Tutorial 17: Agent-to-Agent Communication - Distributed Systems

Difficulty: advanced

Reading Time: 2 hours

Tags: advanced, agent-communication, distributed, delegation, coordination

Description: Build distributed agent systems with agent-to-agent communication, delegation, and coordination for complex multi-agent orchestration.

:::info FULLY WORKING A2A IMPLEMENTATION - TESTED & VERIFIED

This tutorial features a complete, tested A2A implementation using the official Google ADK.

✓ **All Issues Resolved**: Agent card URLs fixed, context handling improved
✓ **Working End-to-End**: Complete orchestration with meaningful responses
✓ **Tested Implementation**: Real working code with successful test results

Key approach: `uvicorn + to_a2a()` for servers, `RemoteA2aAgent` for clients, with proper A2A context handling for intelligent remote agent responses.

Latest Update: January 10, 2025 - Fixed context handling for production-ready A2A.

:::

Tutorial 17: Agent-to-Agent (A2A) Communication

Goal: Enable agents to communicate and collaborate with other remote agents using the **official ADK Agent-to-Agent (A2A) protocol**, creating distributed multi-agent systems with the built-in `RemoteA2aAgent` class.

Prerequisites:

- Tutorial 01 (Hello World Agent)
- Tutorial 06 (Multi-Agent Systems)
- Understanding of HTTP APIs and authentication
- Basic knowledge of REST principles

What You'll Learn:

- Understanding the official ADK Agent-to-Agent (A2A) protocol
- Using `RemoteA2aAgent` to communicate with remote agents
- Setting up A2A servers with ADK's built-in `api_server --a2a` command
- Agent discovery with official agent cards (`.well-known/agent-card.json`)
- Building distributed agent orchestration with the official ADK approach
- Error handling in A2A communication using ADK patterns
- Best practices for production A2A systems with ADK

Time to Complete: 50-65 minutes

Why A2A Matters

Problem: Agents are often isolated - they can't leverage capabilities of other specialized agents deployed elsewhere.

Solution: **Agent-to-Agent (A2A)** protocol enables agents to discover and communicate with remote agents over HTTP, creating distributed AI systems.

Benefits:

- 🌐 **Distributed Intelligence**: Leverage agents across organizations
- 🔍 **Discovery**: Find agents by capability via agent cards
- 🔐 **Secure**: Built-in authentication and authorization
- 🎯 **Specialization**: Each agent focuses on its expertise
- [FLOW] **Reusability**: Use same agent from multiple orchestrators
- ⚡ **Scalability**: Scale agents independently

Use Cases:

- Enterprise: Customer service agent calls internal knowledge agent
- Multi-org: Legal agent consults external compliance agent
- Microservices: Specialized agents as independent services
- Multi-cloud: Agents distributed across cloud providers

A2A System Architecture:

```
┌─────────────────┐    ┌─────────────────┐    ┌─────────────────┐
│   Orchestrator  │────│ RemoteA2aAgent  │────│ Remote Agent A  │
│   Agent         │    │  (ADK Built-in) │    │  (Specialized)  │
└─────────────────┘    └─────────────────┘    └─────────────────┘
         │                        │                        │
         │                        │                        │
┌─────────────────┐    ┌─────────────────┐    ┌─────────────────┐
│   Orchestrator  │────│ RemoteA2aAgent  │────│ Remote Agent B  │
│   Agent         │    │  (ADK Built-in) │    │  (Specialized)  │
└─────────────────┘    └─────────────────┘    └─────────────────┘
         │                        │                        │
         │                        │                        │
┌─────────────────┐    ┌─────────────────┐    ┌─────────────────┐
│   Orchestrator  │────│ RemoteA2aAgent  │────│ Remote Agent C  │
│   Agent         │    │  (ADK Built-in) │    │  (Specialized)  │
└─────────────────┘    └─────────────────┘    └─────────────────┘

    HTTP/A2A Protocol        Auto-Discovery       Specialization
    Communication            via Agent Cards      by Expertise
```

1. A2A Protocol Basics

What is Agent-to-Agent Protocol?

A2A defines a standard way for agents to:

1. **Discover** other agents via agent cards
2. **Authenticate** with other agents
3. **Invoke** remote agent capabilities

4. **Receive** responses from remote agents

Architecture with Working ADK Implementation:

```
Local Agent (Orchestrator)
    ↓
RemoteA2aAgent (ADK Built-in)
    ↓
HTTP Request with A2A Protocol
    ↓
Remote A2A Server (uvicorn + to_a2a())
    ↓
Remote Agent Execution
    ↓
Response back to Local Agent
```

Source: ADK Built-in `RemoteA2aAgent` class + `to_a2a()` function

Agent Card (Discovery)

Remote agents expose an **agent card** at `.well-known/agent-card.json`:

```
{
  "capabilities": {},
  "defaultInputModes": ["text/plain"],
  "defaultOutputModes": ["application/json"],
  "description": "Conducts web research and fact-checking",
  "name": "research_specialist",
  "url": "http://localhost:8001/a2a/research_specialist",
  "version": "1.0.0",
  "skills": [
    {
      "id": "research_web",
      "name": "Web Research",
      "description": "Research topics using web sources",
      "tags": ["research", "web", "information"]
    }
  ]
}
```

Well-Known Path:

```
# Standard location for agent cards in ADK
# http://localhost:8001/.well-known/agent-card.json
# Note: "agent-card.json" not "agent.json"
```

2. Using Official ADK A2A with RemoteA2aAgent

Basic Setup

```
from google.adk.agents import Agent
from google.adk.agents.remote_a2a_agent import RemoteA2aAgent, AGENT_CARD_WELL
from google.adk.tools import FunctionTool

# Create remote agent using official ADK RemoteA2aAgent
research_agent = RemoteA2aAgent(
    name="research_specialist",
    description="Conducts web research and fact-checking",
    agent_card=(
        f"http://localhost:8001/a2a/research_specialist{AGENT_CARD_WELL_KNOWN_
    )
)

# Use as sub-agent in orchestrator
orchestrator = Agent(
    model='gemini-2.0-flash',
    name='a2a_orchestrator',
    instruction="""
You coordinate research tasks using remote A2A agents.
Delegate research tasks to the research_specialist sub-agent.
    """,
    sub_agents=[research_agent]  # Use as sub-agent
)
```

How It Works

Step-by-Step Flow with Official ADK:

1. **Discovery**: ADK fetches agent card from `.well-known/agent-card.json`
2. **RemoteA2aAgent**: ADK's built-in class handles A2A communication
3. **Sub-Agent Integration**: Remote agent works like any other sub-agent
4. **Invocation**: ADK handles protocol details automatically
5. **Execution**: Remote agent processes the request via A2A server
6. **Response**: ADK extracts response and integrates into workflow

Internal ADK Flow (managed automatically):

```python
# ADK handles this internally in RemoteA2aAgent
class RemoteA2aAgent:
    def __init__(self, name: str, description: str, agent_card: str):
        self.name = name
        self.description = description
        self.agent_card_url = agent_card
        # ADK manages HTTP client, authentication, and protocol details

    async def invoke(self, query: str) -> str:
        # ADK automatically:
        # 1. Fetches agent card
        # 2. Handles A2A protocol communication
        # 3. Manages authentication
        # 4. Extracts and returns response text
        pass
```

A2A Communication Flow:

```
┌─────────────────┐     ┌─────────────────┐     ┌─────────────────┐
│ Orchestrator    │     │ RemoteA2aAgent  │     │ Remote A2A      │
│ Agent           │────▶│ (ADK Built-in)  │────▶│ Server          │
│ "Research AI"   │     │                 │     │ (uvicorn +      │
└─────────────────┘     └─────────────────┘     │ to_a2a())       │
                                                       └─────────────────┘
                                                               │
                                                               ▼

┌─────────────────┐     ┌─────────────────┐     ┌─────────────────┐
│ 1. Discovery    │────▶│ 2. Fetch Agent  │────▶│ 3. A2A Protocol │
│    Request      │     │    Card         │     │    Message      │
│                 │     │                 │     │                 │
│ GET /.well-known│     │ {               │     │ {               │
│ /agent-card.json│     │   "name": "..." │     │   "query": "..." │
└─────────────────┘     │ }               │     │ }               │
                        └─────────────────┘     └─────────────────┘
                                                       │
                                                       ▼

┌─────────────────┐     ┌─────────────────┐     ┌─────────────────┐
│ Remote Agent    │     │ 4. Process      │────▶│ 5. Return       │
│ Execution       │     │    Request      │     │    Response     │
│                 │     │                 │     │                 │
│ Uses Tools &    │     │ AI Model +      │     │ Structured      │
│ Model           │     │ Functions       │     │ JSON Response   │
└─────────────────┘     └─────────────────┘     └─────────────────┘
                                                               │
                                                               ▼

┌─────────────────┐     ┌─────────────────┐     ┌─────────────────┐
│ 6. Extract &    │◀────│ 7. ADK Handles  │◀────│ 8. HTTP Response │
│    Return       │     │    Protocol     │     │    Back to       │
│ Response Text   │     │    Details      │     │    Orchestrator  │
└─────────────────┘     └─────────────────┘     └─────────────────┘
```

3. Complete Implementation: Official ADK A2A System

Let's examine the complete working implementation using the official ADK `to_a2a()` function and `RemoteA2aAgent` class that was successfully tested and deployed.

Complete Working Implementation

```python
"""
Working ADK A2A Orchestrator - Agent-to-Agent Communication

This demonstrates the working ADK approach to distributed agent orchestration
using RemoteA2aAgent and the to_a2a() function pattern.
"""

from google.adk.agents import Agent
from google.adk.agents.remote_a2a_agent import RemoteA2aAgent, AGENT_CARD_WELL
from google.adk.tools import FunctionTool
from google.genai import types

# Tool function to validate agent availability
def check_agent_availability(agent_name: str, base_url: str) -> dict:
    """Check if a remote A2A agent is available."""
    try:
        import requests
        card_url = f"{base_url}{AGENT_CARD_WELL_KNOWN_PATH}"
        response = requests.get(card_url, timeout=5)

        if response.status_code == 200:
            return {
                "status": "success",
                "available": True,
                "report": f"Agent {agent_name} is available",
                "agent_card": response.json()
            }
        else:
            return {
                "status": "error",
                "available": False,
                "report": f"Agent {agent_name} returned status {response.statu
            }
    except Exception as e:
        return {
            "status": "error",
            "available": False,
            "report": f"Failed to check {agent_name}: {str(e)}"
        }

# Remote agents using official ADK RemoteA2aAgent
research_agent = RemoteA2aAgent(
    name="research_specialist",
    description="Conducts web research and fact-checking",
    agent_card=(
        f"http://localhost:8001/a2a/research_specialist{AGENT_CARD_WELL_KNOWN_
    )
)

analysis_agent = RemoteA2aAgent(
    name="data_analyst",
    description="Analyzes data and generates insights",
    agent_card=(
        f"http://localhost:8002/a2a/data_analyst{AGENT_CARD_WELL_KNOWN_PATH}"
    )
)

content_agent = RemoteA2aAgent(
    name="content_writer",
    description="Creates written content and summaries",
    agent_card=(
        f"http://localhost:8003/a2a/content_writer{AGENT_CARD_WELL_KNOWN_PATH}
    )
)

# Main orchestrator agent using working ADK patterns
root_agent = Agent(
    model="gemini-2.0-flash",
    name="a2a_orchestrator",
    description="Coordinates multiple remote specialized agents using official
    instruction="""
You are an orchestration agent that coordinates specialized remote agents
using the official ADK Agent-to-Agent (A2A) protocol.

**Available Remote Agents (Sub-Agents):**

1. **research_specialist**: Use for web research, fact-checking, current event
2. **data_analyst**: Use for data analysis, statistics, insights
3. **content_writer**: Use for content creation, summaries, writing

**Working A2A Workflow:**
1. Delegate research tasks to research_specialist sub-agent
2. Delegate analysis tasks to data_analyst sub-agent
3. Delegate content creation to content_writer sub-agent
4. Use check_agent_availability to verify agent status

The remote agents are exposed using uvicorn + to_a2a() and work
seamlessly as sub-agents in your orchestration workflow.

Always explain which remote agent you're delegating to and why.
    """,
    sub_agents=[research_agent, analysis_agent, content_agent],
    tools=[FunctionTool(check_agent_availability)],
    generate_content_config=types.GenerateContentConfig(
        temperature=0.5,
        max_output_tokens=2048
    )
)
```

Quick Start Guide

1. **Setup Environment**:

```
# Install ADK with A2A support
pip install google-adk[a2a]

# Copy environment template
cp a2a_orchestrator/.env.example a2a_orchestrator/.env
# Edit .env and add your GOOGLE_API_KEY
```

1. **Start Remote A2A Agents**:

```
# Start research agent with uvicorn + to_a2a() function
uvicorn research_agent.agent:a2a_app --host localhost --port 8001

# Start analysis agent
uvicorn analysis_agent.agent:a2a_app --host localhost --port 8002

# Start content agent
uvicorn content_agent.agent:a2a_app --host localhost --port 8003

# Or use the provided script:
./start_a2a_servers.sh
```

1. **Verify Agent Status**:

```
# Check agent cards are available
curl http://localhost:8001/.well-known/agent-card.json
curl http://localhost:8002/.well-known/agent-card.json
curl http://localhost:8003/.well-known/agent-card.json
```

1. **Start Orchestrator**:

```
# Start ADK web interface
adk web a2a_orchestrator/
# Open http://localhost:8000 and select 'a2a_orchestrator'
```

1. **Test A2A Communication**:

```
# Run integration test
python -m pytest tests/test_a2a_integration.py -v
```

Expected Behavior

When you query: `"Research quantum computing trends and create a summary"`

The orchestrator will:

1. 🎯 Log coordination step: Starting research phase
2. 🤖 Delegate to research_specialist sub-agent (via RemoteA2aAgent)
3. 🎯 Log coordination step: Starting analysis phase
4. 🤖 Delegate to data_analyst sub-agent (via RemoteA2aAgent)
5. 🎯 Log coordination step: Creating content phase
6. 🤖 Delegate to content_writer sub-agent (via RemoteA2aAgent)
7. 📊 Return integrated final result

Note: All A2A communication is handled transparently by ADK's `RemoteA2aAgent` class - no manual protocol handling required!

Orchestration Workflow:

```
User Query: "Research quantum computing trends and create a summary"
                                                                 │
                                                                 ▼
┌─────────────────────────────────────────────────────────────────┐
│                    A2A_ORCHESTRATOR                             │
│                    (Main Coordinator)                           │
└─────────────────────────────────────────────────────────────────┘
                                 │
                                 ▼
┌─────────────────────────────────────────────────────────────────┐
│ 🎯 Step 1: Starting research phase                              │
│ 🤖 Delegate to research_specialist sub-agent (RemoteA2aAgent)   │
└─────────────────────────────────────────────────────────────────┘
                                 │
                                 ▼
┌─────────────────────────────────────────────────────────────────┐
│                    RESEARCH_SPECIALIST                          │
│                    (Remote Agent on :8001)                      │
│                                                                 │
│ 1. Receives A2A request via HTTP                                │
│ 2. Processes with Gemini model + research tools                 │
│ 3. Returns research findings                                    │
└─────────────────────────────────────────────────────────────────┘
                                 │
                                 ▼
┌─────────────────────────────────────────────────────────────────┐
│ 🎯 Step 2: Starting analysis phase                              │
│ 🤖 Delegate to data_analyst sub-agent (RemoteA2aAgent)          │
└─────────────────────────────────────────────────────────────────┘
                                 │
                                 ▼
┌─────────────────────────────────────────────────────────────────┐
│                    DATA_ANALYST                                  │
│                    (Remote Agent on :8002)                      │
│                                                                 │
│ 1. Receives A2A request via HTTP                                │
│ 2. Analyzes research data with analysis tools                   │
│ 3. Returns insights and trends                                  │
└─────────────────────────────────────────────────────────────────┘
                                 │
                                 ▼
┌─────────────────────────────────────────────────────────────────┐
│ 🎯 Step 3: Creating content phase                               │
│ 🤖 Delegate to content_writer sub-agent (RemoteA2aAgent)        │
└─────────────────────────────────────────────────────────────────┘
                                 │
```

```
                              ▼
┌──────────────────────────────────────────────────────────────┐
│                  CONTENT_WRITER                                │
│                  (Remote Agent on :8003)                      │
│                                                               │
│ 1. Receives A2A request via HTTP                              │
│ 2. Creates summary content using writing tools                │
│ 3. Returns formatted summary                                  │
└──────────────────────────────────────────────────────────────┘
                              │
                              ▼
┌──────────────────────────────────────────────────────────────┐
│ 📊 Final Result: Integrated quantum computing trends summary  │
│     combining research, analysis, and content creation       │
└──────────────────────────────────────────────────────────────┘
```

4. Critical: Proper A2A Context Handling

The Context Handling Challenge

When implementing A2A communication, remote agents receive the full conversation context including orchestrator tool calls. Without proper handling, remote agents may respond with errors like:

```
"I cannot use a tool called transfer_to_agent. The available tools lack
the ability to interact with other agents."
```

Solution: Smart Context Processing

Update all remote agents with **A2A Context Handling** instructions:

```
# Example for content_writer agent
root_agent = Agent(
    model="gemini-2.0-flash",
    name="content_writer",
    description="Creates written content and summaries",
    instruction="""
You are a content creation specialist focused on producing high-quality writte

**IMPORTANT - A2A Context Handling:**
When receiving requests via Agent-to-Agent (A2A) protocol, focus on the core u
Ignore any mentions of orchestrator tool calls like "transfer_to_agent" in the
Extract the main content creation task from the conversation and complete it d

**When working via A2A:**
- Focus on the actual content request from the user (e.g., "Write a report abo
- Ignore orchestrator mechanics and tool calls in the context
- Provide direct, helpful content creation services using your tools
- If the request is unclear, ask for clarification about the content type and

Always consider the target audience and intended use of the content.
    """,
    tools=[FunctionTool(create_content), FunctionTool(format_content)]
)
```

Context Handling Results

Before Fix:

```
User: "Write a report about AI"
→ Orchestrator calls transfer_to_agent
→ Remote agent: "I cannot use transfer_to_agent tool..."
```

After Fix:

```
User: "Write a report about AI"
→ Orchestrator calls transfer_to_agent
→ Remote agent extracts core request
→ Remote agent: [Creates AI report using create_content tool]
```

A2A Context Handling Flow:

```
BEFORE FIX (Broken):
┌─────────────────┐    ┌─────────────────┐    ┌─────────────────┐
│  User Request   │───▶│  Orchestrator   │───▶│ Remote Agent    │
│ "Write AI report"│   │ (A2A Context)   │    │ (Receives Full  │
└─────────────────┘    └─────────────────┘    │ Context)        │
                                                     └─────────────────┘
                                                         │
                                                         ▼
┌─────────────────┐    ┌─────────────────┐    ┌─────────────────┐
│ Orchestrator    │    │ Context Includes │   │ Remote Agent    │
│ Tool Call:      │───▶│ transfer_to_agent│──▶│ Sees:           │
│ transfer_to_agent│   │ (orchestrator    │   │ "transfer_to_   │
└─────────────────┘    │ mechanics)       │   │  agent tool"    │
                       └─────────────────┘    └─────────────────┘
                                                         │
                                                         ▼
┌─────────────────┐    ┌─────────────────┐    ┌─────────────────┐
│ Remote Agent    │    │ ❌ ERROR:       │───▶│ No Response or  │
│ Response        │◀───│ "I cannot use   │    │ Wrong Response  │
│                 │    │ transfer_to_agent│   └─────────────────┘
│ "I cannot use   │    └─────────────────┘
│ transfer_to_agent│
│ tool..."        │
└─────────────────┘

AFTER FIX (Working):
┌─────────────────┐    ┌─────────────────┐    ┌─────────────────┐
│  User Request   │───▶│  Orchestrator   │───▶│ Remote Agent    │
│ "Write AI report"│   │ (A2A Context)   │    │ (Smart Context  │
└─────────────────┘    └─────────────────┘    │ Handling)       │
                                                     └─────────────────┘
                                                         │
                                                         ▼
┌─────────────────┐    ┌─────────────────┐    ┌─────────────────┐
│ Orchestrator    │    │ Context Includes │   │ Remote Agent    │
│ Tool Call:      │───▶│ transfer_to_agent│──▶│ Instructions:   │
│ transfer_to_agent│   │ (orchestrator    │   │ "Ignore orch.   │
└─────────────────┘    │ mechanics)       │   │  mechanics"     │
                       └─────────────────┘    └─────────────────┘
                                                         │
                                                         ▼
┌─────────────────┐    ┌─────────────────┐    ┌─────────────────┐
│ Remote Agent    │    │ ✔ Extracts      │───▶│ Processes Core  │
│ Focuses on Core │◀───│ Core Request:   │    │ Request         │
│ Request         │    │ "Write AI report"│   └─────────────────┘
│                 │    └─────────────────┘
│ "Write AI report│
│ using tools"    │
└─────────────────┘
                                                                  │
                                                                  ▼
┌─────────────────┐      ┌─────────────────┐      ┌─────────────────┐
│ Remote Agent    │      │ ✓ SUCCESS:      │─────►│ Returns Useful  │
│ Response        │◄─────│ "Here's your AI │      │ Content         │
│                 │      │ report with     │      └─────────────────┘
│ "Here's your AI │      │ analysis..."    │
│ report..."      │      └─────────────────┘
└─────────────────┘
```

Implementation for All Remote Agents

Apply this pattern to **all remote agents** (research_agent, analysis_agent, content_agent):

1. Add **"IMPORTANT - A2A Context Handling"** section to instructions
2. Teach agents to ignore orchestrator tool calls in context
3. Focus agents on extracting and fulfilling core user requests
4. Restart A2A servers with updated instructions

5. Authentication in Official ADK A2A

Authentication Configuration

Authentication in ADK A2A is handled automatically by the `RemoteA2aAgent` class. For local development, authentication is typically optional:

```python
from google.adk.agents.remote_a2a_agent import RemoteA2aAgent

# ADK automatically handles authentication based on agent card
research_agent = RemoteA2aAgent(
    name="research_specialist",
    description="Conducts web research and fact-checking",
    agent_card="http://localhost:8001/a2a/research_specialist/.well-known/agen
)

# ADK manages:
# - Agent card fetching
# - Authentication negotiation
# - Token management (if required)
# - Error handling for auth failures
```

Agent Card Authentication

Local agents using `adk api_server --a2a` expose agent cards with authentication configuration:

```json
{
  "capabilities": {},
  "defaultInputModes": ["text/plain"],
  "defaultOutputModes": ["application/json"],
  "description": "Conducts web research and fact-checking",
  "name": "research_specialist",
  "url": "http://localhost:8001/a2a/research_specialist",
  "version": "1.0.0",
  "authentication": {
    "type": "none",
    "required": false
  }
}
```

Production Authentication

For production deployments, update agent configuration for authentication:

```json
{
  "name": "secure_research_agent",
  "description": "Secure research agent with authentication",
  "url": "https://research.example.com/a2a/research_agent",
  "authentication": {
    "type": "bearer",
    "required": true,
    "realm": "research-api"
  }
}
```

6. Advanced ADK A2A Patterns

Pattern 1: Error Handling and Retry

ADK provides built-in error handling for `RemoteA2aAgent`:

```python
from google.adk.agents.remote_a2a_agent import RemoteA2aAgent
from google.adk.agents import Agent
from google.adk.tools import FunctionTool

# Tool to check remote agent health
def validate_agent_health(agent_name: str, agent_url: str) -> dict:
    """Validate if remote agent is healthy before delegation."""
    try:
        import requests
        response = requests.get(f"{agent_url}/.well-known/agent-card.json", ti

        if response.status_code == 200:
            return {
                "status": "success",
                "healthy": True,
                "report": f"Agent {agent_name} is healthy"
            }
        else:
            return {
                "status": "error",
                "healthy": False,
                "report": f"Agent {agent_name} health check failed"
            }
    except Exception as e:
        return {
            "status": "error",
            "healthy": False,
            "report": f"Cannot reach agent {agent_name}: {str(e)}"
        }

# Robust orchestrator with health checking
robust_research_agent = RemoteA2aAgent(
    name="research_specialist",
    description="Research agent with automatic error handling",
    agent_card="http://localhost:8001/a2a/research_specialist/.well-known/agen
)

orchestrator_with_health_checks = Agent(
    model="gemini-2.0-flash",
    name="robust_orchestrator",
    instruction="""
Before delegating to any remote agent, use validate_agent_health
to ensure the agent is available. If an agent is unhealthy,
inform the user and suggest alternatives.
    """,
    sub_agents=[robust_research_agent],
    tools=[FunctionTool(validate_agent_health)]
)
```

Pattern 2: Parallel A2A Execution

Use ADK's `ParallelAgent` for concurrent remote agent execution:

```
from google.adk.agents import ParallelAgent
from google.adk.agents.remote_a2a_agent import RemoteA2aAgent

# Multiple remote agents
research_agent = RemoteA2aAgent(
    name="research_specialist",
    description="Conducts research",
    agent_card="http://localhost:8001/a2a/research_specialist/.well-known/agen
)

analysis_agent = RemoteA2aAgent(
    name="data_analyst",
    description="Analyzes data",
    agent_card="http://localhost:8002/a2a/data_analyst/.well-known/agent-card.
)

# Parallel execution of remote agents
parallel_processor = ParallelAgent(
    name="parallel_a2a_processor",
    description="Process tasks in parallel across remote agents",
    sub_agents=[research_agent, analysis_agent]
)

# Use in main orchestrator
main_orchestrator = Agent(
    model="gemini-2.0-flash",
    name="main_orchestrator",
    instruction="""
When users request both research and analysis, delegate to
parallel_a2a_processor to execute both tasks simultaneously.
    """,
    sub_agents=[parallel_processor]
)
```

Pattern 3: Agent Health Monitoring

Monitor multiple A2A agents with centralized health checking:

```
def monitor_all_a2a_agents() -> dict:
    """Monitor health of all A2A agents in the system."""
    agents_to_check = [
        ("research_specialist", "http://localhost:8001/a2a/research_specialist
        ("data_analyst", "http://localhost:8002/a2a/data_analyst"),
        ("content_writer", "http://localhost:8003/a2a/content_writer")
    ]

    results = {}
    overall_healthy = True

    for agent_name, agent_url in agents_to_check:
        health_result = validate_agent_health(agent_name, agent_url)
        results[agent_name] = health_result

        if not health_result.get("healthy", False):
            overall_healthy = False

    return {
        "status": "success" if overall_healthy else "error",
        "overall_healthy": overall_healthy,
        "individual_results": results,
        "report": f"System health: {'All healthy' if overall_healthy else 'Som
    }

# Health monitoring orchestrator
health_monitor = Agent(
    model="gemini-2.0-flash",
    name="health_monitor",
    instruction="""
Use monitor_all_a2a_agents to check the health of all remote agents
before performing complex orchestration tasks. Report any issues to users.
    """,
    tools=[FunctionTool(monitor_all_a2a_agents)]
)
```

7. Understanding the Official ADK A2A Implementation

Project Structure

The official ADK A2A implementation follows this structure:

```
tutorial17/
├── a2a_orchestrator/          # Main orchestrator using RemoteA2aAgent
│   ├── __init__.py           # Package initialization
│   ├── agent.py              # Orchestrator with RemoteA2aAgent instances
│   └── .env.example          # Environment template
├── research_agent/           # Remote Research Agent
│   ├── __init__.py
│   ├── agent.py             # Research agent implementation
│   └── agent-card.json      # Agent card for A2A discovery
├── analysis_agent/          # Remote Analysis Agent
│   ├── __init__.py
│   ├── agent.py             # Analysis agent implementation
│   └── agent-card.json      # Agent card for A2A discovery
├── content_agent/           # Remote Content Agent
│   ├── __init__.py
│   ├── agent.py             # Content agent implementation
│   └── agent-card.json      # Agent card for A2A discovery
├── start_a2a_servers.sh     # Script to start all A2A servers
├── stop_a2a_servers.sh      # Script to stop all A2A servers
└── tests/                   # Test suite
```

A2A Project Architecture:

```
tutorial17/
├── 📁 a2a_orchestrator/           # 🎯 MAIN COORDINATOR
│   ├── __init__.py               # Package setup
│   ├── agent.py                  # RemoteA2aAgent instances
│   │   ├── research_agent        # → http://localhost:8001
│   │   ├── analysis_agent        # → http://localhost:8002
│   │   └── content_agent         # → http://localhost:8003
│   └── .env.example              # GOOGLE_API_KEY template
│
├── 📁 research_agent/             # 🔬 SPECIALIZED AGENT
│   ├── __init__.py               # Package setup
│   ├── agent.py                  # Root agent + a2a_app export
│   │   ├── root_agent            # Agent with research tools
│   │   └── a2a_app = to_a2a()    # A2A server app
│   └── agent-card.json           # Auto-generated by to_a2a()
│
├── 📁 analysis_agent/             # 📊 SPECIALIZED AGENT
│   ├── __init__.py               # Package setup
│   ├── agent.py                  # Root agent + a2a_app export
│   │   ├── root_agent            # Agent with analysis tools
│   │   └── a2a_app = to_a2a()    # A2A server app
│   └── agent-card.json           # Auto-generated by to_a2a()
│
├── 📁 content_agent/              # ✍️ SPECIALIZED AGENT
│   ├── __init__.py               # Package setup
│   ├── agent.py                  # Root agent + a2a_app export
│   │   ├── root_agent            # Agent with content tools
│   │   └── a2a_app = to_a2a()    # A2A server app
│   └── agent-card.json           # Auto-generated by to_a2a()
│
├── 🛠️ start_a2a_servers.sh        # Server management script
│   ├── uvicorn research_agent.agent:a2a_app --port 8001
│   ├── uvicorn analysis_agent.agent:a2a_app --port 8002
│   └── uvicorn content_agent.agent:a2a_app --port 8003
│
├── 🛑 stop_a2a_servers.sh         # Clean shutdown script
└── 🧪 tests/                      # Test suite
    ├── test_a2a_integration.py   # End-to-end A2A tests
    └── test_agent_structure.py   # Agent configuration tests
```

8. Best Practices for Working ADK A2A

✔️ DO: Use to_a2a() Function for Agent Exposure

```
# ✔️ Good - Use working to_a2a() pattern
from google.adk.a2a.utils.agent_to_a2a import to_a2a

# Create A2A application using the working ADK to_a2a() function
a2a_app = to_a2a(root_agent, port=8001)

# Start with: uvicorn research_agent.agent:a2a_app --host localhost --port 800

# ❌ Bad - Experimental adk api_server approach
# adk api_server --a2a --port 8001 research_agent/
```

✔️ DO: Use uvicorn for A2A Server Hosting

```
# ✔️ Good - Working uvicorn + to_a2a() pattern
uvicorn research_agent.agent:a2a_app --host localhost --port 8001

# ❌ Bad - Experimental adk command
# adk api_server --a2a --port 8001 research_agent/
```

✔️ DO: Use Sub-Agents Pattern

```
# ✔️ Good - Use RemoteA2aAgent as sub-agent
orchestrator = Agent(
    model="gemini-2.0-flash",
    name="orchestrator",
    instruction="Delegate tasks to specialized sub-agents...",
    sub_agents=[research_agent, analysis_agent]  # Clean delegation
)

# ❌ Bad - Manual tool functions for A2A
# Don't create tool functions that manually handle A2A communication
```

✔️ DO: Use Proper Agent Card URLs

```
# ✔️ Good - Use AGENT_CARD_WELL_KNOWN_PATH constant
from google.adk.agents.remote_a2a_agent import AGENT_CARD_WELL_KNOWN_PATH

agent_card_url = f"http://localhost:8001/a2a/research_specialist{AGENT_CARD_WE

# ❌ Bad - Hardcode path or wrong path
agent_card_url = "http://localhost:8001/.well-known/agent.json"  # Wrong!
```

✔️ DO: Use Automated Server Management

```
# ✔️ Good - Use the provided scripts with health checks
./start_a2a_servers.sh   # Starts all servers with verification
./stop_a2a_servers.sh    # Clean shutdown

# ❌ Bad - Manual server management without health checks
# uvicorn ... & (without verification or proper cleanup)
```

9. Troubleshooting Working ADK A2A

Error: "Agent card not found"

Problem: Remote agent doesn't expose agent card or A2A server not running

Solutions:

1. **Check if uvicorn servers are running**:

```
# Check if agent card endpoints are accessible
curl http://localhost:8001/.well-known/agent-card.json
curl http://localhost:8002/.well-known/agent-card.json
curl http://localhost:8003/.well-known/agent-card.json
```

1. **Restart A2A servers using working scripts**:

```
# Stop existing servers cleanly
./stop_a2a_servers.sh

# Start fresh servers with health checks
./start_a2a_servers.sh
```

Error: "Connection timeout" or "Connection refused"

Problem: Network issues or uvicorn server ports not available

Solutions:

1. **Check port conflicts**:

```
# See what's using the A2A ports
lsof -i :8001
lsof -i :8002
lsof -i :8003
```

1. **Clean restart with port cleanup**:

```
# Kill processes on A2A ports (working pattern)
pkill -f "uvicorn.*research_agent\|uvicorn.*analysis_agent\|uvicorn.*content_a

# Start servers using working scripts
./start_a2a_servers.sh
```

Issue: "RemoteA2aAgent not responding"

Problem: A2A communication or agent processing issues

Solutions:

1. **Test direct A2A endpoint**:

```
# Test agent card retrieval
curl -v http://localhost:8001/.well-known/agent-card.json

# Check uvicorn server logs for errors
uvicorn research_agent.agent:a2a_app --host localhost --port 8001 --log-level
```

1. **Verify agent implementation uses to_a2a()**:

```python
# Check that remote agent has proper a2a_app export
# In research_agent/agent.py:
from google.adk.a2a.utils.agent_to_a2a import to_a2a

root_agent = Agent(
    model="gemini-2.0-flash",
    name="research_specialist",
    # ... agent configuration
)

# Critical: Export a2a_app using to_a2a()
a2a_app = to_a2a(root_agent, port=8001)
```

Lesson Learned: adk api_server --a2a vs uvicorn + to_a2a()

Common Mistake: Using `adk api_server --a2a` (experimental/incorrect)
Working Solution: Using `uvicorn + to_a2a()` (tested/working)

```
# ❌ This doesn't work reliably:
# adk api_server --a2a --port 8001 research_agent/

# ✔️ This works (tested implementation):
uvicorn research_agent.agent:a2a_app --host localhost --port 8001
```

Development Tips for Working Implementation

- **Use** `./start_a2a_servers.sh` for consistent server setup with health checks
- **Check agent card format** at `/.well-known/agent-card.json` endpoints
- **Use** `uvicorn + to_a2a()` instead of experimental adk commands

- **Verify** `a2a_app` **export** in each remote agent module using `to_a2a()`
- **Test with** `--log-level debug` for detailed troubleshooting
- **Use provided scripts** instead of manual server management

Key Implementation Lessons Learned

During the development and testing of this A2A implementation, several critical lessons emerged that are essential for successful A2A deployment:

🎯 Lesson 1: Use to_a2a() Function, Not adk api_server

Discovery: The `adk api_server --a2a` command is experimental and unreliable.
Solution: Use the `to_a2a()` function with uvicorn for stable A2A servers.

```
# ✔️ Working pattern (tested and verified)
from google.adk.a2a.utils.agent_to_a2a import to_a2a
a2a_app = to_a2a(root_agent, port=8001)

# Start with: uvicorn research_agent.agent:a2a_app --host localhost --port 800

# ❌ Problematic pattern
# adk api_server --a2a --port 8001 research_agent/
```

🎯 Lesson 2: Auto-Generated Agent Cards are Key

Discovery: Agent cards are automatically generated by `to_a2a()` - no manual creation needed.
Benefit: Eliminates agent card sync issues and reduces configuration errors.

```
# These are created automatically when using to_a2a():
# http://localhost:8001/.well-known/agent-card.json
# http://localhost:8002/.well-known/agent-card.json
# http://localhost:8003/.well-known/agent-card.json
```

🎯 Lesson 3: Health Checks Are Essential

Discovery: A2A servers need proper health checking and process management.
Solution: Use scripts with server verification and clean shutdown.

```
# Working pattern with health checks
./start_a2a_servers.sh    # Includes health verification
./stop_a2a_servers.sh     # Clean process termination
```

🎯 Lesson 4: Agent Card URL Construction

Discovery: Precise agent card URL construction is critical for discovery.
Pattern: Use `AGENT_CARD_WELL_KNOWN_PATH` constant for consistency.

```
from google.adk.agents.remote_a2a_agent import AGENT_CARD_WELL_KNOWN_PATH

# ✔️ Correct pattern
agent_card = f"http://localhost:8001/a2a/research_specialist{AGENT_CARD_WELL_K

# ❌ Common mistakes
# "http://localhost:8001/.well-known/agent.json"  # Wrong filename
# "http://localhost:8001/agent-card.json"         # Missing path
```

🎯 Lesson 5: Sub-Agent Pattern Simplifies Architecture

Discovery: Using RemoteA2aAgent as sub-agents creates clean, maintainable code.
Benefit: Orchestration becomes simple delegation without manual protocol handling.

```
# ✔️ Clean sub-agent pattern
root_agent = Agent(
    name="a2a_orchestrator",
    instruction="Delegate to specialized sub-agents...",
    sub_agents=[research_agent, analysis_agent, content_agent]
)
```

🎯 Lesson 6: Process Management Matters

Discovery: Proper process cleanup prevents port conflicts and resource leaks.
Solution: Use targeted process killing and health verification.

```
# Working cleanup pattern
pkill -f "uvicorn.*research_agent\|uvicorn.*analysis_agent\|uvicorn.*content_a
```

🎯 Lesson 7: Proper A2A Context Handling is Critical

Discovery: Remote agents were misinterpreting orchestrator context and responding with
"I cannot use transfer_to_agent tool" instead of processing the actual user request.
Solution: Update remote agent instructions to focus on the core user request and ignore
orchestrator mechanics in A2A contexts.

```
# ✔️ Working A2A context handling pattern
instruction="""
**IMPORTANT - A2A Context Handling:**
When receiving requests via Agent-to-Agent (A2A) protocol, focus on the core u
Ignore any mentions of orchestrator tool calls like "transfer_to_agent" in the
Extract the main task from the conversation and complete it directly.

**When working via A2A:**
- Focus on the actual request from the user (e.g., "Write a report about AI")
- Ignore orchestrator mechanics and tool calls in the context
- Provide direct, helpful services using your tools
- If the request is unclear, ask for clarification about the task
"""
```

Impact: This fix transformed A2A communication from broken responses to meaningful,
intelligent agent interactions that properly utilize tools and provide valuable content.

Summary

You've mastered **working ADK Agent-to-Agent communication** through a tested implementation:

Key Takeaways:

- ✔️ `to_a2a()` function enables stable A2A servers with uvicorn
- ✔️ `RemoteA2aAgent` creates distributed agent systems with ADK
- ✔️ Auto-generated agent cards at `.well-known/agent-card.json`
- ✔️ Sub-agent pattern for clean delegation to remote agents
- ✔️ Health monitoring with proper server management scripts
- ✔️ Proper agent card URL construction with constants
- ✔️ Working process management and cleanup patterns
- ✔️ Proper A2A context handling for intelligent remote agent responses

Production Checklist:

- [ ] Remote agents use `a2a_app = to_a2a(root_agent, port=XXXX)`
- [ ] A2A servers deployed with `uvicorn agent.agent:a2a_app`
- [ ] `RemoteA2aAgent` instances configured with correct agent_card URLs
- [ ] Health monitoring scripts implemented (start/stop_a2a_servers.sh)
- [ ] Agent card URLs use `AGENT_CARD_WELL_KNOWN_PATH` constant
- [ ] Process cleanup handles uvicorn processes correctly
- [ ] All remote agents export proper `a2a_app` using `to_a2a()`
- [ ] Remote agents have proper A2A context handling instructions

Working Implementation Verified:

This tutorial reflects a real, tested A2A implementation with:
- ✔️ All servers starting successfully with health checks
- ✔️ Auto-generated agent cards accessible
- ✔️ Clean orchestration via sub-agent pattern
- ✔️ Proper process management and cleanup
- ✔️ 24 passing tests verifying functionality

Next Steps:

- **Tutorial 18**: Learn Events & Observability

- **Tutorial 19**: Implement Artifacts & File Management
- **Tutorial 20**: Master YAML Configuration

Resources:

- Official ADK A2A Documentation (https://google.github.io/adk-docs/a2a/)
- ADK RemoteA2aAgent API Reference (https://google.github.io/adk-docs/api-reference/)
- A2A Protocol Official Website (https://a2a-protocol.org/)

🎉 **Tutorial 17 Complete!** You now know how to build distributed multi-agent systems using the **official ADK A2A protocol**. Continue to Tutorial 18 to learn about events and observability.

Process Management

The working implementation includes tested scripts for reliable A2A server management:

```
# start_a2a_servers.sh - Start all A2A servers
#!/bin/bash

echo "🚀 Starting ADK A2A servers using to_a2a() function..."

# Clean up any existing processes
pkill -f "uvicorn.*research_agent\|uvicorn.*analysis_agent\|uvicorn.*content_a

# Start research agent using uvicorn + to_a2a()
echo "🔬 Starting Research Agent on port 8001..."
uvicorn research_agent.agent:a2a_app --host localhost --port 8001 &
RESEARCH_PID=$!

# Start analysis agent using uvicorn + to_a2a()
echo "📊 Starting Analysis Agent on port 8002..."
uvicorn analysis_agent.agent:a2a_app --host localhost --port 8002 &
ANALYSIS_PID=$!

# Start content agent using uvicorn + to_a2a()
echo "✍️  Starting Content Agent on port 8003..."
uvicorn content_agent.agent:a2a_app --host localhost --port 8003 &
CONTENT_PID=$!

# Wait for servers to start and verify they're running
echo "🔄 Waiting for all agents to be ready..."

# Function to check server health
wait_for_server() {
    local port=$1
    local agent_name=$2
    local max_attempts=30
    local attempt=1

    echo "⏳ Waiting for $agent_name to be ready on port $port..."

    while [ $attempt -le $max_attempts ]; do
        if curl -s "http://localhost:$port/.well-known/agent-card.json" >/dev/
            echo "✔️ $agent_name is ready on port $port"
            return 0
        fi
        sleep 1
        attempt=$((attempt + 1))
    done

    echo "❌ $agent_name failed to start on port $port"
    return 1
}

# Verify all servers started successfully
if wait_for_server 8001 "Research Agent" && \
   wait_for_server 8002 "Analysis Agent" && \
   wait_for_server 8003 "Content Agent"; then

    echo "🎉 All A2A servers are running successfully!"
    echo "🔗 Agent Cards (auto-generated by to_a2a()):"
    echo "   • Research: http://localhost:8001/.well-known/agent-card.json"
    echo "   • Analysis: http://localhost:8002/.well-known/agent-card.json"
    echo "   • Content:  http://localhost:8003/.well-known/agent-card.json"
else
    echo "❌ Some servers failed to start. Check the logs for errors."
    exit 1
fi
```

A2A Server Startup Process:

```
┌─────────────────┐     ┌─────────────────┐     ┌─────────────────┐
│ ./start_a2a_    │     │ 1. Clean Up     │────▶│ Kill existing   │
│ servers.sh      │────▶│    Processes    │     │ uvicorn processes│
└─────────────────┘     └─────────────────┘     └─────────────────┘
        │
        ▼
┌─────────────────┐     ┌─────────────────┐     ┌─────────────────┐
│ 2. Start        │────▶│ Research Agent  │────▶│ uvicorn research│
│    Servers      │     │ (Port 8001)     │     │ _agent.agent:    │
│                 │     └─────────────────┘     │ a2a_app --port   │
│                 │                             │ 8001 &           │
└─────────────────┘                             └─────────────────┘
        │
        ▼
┌─────────────────┐     ┌─────────────────┐     ┌─────────────────┐
│ 3. Start        │────▶│ Analysis Agent  │────▶│ uvicorn analysis│
│    Servers      │     │ (Port 8002)     │     │ _agent.agent:    │
│                 │     └─────────────────┘     │ a2a_app --port   │
│                 │                             │ 8002 &           │
└─────────────────┘                             └─────────────────┘
        │
        ▼
┌─────────────────┐     ┌─────────────────┐     ┌─────────────────┐
│ 4. Start        │────▶│ Content Agent   │────▶│ uvicorn content │
│    Servers      │     │ (Port 8003)     │     │ _agent.agent:    │
│                 │     └─────────────────┘     │ a2a_app --port   │
│                 │                             │ 8003 &           │
└─────────────────┘                             └─────────────────┘
        │
        ▼
┌─────────────────┐     ┌─────────────────┐     ┌─────────────────┐
│ 5. Health       │────▶│ Check Agent     │────▶│ curl /.well-    │
│    Checks       │     │ Cards Available │     │ known/agent-    │
│                 │     └─────────────────┘     │ card.json        │
└─────────────────┘                             └─────────────────┘
        │
        ▼
┌─────────────────┐     ┌─────────────────┐     ┌─────────────────┐
│ 6. Verification │────▶│ All Servers     │────▶│ ✔ SUCCESS: All  │
│                 │     │ Running & Ready │     │ servers running │
│                 │     └─────────────────┘     │ 🔗 Agent cards   │
└─────────────────┘                             │    accessible   │
                                                └─────────────────┘
```

Generated on 2025-10-19 17:56:53 from 17_agent_to_agent.md

Source: Google ADK Training Hub