CONNECTIONS: WIRING EVENTS TO SLOTS

Connections define control flow by wiring events to slots:

Connections:

```
.started: .nodeA.start      # Graph starts → start A
.nodeA.stopped: .nodeB.start # A finishes → start B
.nodeB.stopped: .stop       # B finishes → stop graph
```

Combinators:

- **All**: — fire when *every* event has occurred
- **Any**: — fire when *any* event occurs

- **Sequential**: — auto-chains nodes

Syntactic Sugar:

- **Parallel**: — auto-fans out + joins

WHAT IT LOOKS LIKE

A pick-and-place operation in ActionGraph:

Sequential:

- MOVEMENT: {movable: arm, target: pick_position}
- DIGITAL_SIGNAL_VALUE_SETTER: {signal: gripper, value: 1}
- DELAY: {delay: 100}
- MOVEMENT: {movable: arm, target: place_position}
- DIGITAL_SIGNAL_VALUE_SETTER: {signal: gripper, value: 0}

No explicit state machines. No callback registration. No manual memory management.
Just a **declaration of what should happen**.

THINKING IN PROCESSES

IMPERATIVE (DON'T)

1. Initialize motor 1
2. Initialize motor 2
3. Move motor 1 to A
4. Wait for motor 1
5. Move motor 2 to B
6. ...

ACTIONGRAPH (DO)

- What **concurrent processes** exist?
- What are their **responsibilities**?
- What **events** trigger them?
- What **dependencies** exist?

Decompose tasks into independent processes with clear boundaries — not sequential steps.

COMMON PATTERNS

PARALLEL WORKERS + SYNC

"Home all 6 axes, *then* run main program."

Uses `Parallel` + `All` combinator.

SEQUENTIAL STATE MACHINE

"Fast approach → back off → slow approach → set zero."

Chain via `.stopped` → `.start`.

EVENT-DRIVEN ARMED EXECUTION

"Prepare all axes, then trigger simultaneous start."

Custom slots + external trigger.

CONTINUOUS MONITORING

"Report position every 100ms while running."

Parallel reporter with self-loop.

HARDWARE ABSTRACTION

Program logic is fully separated from physical hardware:

AG Program Logic — Graphs, Nodes, Connections

AG Hardware Modules — Actuators, Sensors (standard library)

HAL Layer — 5 signal types: analog/digital × in/out + requests

Physical Hardware — Motors, encoders, sensors, I/O

Same AG program runs on real hardware *or* a Python simulator — no code changes.

HYBRID ARCHITECTURE

ActionGraph bridges realtime control with high-level software:

AG VM (Realtime)

Runs on MCU or RT Linux
Deterministic timing
Motion, I/O, safety

Python Host (Non-RT)

UI, network, database
High-level coordination
Monitoring & analytics

Communication via **4 mechanisms only**:

Set Parameter

Get Parameter

Send Event

Receive Event

SCALABILITY: THE REPEATER MACRO

Define N similar components from a single template:

```
DefaultMacroDefinitions:
```

```
  ACTUATORS_COUNT: 10
```

```
Parallel:
```

```
  Repeater(ACTUATORS_COUNT):
```

```
    - Type: WaveSingle
```

```
    Parameters:
```

```
      index: <<index>>
```

```
      movable: Hardware.Actuator<<index>>
```

Change **ACTUATORS_COUNT** from 10 to 100 — **zero code changes** needed.
Macro constants are shared between AG and Python via the generated API.

DEVELOPMENT WORKFLOW



YAML SOURCE

Declarative program definition — human-readable, version-controllable, diffable.

COMPILER

Transforms YAML into VM bytecode. Validates structure at compile time.

PYTHON SIMULATOR

Replace hardware with Python callbacks. Test logic without physical risk.

STANDARD LIBRARY

Reusable packages: actuators, calibration routines, pick-and-place operations.

DELIBERATE TRADEOFFS

ActionGraph does *less* so it can *guarantee more*:

Constraint	What You Get
No general-purpose computation	Deterministic timing, no unexpected behavior
No dynamic memory allocation	No memory leaks, no fragmentation, predictable footprint
Limited feature set	Small attack surface, easy to audit, fast to learn
YAML-only syntax	Human-readable, machine-parseable, diffable
4-channel Python interface	Clean boundary between RT and non-RT, no hidden coupling

Use Python for computation. Use ActionGraph for processes.

USE CASES

ROBOTICS

Delta robots, SCARA arms, multi-axis systems with jerk-limited motion planning.

INDUSTRIAL AUTOMATION

PLC-style control with composable, reusable process definitions.

EMBEDDED SYSTEMS

Bare-metal MCUs with hard realtime requirements and minimal resources.

HYBRID SYSTEMS

RT control + Python UI, analytics, cloud connectivity, and ML integration.

KEY TAKEAWAYS

1. Process-oriented, not computation-oriented

Describe what should happen; the runtime handles how.

2. Realtime by design

No dynamic memory, deterministic execution, guaranteed timing.

3. Clean separation of concerns

Logic vs. hardware vs. high-level coordination — each at the right layer.

4. Scales from simple to complex

Same model for 1 LED or 100 coordinated actuators.

5. Constraints are features

Doing less means guaranteeing more — safety, predictability, reliability.

ACTIONGRAPH

A Process-Oriented Language for Realtime Automation

Questions?