Human-in-the-Loop Workflows with State Updates in Google ADK

Overview

This guide shows how to implement human-in-the-loop workflows in Google ADK where users can edit data and have those changes flow to subsequent agents. Based on real implementation experience.

Core Problem

In ADK, when you pause a workflow for human approval and the user edits data, getting those changes to subsequent agents requires specific patterns. Direct session state modification **does not work**.

Simple Architecture

Agent A → Creates Data → Session State
Agent B → Shows Form → User Edits Data
Resume → Updates State → Agent C Gets Edited Data

Working Implementation

1. Data Producer Agent

```
def create_order_data(tool_context: ToolContext) -> dict:
    """Creates order data and stores it in session state."""
    order_data = {
        "customer": "John Doe",
        "amount": 100.00,
        "items": ["Widget A", "Widget B"]
    }

    # Store in session state for next agent
    tool_context.state["order_data"] = order_data

    return {"status": "success", "data": order_data}
```

```
producer_agent = Agent(
    name="data_producer",
    model="gemini-2.0-flash",
    instruction="Create order data and store it.",
    tools=[create_order_data]
)
```

2. Human Approval Agent

```
# Global state for tracking approvals
PENDING_APPROVALS = {}

def approval_form(purpose: str, data_json: str, tool_context: ToolContext) -> dict:
    """Shows approval form and pauses execution."""

    # Parse the data
    data = json.loads(data_json) if isinstance(data_json, str) else data_json

    # Generate approval ID
    approval_id = f"approval_{int(time.time())}"

    # Store in global state
    PENDING_APPROVALS[approval_id] = {
        "status": "pending",
        "data": data,
        "created_at": time.time()
    }

    # Show interactive form
    display_approval_form(data, approval_id)

    # Return pending status - this pauses agent execution
    return {
        "status": "pending",
        "approval_id": approval_id,
        "message": "Approval form displayed. Execution paused."
    }

def display_approval_form(data, approval_id):
    """Display interactive form with edit capabilities."""

    # Create editable widgets
    customer_input = widgets.Text(value=data.get("customer", ""), description="Customer:")
    amount_input = widgets.FloatText(value=data.get("amount", 0), description="Amount:")
```

```
    # Buttons
    approve_btn = widgets.Button(description="✅ Approve", button_style='success')
    reject_btn = widgets.Button(description="❌ Reject", button_style='danger')

    def on_approve(b):
        # Update data with user edits
        updated_data = data.copy()
        updated_data["customer"] = customer_input.value
        updated_data["amount"] = amount_input.value

        # Update global state with edited data
        PENDING_APPROVALS[approval_id]["status"] = "approved"
        PENDING_APPROVALS[approval_id]["data"] = updated_data

        print(f"✅ Approved! ID: {approval_id}")
        approve_btn.disabled = True
        reject_btn.disabled = True

    def on_reject(b):
        PENDING_APPROVALS[approval_id]["status"] = "rejected"
        print(f"❌ Rejected! ID: {approval_id}")
        approve_btn.disabled = True
        reject_btn.disabled = True

    approve_btn.on_click(on_approve)
    reject_btn.on_click(on_reject)

    # Display form
    display(widgets.HTML("<h3>Order Approval Required</h3>"))
    display(customer_input, amount_input)
    display(widgets.HBox([approve_btn, reject_btn]))

# Create long-running tool
approval_tool = LongRunningFunctionTool(func=approval_form)

approval_agent = Agent(
    name="approver",
    model="gemini-2.0-flash",
    instruction="Show approval form for the order data from previous agent.",
    tools=[approval_tool]
)
```

3. Resume Function with EventActions

```
async def resume_workflow():
    """Resume workflow with approved (edited) data."""

    # Get approved data from global state
    approval_id = "your_approval_id"  # Get from your workflow tracking
    approval_info = PENDING_APPROVALS[approval_id]

    if approval_info["status"] != "approved":
        print("❌ Not approved yet")
        return

    approved_data = approval_info["data"]  # Contains user edits

    # CRITICAL: Use EventActions to update session state
    from google.adk.events import Event, EventActions
    import time

    state_changes = {
        "order_data": approved_data,  # Replace original with approved data
        "approval_status": "approved"
    }

    # Create event with state changes
    actions = EventActions(state_delta=state_changes)
    event = Event(
        invocation_id="approval_update",
        author="system",
        actions=actions,
        timestamp=time.time()
    )

    # Apply changes through ADK event system
    await session_service.append_event(session, event)
    print("✅ State updated with approved data")

    # Resume workflow with simple message
    resume_content = types.Content(
        role='user',
        parts=[types.Part(text="Data approved. Continue.")]
    )

    # Continue workflow
    resume_events = runner.run_async(
```

```
        user_id="user1",
        session_id=session.id,
        new_message=resume_content
    )

    async for event in resume_events:
        # Process resumed execution
        print(f"[{event.author}]: {event.content}")
```

4. Data Consumer Agent

```
def process_approved_data(tool_context: ToolContext) -> str:
    """Process the approved (user-edited) order data."""

    # Get data from session state (now contains user edits)
    order_data = tool_context.state.get("order_data")

    if not order_data:
        return "No order data found"

    print(f"Processing order for: {order_data['customer']}")
    print(f"Amount: ${order_data['amount']}")
    print(f"Items: {order_data['items']}")

    return f"Order processed for {order_data['customer']}"

consumer_agent = Agent(
    name="processor",
    model="gemini-2.0-flash",
    instruction="Process the approved order data.",
    tools=[process_approved_data]
)
```

5. Sequential Workflow

```
workflow = SequentialAgent(
    name="approval_workflow",
    sub_agents=[
        producer_agent,    # Creates data
        approval_agent,    # Human approval
        consumer_agent     # Processes approved data
    ]
)
```

Critical Pitfalls

❌ Pitfall 1: Direct State Modification

```
# THIS DOESN'T WORK
session.state["order_data"] = approved_data
```

✅ Solution: Use EventActions

```
# THIS WORKS
state_changes = {"order_data": approved_data}
actions = EventActions(state_delta=state_changes)
event = Event(invocation_id="update", author="system", actions=actions,
timestamp=time.time())
await session_service.append_event(session, event)
```

❌ Pitfall 2: Manual Function Responses

```
# THIS BREAKS ADK
response = types.FunctionResponse(id="some_id", name="func", response={})
```

✅ Solution: Simple Text Messages

```
# THIS WORKS
content = types.Content(role='user', parts=[types.Part(text="Continue workflow")])
```

❌ Pitfall 3: Wrong Data Access

```
# Agent can't see updates made this way
session.state["data"] = new_data
```

✅ Solution: EventActions Updates

```
# Agent sees updates made this way
await session_service.append_event(session, event_with_state_delta)
```

Troubleshooting

Problem: Agent doesn't see user edits

Symptoms: Debug shows edited data, but final agent sees original data

Solution: Use EventActions pattern to update session state

Problem: "Function call not found" errors

Symptoms: `ValueError: No function call event found`

Solution: Don't create manual function responses, use simple text messages

Problem: Form edits not saved

Symptoms: User changes form but data unchanged

Solution: Check button click handlers update global approval state

Minimal Example

```
# 1. Store data
tool_context.state["my_data"] = {"value": "original"}

# 2. User edits in form (stored in global state)
PENDING_APPROVALS[id]["data"] = {"value": "edited"}

# 3. Update session state with EventActions
state_changes = {"my_data": {"value": "edited"}}
actions = EventActions(state_delta=state_changes)
event = Event(invocation_id="update", author="system", actions=actions,
timestamp=time.time())
await session_service.append_event(session, event)

# 4. Next agent sees edited data
edited_data = tool_context.state.get("my_data")  # {"value": "edited"}
```

Key Takeaway

The only way to pass user-edited data between agents in ADK is through the EventActions + append_event pattern. Direct session state modification will not work.

This pattern ensures that user edits flow properly through the ADK event system and are available to subsequent agents in your workflow.