WFO Agent2Agent Integration and Multi-Agent Troubleshooting Proposal

Name and Organisation: Peter Boers, SURF

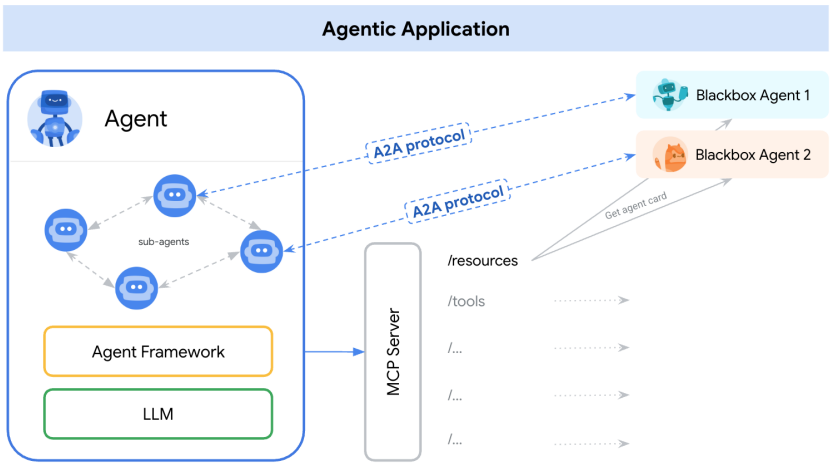
Title: Workflow Orchestrator (WFO) – A2A Protocol Integration, Agent Refactoring, and Multi-Agent Troubleshooting

Rationale:

The next stage in enhancing the Workflow Orchestrator (WFO) is to evolve the existing RAG agent into a modular, multi-agent system capable of interacting through the Agent-to-Agent (A2A) protocol. Multi-agent architectures allow autonomous components to collaborate, exchange capabilities, and coordinate actions. Compared to MCP, A2A allows a group of agents to work together and collaborate, whereas just using MCP you can only execute tool calling which is (mostly) stateless. This potentially places a heavy burden on a single agent that may have a complex troubleshooting workflow.

One of the outcomes of the previous incubator (WFO Rag Agent and LLM search) was that when you build an AI agent, you need to curate the context used whilst interacting with an LLM very carefully. Firstly, because of the limited size of context available and secondly, because the system prompt of the agent needs to be (very) well defined. Adjusting the system prompt might cause unexpected behavior and to extract the best performance from smaller LLM's, it needs to be very clear and instructive. Due to this fact, adding functionality (tool calling) was trial and error and quite cumbersome. Whilst evaluating several different LLM's we saw that only the most recent (expensive) models were able to handle various flows through a system prompt, as they are trained with (complex) tool calling in mind and could handle the complex tasks we asked of them. Going down this route forces lock-in, decreases reliability and slows the velocity in building new functionality, as you need to test extensively and are optimizing for the LLM you are developing with.

There is another way to approach this problem and that is to implement the A2A protocol by combining multiple specialized agents to perform an overarching task. Going down this route also unlocks the possibility of re-use and maps these smaller agents correctly on a micro-service architecture. It also allows the user to use LLM's that are specialized in certain fields for example, reasoning, network configuration etc to do what they are optimized for. This enables running smaller LLM's and limits the use of expensive models to perhaps the User to Agent interaction phase.



Source: a2a-protocol.org Agents interacting with tools and through the A2A protocol

This figure shows how a multi-agent architecture leverages MCP for (stateless) tool calling, and the A2A protocol for more complex tasks that need reasoning and analysis. A2A is about agents partnering on tasks, while MCP is more about agents using capabilities.

Considering a typical network troubleshooting use-case, what would the agentic architecture look like?

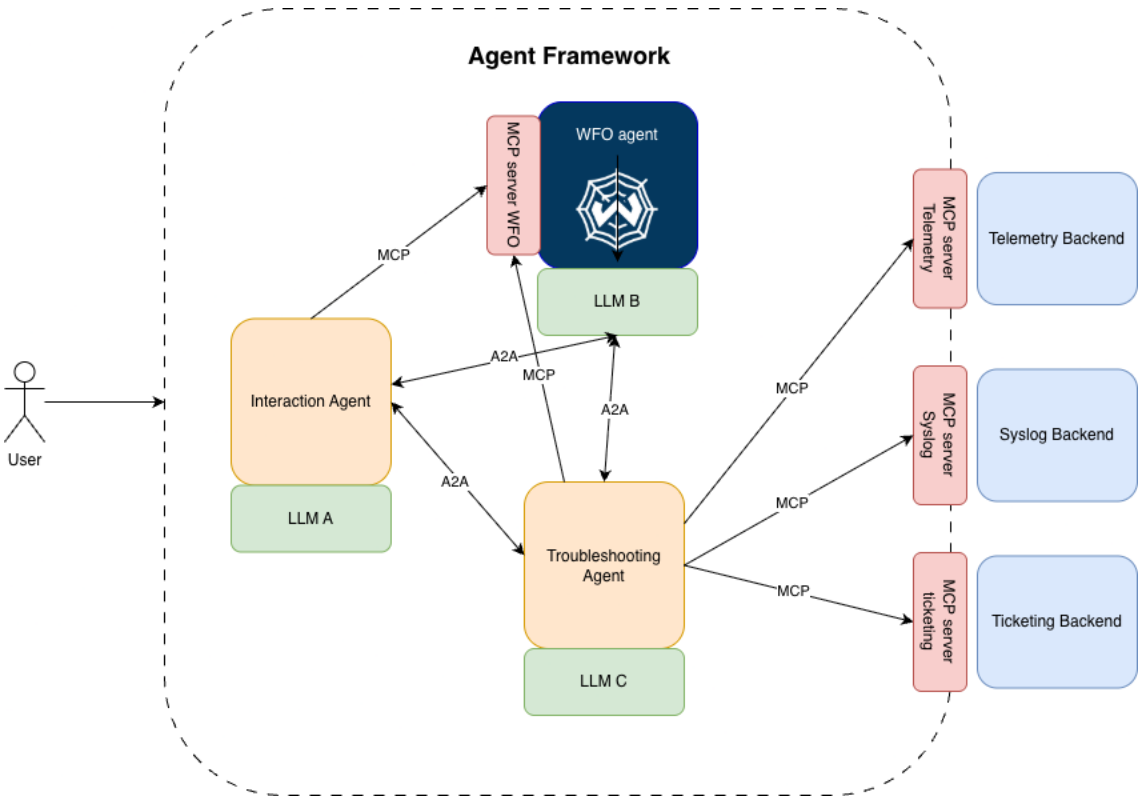
Scenario: A customer emails the NOC that their internet connection is experiencing a problem. The NOC has a number of specialized tools that enable troubleshooting and customer interaction. The following agents might be part of the interaction to troubleshoot a ticket.

- **NOC interaction:** In this User to the Troubleshooting Agent interaction, the NOC would begin the troubleshoot session with the following query: “Customer A has a problem with service Y, please investigate?
- **Diagnostic conversation (A2A):** The Troubleshooting agent will interact with the NOC engineer multiple time to get as much information as possible. It could ask: At what time did the incident start? Or; Are there any logs available from the customer?
- **Investigation agent (MCP):** The troubleshooting Agent would call out to the Analysis agent who could do multiple tool calls in systems like telemetry and syslog to glean more information.
- **Impact Analysis(A2A):** The results of the investigation agent could be passed to the Impact Analysis agent. It would make use of the A2A interface of the Orchestrator to search the relevant subscriptions to accurately define the impact.
- **Returning results (A2A):** The impact could then be returned to the Troubleshooting Agent through A2A and presented to the NOC user. This could then be the starting point of a follow up interaction that could help fix the issue.

The above describes an advanced scenario. In this phase of the incubator we propose a more limited scope to lay the foundations of such a multi-agent architecture, by using at most 3 agents.

In this phase, we propose:

1. Integrating and evaluating the A2A protocol inside the orchestrator, enabling the orchestrator agent to offer skills such as search, retrieval, and aggregation to other agents.
2. Refactoring the existing RAG agent into a standalone module for reuse and future extension.
3. Showcasing a multi-agent workflow involving an interaction agent, the orchestrator agent as a coordinating agent and a troubleshooting agent. This should demonstrate iterative reasoning across telemetry and logs until a conclusion is reached.



Key Components to be investigated

- A2A Protocol
- MCP Protocol
- Orchestrator Agent refactor to enable A2A interaction standalone operations
- Troubleshooting Agent capable of doing tool calls
- Client Agent that is responsible for user interaction.

- Frameworks that enable the running of agents and MCP-servers like Kagent.

Scope of work

The work we envision during this incubator includes the following:

- Refactoring the RAG agent Implementation in the Orchestrator to a standalone module
- Adapting and changing the current Orchestrator UI with new components to display results of the interactions.
- Defining a blueprint architecture and developing/curating a set of troubleshooting tools. Either by selecting opensource MCP-servers and/or developing a library of tools.
- Making the current tools in the orchestrator available through MCP
- Implementing Auth(N|Z) on the MCP servers
- Implementing the A2A protocol in the orchestrator
- Evaluating an Agent framework (Kagent) that enables declarative running of Agents in Kubernetes

Output (Deliverables):

1. A2A integration in the example orchestrator.
2. Refactored standalone agent module.
3. Multi-agent troubleshooting demonstrator.
4. Integration with Pydantic-AI, A2A (ADK), Kagent, MCP servers.
5. Documentation for setup and extension.
6. Open-source code on GitHub.
7. A GEANT infoshare demonstration.

Users: Any WFO orchestrator user and NREN's investigating Agentic architectures

Value: This proposal moves WFO toward autonomous operations, reducing integration effort and enabling automated reasoning and troubleshooting.

Team: Peter Boers – Architect, SURF**,** Rene Dohmen – Software Engineer, SURF

Timeline: 1 Jan 2026 – 31 December 2026

Resources: Funding for 1 FTE to be hired by SURF.

Intellectual Property Rights: No

Gathering and Processing of Personal Data: No

Existing Project: Extension of WP7 led by Simone Spinelli.

Organizations Supporting:

- Simone Spinelli, GEANT

- Matteo Colantonio, GARR