Dialogues

Overview

Dialogues are structured and optimized way for agents to communicate. It provides an approach for agents to communicate with each other with consistent message pattern, allowing for multi-message sessions, state tracking, and parallel conversations.

Understanding Dialogues

Dialogues can be taken as super-powered Protocols, It provides more robust structure for communication between agents. Protocol define interaction patterns but lack state enforcement, leaving developers to manage all possible messaging cases. Dialogues address this by offering predefined patterns and clear state management.

Difference between Dialogues and Protocols

- Protocols :
- Protocols provide full flexibility to interactions between agents. Developers are responsible for ensuring communication is working correctly by defining right data models and message handlers.
- Dialogues :
- Dialogues add a more rigid structure to communication with enforced state transitions, multi-session state tracking and reusable complex communication patterns. Technically speaking, Dialogues are represented by a directed graph where nodes are the states and edges the messages (message handlers) of a well-defined communication flow.

Communication Patterns

Communication patterns in dialogues define a general graph-based structure for interactions. They describe every possible flow and branch in the communication process, providing a clear entry, transition and exit point. Patterns are message-agnostic and can be instantiated with various message models as specific Dialogues, adapting to different use cases while maintaining consistent principles.

Application and Core Developers

The distinction between communication patterns and Dialogues supports two typical development scenarios with uagents:

- Application Developers:
- Have clear business requirements and are looking to implement concrete use cases with agents as quickly and as easily as possible. They don't want to waste time for creating the necessary building blocks or defining fundamental yet complex processes. Patterns help streamline their workflow, taking the burden of designing communication structures off them.
- Core Developers:
- Improve the infrastructure for agent ecosystems by carefully defining and hardening relevant building blocks, needing full flexibility and power to lay the groundwork for application developers such as defining sound and complete communication patterns that can be reused for various use cases following the same principle.

How to use dialogues

As explained above dialogues can be used by developers with different requirements. There are different ways in which dialogues can be used but before checking that lets understand what doesstateful communication means in agent communication ecosystem.

Stateful Communication

Dialogues support stateful communication which allows agents to access past messages and state status, perform parallel conversations/sessions and also it enforces the sequence of message exchanges. This overall structure and feature of dialogue helps maintain context, improves irrefutability and allows agents to resume conversation from any point of time.

There are different ways in which dialogues can be used as listed below:

API for Dialogues

Some of the Key APIs exposed to application developers in dialogues include:

- start_dialogue:

- Initiates a dialogue, offering a more explicit way to start communication compared to protocols. Refer defining dialogue structure
- to have better understanding of how to define dialogue nodes and create a handler for event ↗
- .
- get_conversation:
- Provides access to past messages, useful for handling complex scenarios and multiple parallel dialogues. Refer agents on_continue_dialogue handler↗
- on how to access history using context's dialogue function.
- reject_session:
- This is the transition for when the dialogue is rejected.

Similarly you can refer more functions and handler in examples Open Dialogues ↗ and Predefined Dialogues ↗ .

Customization and Predefined Behaviors

Dialogues allow for customization by adding or overwriting message handlers. Predefined behaviors make it easier for application developers to work with dialogues, handling standard interactions while enabling specific customization for unique use-cases.

Sample for creating message handler

Defining dialogue class

from uagents . experimental . dialogues import Dialogue , Edge , Node

class

ChitChatDialogueMessage (Model): text :

str

Defining dialogues class

class

ChitChatDialogue (Dialogue): def

init (self ,

version :

str

|

None

=

None ,

agent_address :

str

|

None

=

None) ->

None : super (). **init** (name = "ChitChatDialogue" , version = version, agent_address = agent_address, nodes = [node1, node2, node3], edges = [init_session, start_dialogue, cont_dialogue, end_session],)

def

on_continue_dialogue (self): return

super (). _on_state_transition (cont_dialogue.name, ChitChatDialogueMessage,)

Instantiate dialogues and defining continue dialogue handler/

from dialogues . hardcoded_chitchat import (ChitChatDialogue , ChitChatDialogueMessage)

Instantiate the dialogues

chitchat_dialogue

ChitChatDialogue (version = "0.1" , agent_address =< agent 's address>,)

@chitchat_dialogue . on_continue_dialogue () async

def

continue_chitchat (ctx : Context , sender :

str , msg : ChitChatDialogueMessage ,): ctx . logger . info (f "Returning: { msg.text } ") await ctx . send (sender, ChitChatDialogueMessage (text = msg.text))

Example to use dialogues

The following examples demonstrate how dialogues can be used in practical scenarios:

- Open Dialogue Chit-Chat ↗
- : This example shows an interactive communication setup between agents, where each step and state transition is visible. It is ideal for understanding the flow of messages and state transitions in real-time.
- Predefined Dialogue Chit-Chat ↗
- : This example illustrates an automated dialogue scenario with hardcoded dialogue management between two agents. It minimizes manual intervention, focusing on predefined responses and interactions.

This guide explains the purpose and key aspects of dialogues, drawing on developer comments and feedback. It discusses communication patterns, stateful communication, API highlights, and customization to guide both application and core developers.

ℹ This guide is part of an experimental feature under heavy development. Backward compatibility is not guaranteed, and updates may cause breaking changes. Each update might introduce significant changes in behavior or functionality.

Was this page helpful?

Make agents AI Engine compatible Almanac Contract