Implementing an **LLM-based network topology traversal** and **event correlation agent**

waylay

INTRODUCTION

Autonomous networks are characterized by their capability to self-provision, scale, heal and maintain themselves in response to external or internal factors. These self-X capabilities are mainly triggered by various events ranging from service provisioning requests to equipment or service alarms, under the larger umbrella of event-driven automation.

The ability to correlate such events across the temporal and spatial domains enables the autonomous network and the NOC engineers to identify any changes in network state and determine the appropriate response to them. Temporal correlation identifies patterns in the frequency and number of event occurrences, while spatial correlation links events at different physical and logical levels of the network topology, driving an intelligent and effective response to the observed conditions.

In this brochure we will leverage the reasoning capabilities of large language models (LLMs), combined with function calling and prompt engineering, to build an intelligent event correlation agent on top of a radio access network (RAN) topology using the Waylay platform.

THE NETWORK ENTITY MODEL

We begin by modeling the RAN topology which will act as the knowledge graph for our agent, supporting the event correlation operations. The topology is based on a set of entities and the relationships (links) defined between them. There are multiple ways to model a radio access network, based on the used technology or vendor-specific characteristics, a relevant example being the **3GPP NR NRM model**.

For our exercise we will adopt a more streamlined representation, shown in Figure 1. Our model is centered on the BaseStation entity, which belongs to a Site, is managed via an EMS and experiences Events, some of which may trigger associated RemediationActions. For example, a power outage event experienced by a base station may trigger the dispatch of an intervention unit to the site.

For more advanced use cases, network functions like the DU and CU can be modeled, together with the UE or NetworkSlice details.

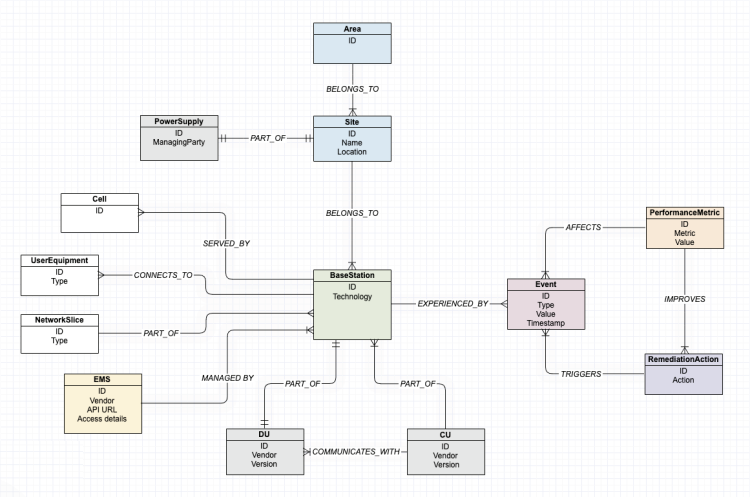


Figure 1. The RAN entity model

NETWORK TOPOLOGY AS A KNOWLEDGE GRAPH

The next step in our project is to populate the RAN topology based on the chosen entity model. We want this topology to act as a knowledge graph for the LLM, enabling it to traverse it and perform spatial event correlations taking into account the relationships between different RAN entities.

During the **GenAI-based toolkit for network & service management** TM Forum Catalyst project, we found Neo4J to be the most suitable way to store graph-based network topology information like the one above. As a bonus, LLMs like OpenAI's GPT-4 or Anthropic Claude 3 possess the intrinsic capability to generate Neo4J Cypher queries, an ability that stems from their extensive training data.

Based on environmental alarm data collected from a lab setup, we populated a Neo4J instance with the RAN topology information, as shown in Figure 2.

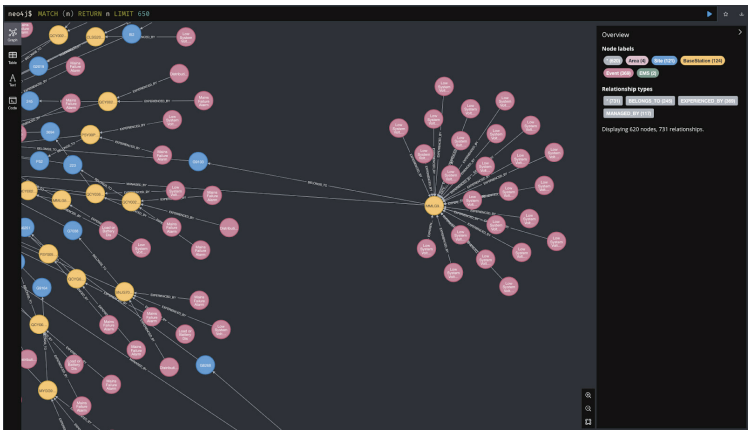


Figure 2. RAN topology detail, showing a base station and its associated alarm events

This network topology plays a double role. Next to storing the network representation and associated events (and ideally keeping it up to date by consuming event streams from sources like the OSS, NMS or EMSes), it also acts as a knowledge graph towards the LLM.

By making a parallel to the concept of augmented retrieval (RAG), where a vector database acts as a knowledge base for the answers provided by the LLM, the Neo4J database containing the network topology becomes a knowledge graph, providing the LLM with the data required to answer queries related to the network state. For example, is site X currently affected by both temperature and power supply alarms? or which are the top 5 sites affected by low power alarms and in which areas?

Unlike RAG, the knowledge graph was not yet supported natively by the latest generation of LLM models at the moment this article was written. However, it can be exposed to the models via other techniques like function calling.

EXPOSING NETWORK TOPOLOGY SEMANTICS VIA ONTOLOGY

As mentioned before, our experience has shown that models like OpenAI's GPT-4 or Anthropic Claude 3 possess the capability to generate Neo4J Cypher queries. If we configure them with a tool (function) that allows them to execute a query against a Neo4J database, they will be able to generate a syntactically correct cypher query from the user prompt and execute it via the provided tool.

However, a syntactically correct query will not be sufficient. As the LLM does not have the innate capability to understand the database model, it cannot generate a semantically correct query, which is what we require to leverage the knowledge stored in the network topology.

To enable the model to understand the knowledge graph, we must provide it with an ontology, which describes the entity model on which the knowledge graph is based. We can regard the ontology as a formal, textual description of Figure 1, which is provided upfront to the LLM either via the system prompt (message) or via other techniques like fine-tuning.

In our case, we provide the RAN ontology via the LLM's system message and encode it in the JSON format for a formal, unambiguous and structured representation. Below is an example of the ontology related to a subset of the RAN model, related to the Area, Site, BaseStation, EMS and Event entities and the relations between them.

As mentioned before, our experience has shown that models like OpenAI's GPT-4 or Anthropic Claude 3 possess the capability to generate Neo4J Cypher queries. If we configure them with a tool (function) that allows them to execute a query against a Neo4J database, they will be able to generate a syntactically correct cypher query from the user prompt and execute it via the provided tool.

However, a syntactically correct query will not be sufficient. As the LLM does not have the innate capability to understand the database model, it cannot generate a semantically correct query, which is what we require to leverage the knowledge stored in the network topology.

```
{
  "relationships": [
    {
      "description": "A Site belongs to an Area.",
      "from": "Site",
      "to": "Area",
      "type": "BELONGS_TO"
    },
    {
      "description": "A BaseStation belongs to a Site.",
      "from": "BaseStation",
      "to": "Site",
      "type": "BELONGS_TO"
    },
    {
      "description": "An Event is experienced by a BaseStation.",
      "from": "Event",
      "to": "BaseStation",
      "type": "EXPERIENCED_BY"
    },
    {
      "description": "A a BaseStation is managed by an EMS.",
      "from": "BaseStation",
      "to": "EMS",
      "type": "MANAGED_BY"
    }
  ],
  "entities": {
    "EMS": {
      "properties": {
        "vendor": {
          "description": "the name of the EMS vendor"
        },
        "id": {
          "description": "the unique EMS identifier, with no semantic details"
        }
      }
    },
    "Area": {
      "properties": {
        "name": {
          "description": "friendly name"
        },
        "id": {
          "description": "unique identifier, with no semantic details"
        }
      }
    },
    "Event": {
      "properties": {
        "description": {
          "description": "full event description"
        },
        "id": {
          "description": "unique identifier, with no semantic details"
        },
        "status": {
          "description": "event status",
          "supportedValues": [
            "Terminated",
            "Active"
          ]
        },
        "severity": {
          "description": "event severity",
          "supportedValues": [
            "Critical",
            "Major",
            "Minor"
          ]
        },
        "type": {
          "description": "the event type",
          "supportedValues": [
            "alarm"
          ]
        },
        "clearedOn": {
          "description": "the event clearance timestamp, in ISO date format (may be missing for events in Active state)"
        },
        "name": {
          "description": "name of the event, also indicating its nature"
        },
        "createdOn": {
          "description": "the event creation timestamp, in ISO date format"
        }
      }
    },
    "Site": {
      "properties": {
        "name": {
          "description": "friendly name"
        },
        "id": {
          "description": "unique identifier, with no semantic details"
        }
      }
    },
    "BaseStation": {
      "properties": {
        "vendor": {
          "description": "the name of the base station vendor"
        },
        "name": {
          "description": "friendly name"
        },
        "id": {
          "description": "unique identifier, with no semantic details"
        },
        "technology": {
          "description": "the base station technology",
          "supportedValues": [
            "2G",
            "3G",
            "4G",
            "LTE",
            "5G"
          ]
        }
      }
    }
  }
}
```

To enable the model to understand the knowledge graph, we must provide it with an ontology, which describes the entity model on which the knowledge graph is based. We can regard the ontology as a formal, textual description of Figure 1, which is provided upfront to the LLM either via the system prompt (message) or via other techniques like fine-tuning.

In our case, we provide the RAN ontology via the LLM's system message and encode it in the JSON format for a formal, unambiguous and structured representation. Below is an example of the ontology related to a subset of the RAN model, related to the Area, Site, BaseStation, EMS and Event entities and the relations between them.

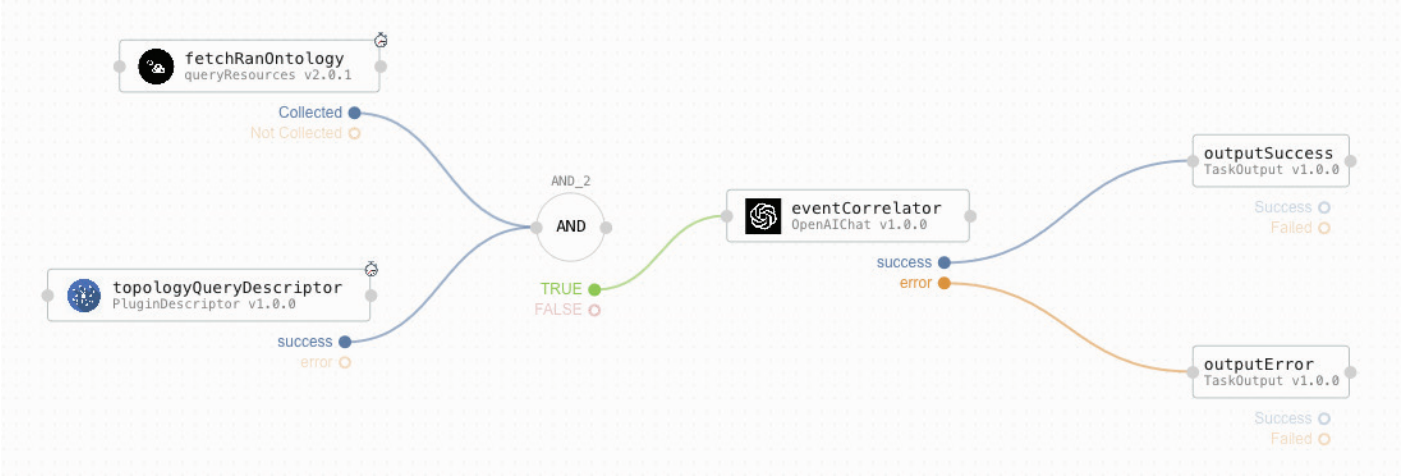
This enables the LLM to understand the structure of the knowledge graph and translate user prompts into cypher queries that are semantically correct and can execute successfully on the Neo4J database.

BUILDING AND EXPOSING THE EVENT CORRELATION AGENT

So far we have prepared the RAN model, topology data and the ontology for our LLM. It is now the time to actually build the agent using the Waylay platform.

The agent executes as a workflow (or **rule template** in Waylay terminology), and is exposed via the platform APIs to a chat-like user interface. The workflow, illustrated in Figure 3, contains the following nodes:

- **fetchRanOntology** - retrieves the RAN ontology JSON representation from the Waylay resources model;
- **topologyQueryDescriptor** - wraps the neo4j Waylay **plugin** as a tool descriptor that can be passed as part of the LLM configuration, enabling the LLM to execute neo4j queries using function calling;
- **eventCorrelator** - interacts with the LLM API to pass the user prompt and receive the agent's answer;
- **outputSuccess, outputError** - handle the LLM invocation result and expose it as the workflow execution result towards the chat user interface.



During testing we found out that the way the system message is crafted has a significant impact on the behavior and accuracy of the agent, as follows on the next page.

- First, we observed that providing graph traversal examples next to the ontology description via prompt engineering significantly increases the accuracy and correctness of the generated cypher queries.
- Second, we discovered that by default, the search for event types (e.g. temperature or power alarms) was done by the LLM in a very strict way, via case sensitive string comparison. Providing fuzzy search capabilities inside event names and descriptions via Neo4J full-text search indexes, and making the LLM aware of it via the system prompt, has made this search more flexible and user-friendly.

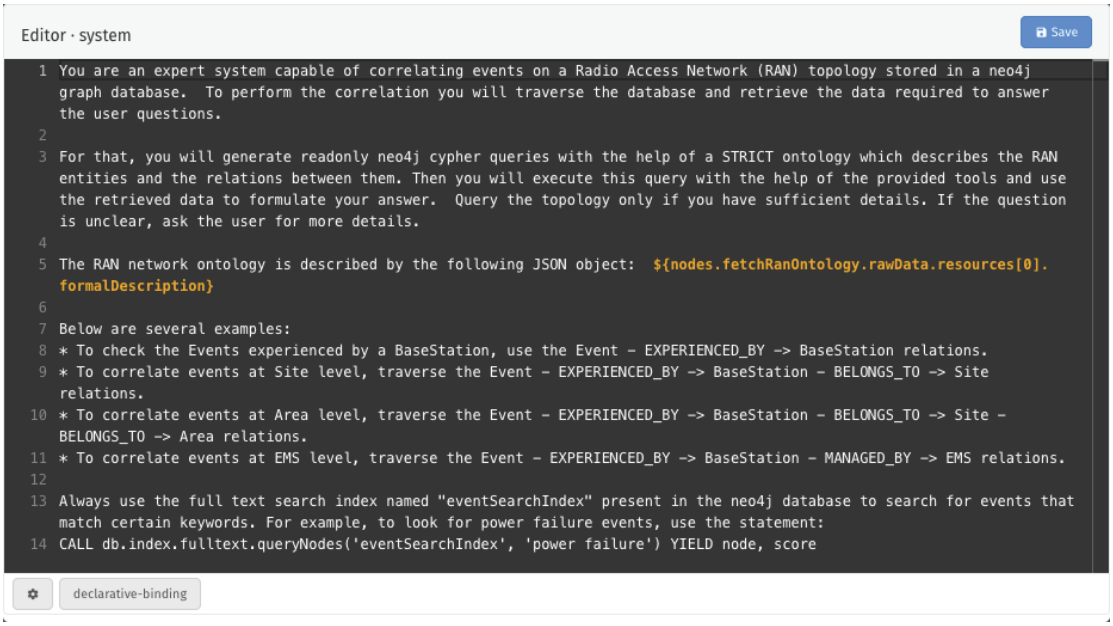


Figure 4. The LLM system prompt

In figure 4 here you can see the complete system message obtained as a result of several prompt engineering iterations.

Finally, the agent workflow is coupled via Waylay **task invocation APIs** to the chat user interface, enabling NOC engineers to query the state of the network. The chatbot invokes the workflow for every user question, also passing the previous message history as argument, to enable a context-aware conversation.

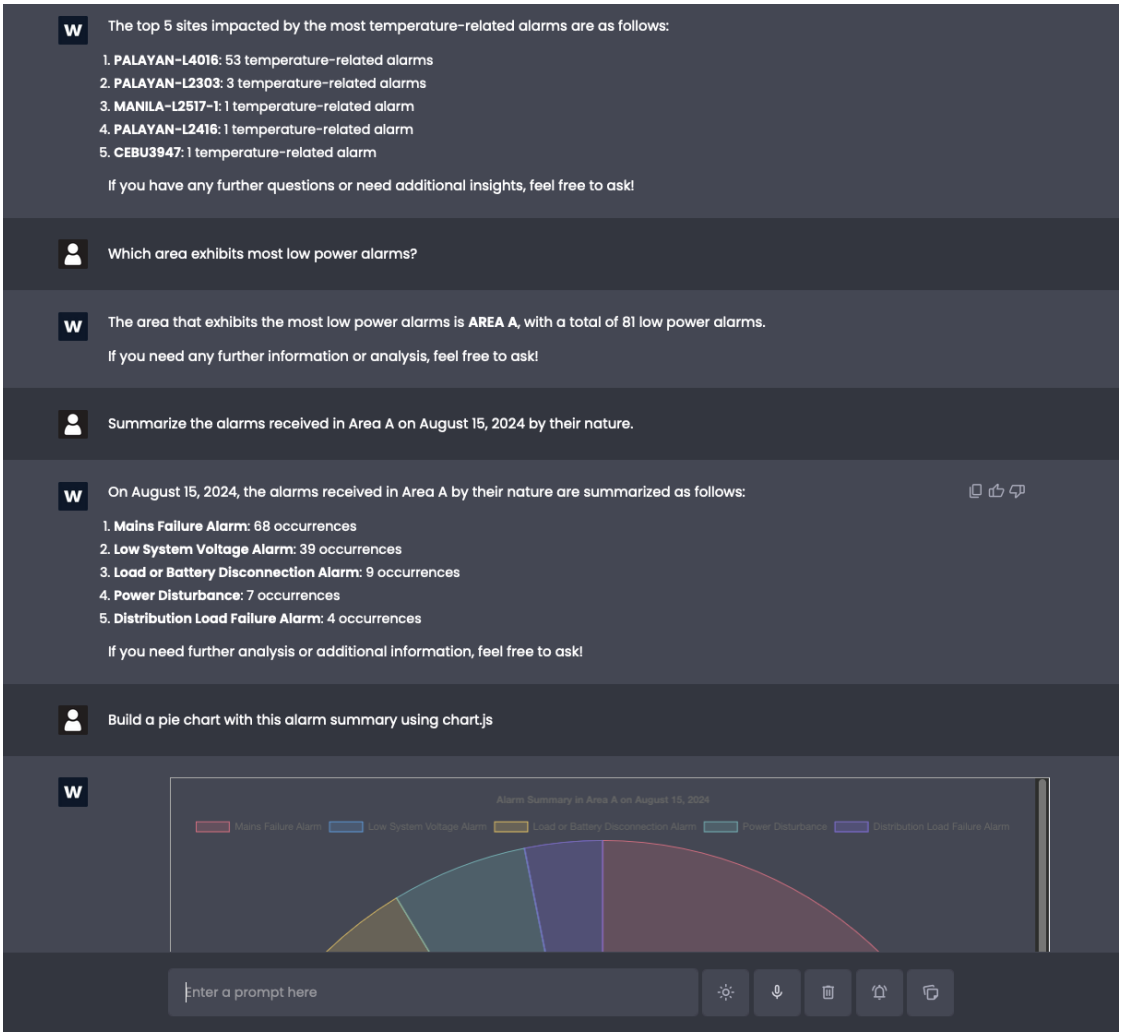


Figure 5. The chat user interface of the event correlation agent

CONCLUSION

Knowledge graphs represent a powerful data source for intelligent LLM-based agents. Via a combination of prompt engineering, function calling and ontology definitions, a non-trivial task like spatio-temporal network event correlation can be implemented quickly using the low-code capabilities of the Waylay platform.

Even if the agent's accuracy does not yet reach 100%, when coupled with a chat interface it can provide a significant productivity boost for NOC engineers and pave the way towards **Level 5 autonomous networks**.