Agent Orchestration and Design Patterns in AG2

Estimated duration: 10 minutes

Learning objectives

At the end of this reading, you will be able to:

- Describe the key orchestration patterns in AG2
- Identify use cases and methods for implementing orchestration patterns

Overview

AG2 (formerly AutoGen) provides a powerful and flexible architecture for orchestrating AI agents through structured interaction patterns, safe-guards, and termination controls. This article introduces AG2's core orchestration designs.

---

1. Two-Agent Chat

Definition
The simplest orchestration where two agents engage directly in a back-and-forth conversation.

How it works

- An agent initiates the conversation using `initiate_chat()`, specifying a recipient, an initial message, and optional settings such as `max_turns` and a summarizer method
- The two agents exchange messages until the maximum number of turns is reached
- After the interaction, the conversation is summarized using the specified strategy, such as LLM-based reflection

Use Case
A student agent asks a teacher agent to explain the triangle inequality theorem. The interaction involves a few clarification messages and concludes with a summary.

---

2. Sequential Chat

Definition
Chains multiple two-agent chats in a specific order, passing the result (called "carryover") from one chat to the next.

How it works

- The initiating agent calls `initiate_chats()` with a list of recipient agents and corresponding message settings
- Each agent interaction happens in sequence
- The output summary of each interaction is passed forward as input context to the next

Use Case
A document passes through stages: content ideation by one agent, followed by drafting by another, and finally formatting by a third. Each stage builds on the prior result.

---

3. Nested Chat

Definition
Encapsulates a complex multi-agent workflow under a single "trigger" agent, making it reusable and modular.

How it works

- The main agent registers the nested workflow using `register_nested_chats()`
- When triggered, the nested interaction (which can itself be sequential or a group chat) runs internally
- The final result is returned or summarized for the outer context

Use Case
A curriculum_planner agent delegates subject-specific planning to sub-agents like math_planner and history_planner. The nested workflow builds a comprehensive lesson plan.

---

4. Group Chats

Definition
Multiple agents interact within a shared conversation space managed by a `GroupChatManager`.

How it works

- A `GroupChat` is instantiated with participating agents and a speaking strategy (pattern)
- A `GroupChatManager` controls the turn-taking logic based on the chosen orchestration pattern
- The conversation proceeds until a defined termination condition is met

Use Case
A support team chat includes `triage_agent`, `tech_support_agent`, and `general_support_agent`. Issues are dynamically escalated or resolved based on the conversation flow.

4a. Orchestration Patterns

Group chats use customizable patterns to determine speaker order.

Pattern	Description	When to Use
`DefaultPattern`	Requires explicit agent handoffs	Strict workflows with predictable transitions
`AutoPattern`	LLM selects next speaker from context	Adaptive, dynamic multi-agent conversations
`RoundRobinPattern`	Rotates turns in fixed sequence	Structured updates or status meetings
`RandomPattern`	Random agent selection (excluding current)	Brainstorming or collecting varied input
`ManualPattern`	Prompts user to choose next speaker	Educational settings or when human oversight is needed

---

4b. Tools & Functions

Definition
Agents can invoke tools or functions and use `ReplyResult` to structure the reply and guide the next step in the conversation.

How it works

- Tools return a `ReplyResult` object, which includes:
  - A response message,
  - A `transition` target to control the next speaker,
  - Optional updates to shared `context_variables`.

Use Case
A classifier function is called to assess the topic of a user query and decide whether to route it to a `finance_agent` or an `hr_agent`.

4c. Context Variables

Definition
Shared memory for group workflows using the `ContextVariables` class.

Features

- Maintained as a key-value store
- Accessible across agents and orchestration patterns
- Not automatically injected into LLM prompts unless explicitly referenced
- Persisted across tool calls and interactions

Use Case
A variable like `issue_severity` is updated as agents assess the problem. If the severity reaches a threshold, routing logic changes to involve an escalation agent.

4d. Handoffs & Routing

Definition
Defines which agent speaks next using rules applied to context or message content.

Routing Methods

- **LLM-Based Routing:** Uses `OnCondition` to evaluate the message via LLM.
- **Context-Based Routing:** Uses `OnContextCondition` to check values in `context_variables`.
- **After-Work/Default Routing:** Specifies fallback behavior when no conditions are met.
- **Tool-Based Routing:** Tools return `ReplyResult` with a `transition` value indicating the next agent.

Use Case
If a support conversation includes the phrase "urgent outage", `OnCondition` evaluates this and transitions control to an escalation_agent.

---

5. Guardrails

Definition
Safety mechanisms that intercept conversations based on rules, redirecting control when needed.

Types

- **RegexGuardrail:** Uses pattern matching to detect items like phone numbers or social security numbers.
- **LLMGuardrail:** Applies semantic filtering using LLMs to detect unsafe or inappropriate content.

Behavior

- Guardrails can be registered on agent inputs or outputs
- On detection, control is redirected to a designated safety or compliance agent

---

Ending a Chat

AG2 offers several termination mechanisms:

- **Maximum Turns:** Set via `max_turns` (two-agent) or `max_round` (group chat).
- **Termination Messages:** A message like "DONE!" can trigger `is_termination_msg` to end the chat.
- **Max Auto-Replies:** Agents can be configured with `max_consecutive_auto_reply` to avoid infinite loops.
- **User-Initiated Exit:** If `human_input_mode` is set to `"ALWAYS"` or `"TERMINATE"`, typing `"exit"` ends the session.
- **No Next Agent:** If the pattern returns `None`, the conversation halts.
- **TerminateTarget Transition:** Used as a fallback when no further handoff is possible.
- **Custom Reply Logic:** Agents can return `(True, None)` to signal an intentional end of conversation.

Summary Table

Architecture	Ideal Use Case	Core Method/Component
Two-Agent Chat	Quick Q&A or task delegation	`initiate_chat()` with `max_turns`
Sequential Chat	Pipelines and multi-step workflows	`initiate_chats()` with carryover
Nested Chat	Encapsulation of reusable workflows	`register_nested_chats()`
Group Chat	Multi-role collaboration	`GroupChatManager`, orchestration patterns
Tools & Functions	Custom logic, branching control	`ReplyResult`
Context Variables	Shared state, routing decisions	`ContextVariables`
Handoffs & Routing	Conditional speaker transitions	`OnCondition`, `OnContextCondition`

Conclusion

AG2's orchestration framework enables the design of sophisticated multi-agent systems by combining reusable interaction patterns, conditional routing, contextual memory, tool integration, and structured terminations. Whether creating a simple dialogue or a layered automation pipeline, AG2 offers composable building blocks to develop robust and adaptive agent ecosystems.

Reference

All content in this article is derived from the official AG2 documentation.

Author(s)

Faranak Heidari is an AI developer at IBM.