Signal search assistant for aerospace V&V: A multi-agent hybrid graph RAG approach

Sangeeth saagar Ponnusamy[1,*], Carsten Strobel[1] and Alain Kerbrat[2]

[1]Airbus Defence & Space Gmbh, Willy-Messerschmitt-Str 1, Taufkirchen 85521, Germany
[2]Airbus Operations S.A.S, 316 Route de Bayonne, Toulouse 31300, France

Abstract

Complex avionics Verification & Validation (V&V) activities often result in a high volume of fragmented test signal data leading to significant traceability challenges in finding and mapping them to requirements. To address these challenges, a virtual assistant based on Multi-Agent hybrid graph RAG technologies is proposed to query and interact with existing signal databases. Our approach leverages hierarchical agents to augment context based on product and internal knowledge for user queries through an intuitive user interface. The application of this framework on a simplified avionic system demonstrates notable benefits for users while highlighting practical challenges in scalable implementation.

Keywords

LLM Agents, Graph RAG, Verification & Validation, Knowledge Graph

1. Introduction

Model-Based Systems Engineering (MBSE) is increasingly being adopted in the industry to engineer complex systems. Implementing MBSE typically involves transitioning from legacy engineering assets—such as standalone documents and heterogeneous knowledge databases—towards formalized, interconnected model sets. Within this paradigm, MBSE models establish explicit traceability links across engineering artifacts that are otherwise segregated in isolated repositories. A predominant challenge in MBSE is that the levels of abstraction of engineering data varies widely, ranging from high-level abstract concepts during requirement formulation to highly granular technical data during Integration, Verification & Validation (IV&V) activities. This variation in abstraction is particularly pronounced in the upper stages of the V-model in Figure 1, [1]. While top-level requirements and specifications are expressed using abstract, system-level concepts, product verification is performed on integrated products at the sensors and equipment level, dealing with far more detailed and voluminous data. Deploying MBSE in such contexts necessitates not only incorporating these differing levels of abstraction directly into models by synthesizing model artifacts (e.g., models of system-level events, states, and interfaces) down to technical data but also to their respective legacy databases.

This paper focuses specifically on the problem of mapping high-level observable events to low-level, testable signals. High-level observable events are typically used to express system requirements, and are the basic elements to define interfaces between functions and logical components in architecture models. Testable signals, on the other hand, especially when dealing with integrated systems, are often too detailed to be managed within the framework meant for architecture models. These signals are usually defined and managed within specific frameworks and databases, with their own management processes.

Mapping test signals to observable events requires adequate context knowledge of the system architectures, design specifications, test procedures and platforms to name a few. Such knowledge is

Knowledge Graphs and Model-driven Systems Engineering (KGMDSE), in ISWC 2026 Workshops Joint Proceedings, October 25–26, 2026, Bari, Italy

*Corresponding author.
sangeeth-saagar.ponnusamy@airbus.com (S. s. Ponnusamy); carsten.strobel@airbus.com (C. Strobel); alain.kerbrat@airbus.com (A. Kerbrat)
0009-0001-5389-4437 (S. s. Ponnusamy)

© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073
published 2026-11-05

CEUR-WS.org/Vol-4313/kgmdse-paper7.pdf

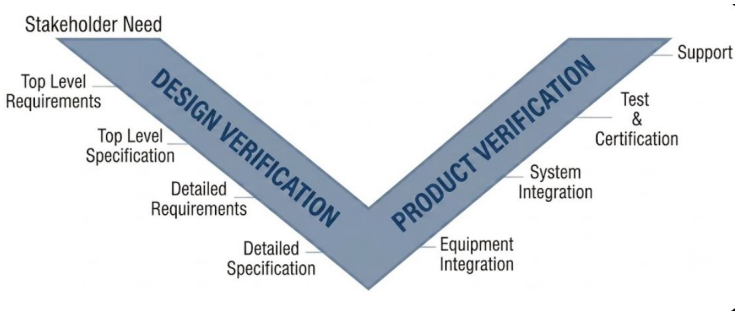


Figure 1: V Model

usually distributed, heterogenous and often implicit. Even with a well defined MBSE model, it is often difficult to navigate and find information without adequate expertise in the tool and methodology [1]. In order to solve this mapping issue, we propose a multi-agent approach leveraging Graph Retrieval-Augmented Generation(Graph RAG) techniques to perform signal search and mapping tasks for avionic system V&V. The motivation for such an approach and its relevant state-of-the-art are presented in Section 2 followed by definition in Section 3 and is experimented on a part of an existing avionics system in Section 4. Section 5 discusses lessons learned, avenues for future work.

2. Motivation & state of art

MBSE provides the structured foundation necessary for rigorous V&V in complex engineering domains. However, effective MBSE deployment often requires significant effort in methodology definition and tool usage [2]. Recent advances in the field of artificial intelligence, particularly Large Language Models (LLMs) present promising opportunities to improve MBSE in general [1, 3] and V&V activities in particular, by generating system modeling artifacts, requirements modeling [4, 5, 6], traceability verification [7], consistency checking etc.

More recently agentic AI systems are becoming a new paradigm [8], by combining LLM agents with Retrieval-Augmented Generation (RAG) and multi-agent orchestration frameworks, these autonomous workflows can query external engineering tools, perform multi-step reasoning dynamically. However, effectively deploying these frameworks in industrial settings requires not only sophisticated data pipelines to convert source documents and knowledge into queryable indexes but also systematic context engineering. Due to the highly domain specific nature of the underlying technical knowledge, the user queries are inherently underspecified and noisy, which often results in semantic drift and hallucinations for unguided RAG. Constructing and enriching the agent's context window prior to execution constrains the retrieval search space, filters semantic noise, and enforces deterministic boundaries on stochastic reasoning as discussed for example in [9].

The need for agents and effective retrieval mechanisms is even more relevant in avionics signal matching tasks, where system signals and telemetry parameters are exceptionally dense, replicated, and context-sensitive. The corresponding database hosts signals spanning all aircraft programmes, including their variants and heads of version, resulting in millions of parameters being stored and managed at any given time. Consequently, a high-level intent such as *GPS ON* can map to dozens of disparate candidates—each carrying distinct flags, mnemonics, or bus signals across heterogeneous equipment, physical topologies, and test configurations. Pre-building architectural context helps ground the search space by integrating structural metadata, such as specific aircraft variants, ATA chapters, functional and hardware configurations etc, before retrieval takes place. The next section outlines RAG in general and graph RAG in particular to build such contexts for agents to perform signal matching tasks.

3. Agents & graph RAG

In general, RAG techniques ground LLMs in domain-specific enterprise knowledge without requiring full model fine-tuning. However, traditional RAG, which relies on dense vector similarity often suffers from inadequate resolution of semantic dependencies and relationships between knowledge fragments in a corpus, often resulting in poor performance [10]. Graph RAG represents these topological information through entities and their relationships, enabling multi-hop reasoning and high-level contextual extraction across a corpus, thereby reducing hallucinations and improving retrieval. Hybrid Graph RAG unifies such symbolic graph representations with classical vector search, coupling the structural logic of knowledge graph ontologies with the broad semantic recall of unstructured dense embeddings. Leveraging such a retrieval mechanism with an agentic framework replaces rigid, single-pass processing with adaptive decision-making logic, empowering the system to autonomously formulate execution plans, evaluate intermediate tool outputs, and perform runtime self-correction [11]. Within this framework, an agent typically interacts with the underlying retrieval infrastructure via the Model Context Protocol (MCP), utilizing a standardized, decoupled interface to dynamically query disparate vector databases and other contextual APIs. A multi-agent architecture where specialized domain agents interact with each other and their MCP tools, either as a self-organizing swarm or through a supervisor-led approach, may often help in deterministic task delegation, structured feedback loops, and scalable resolution.

Even when such agents and MCP tools do exist, in areas such as V&V of complex avionic systems where tests are carried out across different platforms and by different stakeholders, the resulting test signals which are stored across classical databases are often evolving regularly. This necessitates an adaptive approach which not only builds enough context about the signals being searched for from various product specific resources such as Interface Control Documents (ICD), design and test documents, but also flexibly interacts with these evolving signal databases. The context enrichment needs to be as similar as possible to the current manual practice of progressively building and refining relevant information from various data sources, which are often disconnected. In searching the signal database, the search strategies by these agents must be amenable to the internal legacy databases and other infrastructure constraints to retrieve signal candidates. It should also be noted that the objective of such an agentic approach is not finding the signal per se but rather provide the end user with the refined candidate list of signals which shall be validated manually. This human validation step at the end is expressly chosen as our approach, not only to mitigate possible hallucinations or omissions by the agents in safety-critical domain such as avionics, but also due to realistic constraint of challenges in management of implicit engineering knowledge and expertise.

In order to search or map signals to specifications, our approach follows a two-step process, namely, context enrichment of the original query followed by signal matching with this augmented query. In the first step, agent(s) build this context, which may be predefined as list of entities such as associated system, ATA chapters etc with classical RAG and also possibly by interacting with the end user. In the second step, in order to query the signal database through its API, a schema-grounded context engineering approach is employed which first performs entity grounding on a dedicated Knowledge Graph constructed specifically for the API's domain schema. Then the relevant subgraphs around the grounded entities are extracted which is then used as a structured context to synthesize syntactically valid query variations by the agent. These generated variations are then executed via the API to retrieve candidate signals which are also indexed for future cache based retrieval and for final vector based matching. The step-by-step process is described below and illustrated in Figure 2 in next section,

1. **Context Enrichment:** Given a raw query q, extract structured domain contexts $C_{i=1..m} \in \mathcal{C}$ from general and product knowledge indexes via RAG
2. **Graph Grounding & Subgraph Extraction:** Ground entities in $\langle q, C_m \rangle$ onto the knowledge graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ to extract neighborhood subgraph(s) $\mathcal{G}_j \subset \mathcal{G}$ up to depth d from the grounded anchor nodes $n_{j=1..N} \in \mathcal{N}$:

$$\mathcal{G}_j = \{e \in \mathcal{E} \mid \text{dist}(e, n_j) \leq d,\ \forall n_j \in \mathcal{N}\} \tag{1}$$

3. **Query Synthesis & Extraction:** The query q combined with extracted subgraphs $\mathcal{G}_j$ is used to generate syntactically valid query variations $q'_{k=1..K}$, where K is defined by the user. These variations are executed via the API to retrieve a raw candidate signal pool $S_i^0 \subset S$, which are then indexed.
4. **Signal Matching & Human Validation:** Candidate signals $S' \subset S$ matched on the updated index are presented to the user and manually validated.

It may be noted that since the current knowledge graph may not be comprehensive vis-a-vis the context entities, query variation step usually fetches broad range of signals which are then matched again via vector similarity with original query and extracted concepts. The agentic architecture and implementation of this framework is presented in the next section.

4. Implementation

The design and implementation of our assistant is described in this section. The assistant frontend has both an intuitive UI for chat functionalities as well as an API. The agent server acts as a middle end handling input queries and processing outputs from MCP tools which have their dedicated backend. In our current setup, as shown in Figure 2, there are three agents namely, *Supervisor* which in turn manages two sub-agents namely, *Builder* and *Matcher*. The agents have access to different tools, all of them in a dedicated backend server, among them, *context-query* and *signal-match* are pertinent for the current case study. The tools are essentially query engines which are built from different data sources on domain and product specific knowledge. The limited but representative data sources are curated in collaboration with experts and are indexed on a dedicated database server. In addition, it may be recalled that the internal enterprise signal database, denoted as *signal*, has an API and an underlying ontology which in turn is indexed as *signal ontology* and is used for entity grounding. There exists other tools, such as *general query engine* as shown in the Figure 2, which may be used for general query or chat purposes about overall architecture, comparing signals etc which are beyond the scope of this paper. The agentic framework is implemented in Google ADK, leveraging Qdrant vector databases, LlamaIndex data pipelines, Pydantic schemas and Gemini 3.5 models.

As described in section 3, whenever a signal related query is received at the frontend, the *Supervisor* routes it to the respective sub-agent, N=2 in our case, which in turn selects the appropriate tool. For signal mapping tasks, the *Supervisor* calls the *Builder* agent in step (1). This context is then passed to the *Matcher* agent in step (2). To construct a valid query for the signals database, the system first grounds the query and context entities on the Signal knowledge graph in step (3). The matched nodes and subgraphs up to a predefined size of n are extracted and passed on to build query variations, as described in Section 3. In step (4), the queried signals (S') are extracted and indexed in the dedicated signal vector database, which are then used to match final candidate signals, (S^*) in step (5).

In the next section, application of this approach on an avionic case study is discussed.

4.1. Application case

The application case study focuses on a commercial aircraft avionics architecture. In our study, a representative subset of the detailed specification, denoted as $R' \subset R$, is utilized. In the first phase, an AI-based digital assistant [12] extracts the corresponding set of system-level observable events, O'. These system level events are typed with abstract values based on the specification to indicate status such as $system_initialized$: *boolean* or quantitative measures such as $aircraft_altitude$: *feet,* $distance_to_destination$: *nautical miles*. In addition, the modes and states of the system and its functions, while being usually not directly observable, are enumerated as lists, for example $navigation_mode$*:[GPS, IRS]*. This extraction is discussed in [12] and is beyond the scope of current paper, instead the focus is on mapping of such extracted events, O' to their corresponding signals, S'.

In the second step, these observable events $O'_i \in O'$ are processed by our agentic pipeline to produce candidate signal matches $S'_i \in S'$ which are then proposed to domain experts for validation. In the

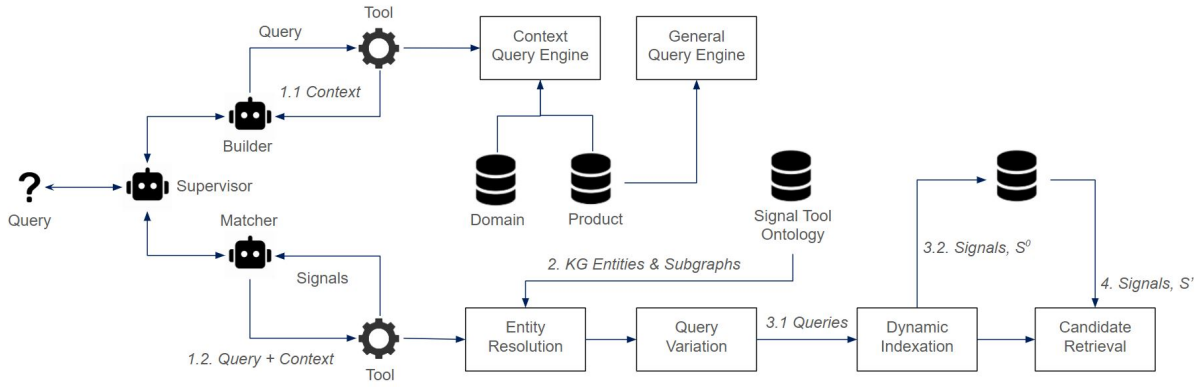


Figure 2: Signal mapping data flow

initial setup, our approach is benchmarked against the available but limited ground truth data, $S_i^* \in S*$, from previous manually performed studies. In order to evaluate the results, an LLM as a judge based on Gemini 3.5 Flash evaluates the execution trace, $\mathcal{T}_i$ over T steps against the ground truth signal S_i^* yielding a j-dimensional composite score $\mathbf{v}_i \in [0,1]^j$. The execution trace is usually defined by:

$$\mathcal{T}_i = \left(a_t,\, \tau_t,\, \mathbf{x}_t,\, \mathbf{y}_t,\, \mu_t,\, j_t\right)_{t=1}^{T} \quad \xrightarrow{\mathcal{E}\left(\mathcal{T}_i, S_i^*\right)} \quad \mathbf{v}_i \in [0,1]^j \tag{2}$$

where each step t records the active agent $a_t \in \mathcal{A}$, invoked tool τ_t, input $\mathbf{x}_t$, output $\mathbf{y}_t$, execution metadata μ_t (latency,queries generated etc), and tool-call justification j_t.

The judge function $\mathcal{E}$ evaluates the trace $\mathcal{T}_i$ against ground truth S_i^* across predominantly three axes: **Output Quality**—measuring *Accuracy* (signal proximity to S_i^*), *Completeness* (recall), *Conciseness* (zero spurious outputs/drift), and *Justification Quality* (reasoning chain validity j_t); **Execution Quality**—assessing *Context Convergence* (domain boundary adherence in Figure 2 step 1); and **Pipeline Quality**—quantifying *Transfer Fidelity* (field preservation across agent handoffs), *Grounding Quality* (matcher-to-signal ontology alignment in Figure 2 step 2), and *Query Coverage Quality* (for query generation in Figure 2 step 3). These preliminary set of metrics are continuously being updated and is presented for the sake of completeness. In addition to such metric evaluation, the judge also assess the execution trace $\mathcal{T}_i$ and qualitatively attribute failure with respect to the ground truth which may help in better understanding and diagnostics of the underlying multi agent interaction. The next section briefly outlines some of the results.

4.1.1. Results

Sensitivity studies were first performed on the chosen set of context entities $C_{i=1..m}$ against a subset of metrics presented in the previous section. Combinations of the most sensitive contexts were initially selected to extract relevant context information for signal matching, before progressively expanding the scope to cover the full range. The system's sensitivity to representative context entities, where m=9, against normalised composite score is shown in Figure 3 for the purpose of illustration.

In the next step, tradeoff studies were performed across agent architectures, tool combinations, collection scopes, and LLM models to identify optimal configurations. For instance, Table 1 illustrates the effect of sub-agent cardinality ($N = 1, 3$) on key metrics, including total execution time, call volume, token consumption, and composite score with respect to our baseline implementation of N=2.

Further studies are currently being carried out in this ongoing research project in collaboration with the stakeholders. Early indications point out to promising trends in faster signal search with better access and interaction with the underlying knowledge. In addition, with our agentic implementation,

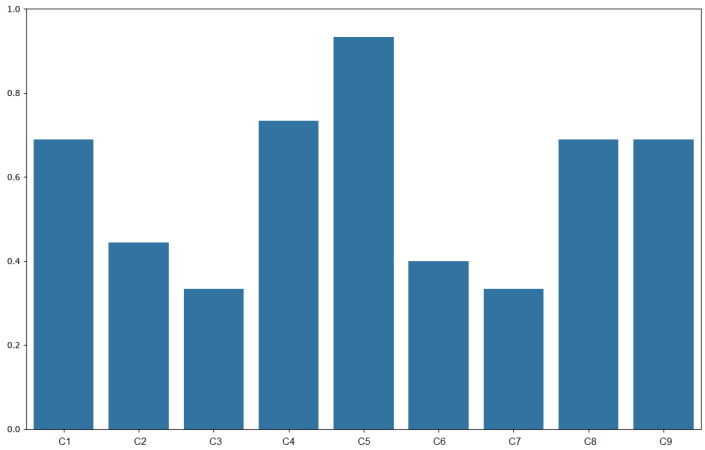


Figure 3: Sensitivity Analysis

Table 1
Agent architecture tradeoff

Description (%)	N=1	N=3
Total Time taken	-76.8	10.6
Token consumption	-81.6	25.5
Composite score	-62.2	5.4

not only the final results but also the intermediate steps such as tool calls, knowledge graph entity groundings, query generations etc are annotated with LLM generated justifications/explanations, which would help in further validation or fine tuning the approach.

5. Conclusion & outlook

The paper discussed a virtual assistant for engineers practicing V&V to search and map signals, leveraging agentic and graph RAG technologies. While early results are promising especially in assisting engineers to link different databases and extract context, further studies need to be performed to adequately quantify the benefits vis-a-vis the current manual baseline. Nevertheless such an incremental indexing of engineering data sources combined with agentic approaches shall be useful for other purposes as well, such as domain-specific engineering assistants, test case authoring and execution to name a few. Future developments will focus on addressing practical scalability challenges and incorporating reinforcement learning [13] techniques to optimize graph traversal and tool orchestration to increase the efficiency and accuracy.

Acknowledgments

The authors would like to thank Adil Soubki from Airbus Operations for his thoughtful suggestions and constructive feedback, Gerritt Schramm and Stefan Richter from Airbus Defence & Space Gmbh for their contributions.

Declaration on Generative AI

The author(s) have not employed any Generative AI tools in preparing this manuscript.

References

[1] S. Ponnusamy, P. Helle, A. Zindel, S. Richter, G. Schramm, C. Strobel, IBIS : An Interactive Virtual Assistant for System Engineers, in: AIAA SCITECH 2023 Forum, 2023. doi:10.2514/6.2023-1960.

[2] K. X. Campo, T. Teper, C. E. Eaton, A. M. Shipman, G. Bhatia, B. Mesmer, Model-based systems engineering: Evaluating perceived value, metrics, and evidence through literature, Systems Engineering 26 (2023) 104–129. doi:https://doi.org/10.1002/sys.21644.

[3] Z. U. Rizqi, O. Madsen, C. Schou, B. C. Dueholm, T. Schwörer, M. B. Jensen, W. Henrik Flach de Neergaard, K. B. Hansen, O. V. Olesen, Model-Based AI-Driven Systems Engineering (MBSE-AI): Towards Fast and Effective Decisions, in: 2025 IEEE International Symposium on Systems Engineering (ISSE), 2025, pp. 1–8. doi:10.1109/ISSE65546.2025.11370010.

[4] R. Gudder, D. Hoffmann, P. Hünecke, A. Lüder, LLM-Driven Requirements Formalization for Model-Driven Manufacturing Systems, in: 2025 IEEE 30th International Conference on Emerging Technologies and Factory Automation (ETFA), 2025, pp. 1–4. doi:10.1109/ETFA65518.2025.11205753.

[5] N. Shevchenko, Requirements in Model-Based Systems Engineering (MBSE), 2021. doi:10.1184/R1/14077232.v1, accessed: 2026-Jul-23.

[6] A. Vogelsang, J. Fischbach, Using Large Language Models for Natural Language Processing Tasks in Requirements Engineering: A Systematic Guideline, Springer Nature Switzerland, Cham, 2025, pp. 435–456. doi:10.1007/978-3-031-73143-3_16.

[7] M. Bonner, M. Zeller, G. Schulz, A. Savu, LLM-based Approach to Automatically Establish Traceability between Requirements and MBSE, INCOSE International Symposium 34 (2024) 2542–2560. doi:https://doi.org/10.1002/iis2.13285.

[8] M. Abou Ali, F. Dornaika, J. Charafeddine, Agentic AI: a comprehensive survey of architectures, applications, and future directions, Artificial Intelligence Review 59 (2025). doi:10.1007/s10462-025-11422-4.

[9] Q. Zhang, C. Hu, S. Upasani, B. Ma, F. Hong, V. Kamanuru, J. Rainton, C. Wu, M. Ji, H. Li, U. Thakker, J. Zou, K. Olukotun, Agentic Context Engineering: Evolving Contexts for Self-Improving Language Models, 2026. URL: https://arxiv.org/abs/2510.04618. arXiv:2510.04618.

[10] B. Peng, Y. Zhu, Y. Liu, X. Bo, H.-Z. Shi, C. Hong, Y. Zhang, S. Tang, Graph Retrieval-Augmented Generation: A Survey, ACM Transactions on Information Systems 44 (2025) 1–52. doi:10.1145/3777378.

[11] F. Neha, D. Bhati, Traditional RAG vs. Agentic RAG: A Comparative Study of Retrieval-Augmented Systems, TechRxiv 2025 (2025). doi:10.36227/techrxiv.175624551.12254549/v1.

[12] S. Ponnusamy, A. Kerbrat, C. Strobel, An End-to-End Agentic Assistant for Avionics V&V, in: International Conference on Model-Driven Engineering Languages and Systems, Industry Days - Accepted, 2026.

[13] M. Hariharan, Reinforcement Learning Integrated Agentic RAG for Software Test Cases Authoring, 2025. URL: https://arxiv.org/abs/2512.06060. arXiv:2512.06060.

6. Online Resources

- ADK
- Gemini
- LlamaIndex
- Model Context Protocol
- Pydantic
- Qdrant