Project Plan Summary

Phase 1 | November 6, 2026 | Deadline Saturday, November 7, 2026, 11:59 PM ET

Your name	David Allen Blubaugh
Project title (as posted)	BIRE Agentic AI–MBSE: Traceable Development and Assurance for Safety-Critical Systems and Software
Link to your project post	https://www.teraopenscience.com/project/11767-bire-agentic-ai-mbse-traceable-development-and-assurance-for-safety-critical-systems-and-software/
Email	davidblubaugh2000@yahoo.com

Jury summary in the organizer's standard template
Prepared following Patrick Deconinck's November 6 suggestion. Every question is answered below.
Supporting document: ProjectPlan_Blubaugh_BIRE_Master_Rev3_XPlane.pdf, Revision 3, November 6, 2026 (23 plan pages plus 30 original presentation pages). Page references below refer to its 23-page plan.
Proposed hours, costs and performance thresholds are planning assumptions, not completed results. The later organizer emails extend the original template deadline from November 6 to November 7.

How your plan is scored

The sections below match the jury's scoring sheet. Spend your effort where the weight is.

#	Section	Weight	What the jury looks for
1	**Innovation**	**25%**	What's new and different
2	**Technical plan**	**25%**	How you'll build and test it
3	**Execution plan**	**25%**	Milestones, resources, supply chain, business
4	**Capacity to deliver**	**15%**	Your skills, support and time
5	**Real-world impact**	**10%**	Who it helps and why it matters

0 Project summary

In 3–5 sentences: what are you building, who is it for, and what will you have by April 23, 2027?

You can adapt your posted short description.

> I propose a research software assistant that connects requirements, SysML v2 models, software, tests and telemetry through a versioned engineering knowledge graph. BIRE provides the aerospace case; small engineering and research teams are the intended users. By April 23, 2027, I plan to deliver a controlled requirement-change demonstration, a small game-subsystem transfer experiment and a reproducible evaluation. Plane Maker and X-Plane establish the aircraft prototype before Unreal Engine 5 visualization. AI proposes changes; people approve engineering baselines.

1 Innovation 25% of score

What is new about your idea?

A new finding, method or application beyond what's known today.

> The proposed contribution is a version-specific evidence contract and controlled change workflow tested in two domains. It will explain why the same telemetry can receive different judgments under different requirement revisions while preserving both evaluations. Traceability, knowledge graphs and agents already exist; the research tests this particular combination rather than claiming those methods are new. Master pp3, 6, 18.

How is it different from existing tools, methods or products?

Name the closest alternatives.

> Closest alternatives include IBM DOORS Next and Jama Connect for traceability and reviews, Git/scripts for versioned checks, relational tables for linked records, and vector retrieval for similar text. The prototype will compare these approaches using equivalent evidence, then test whether specialized agents improve review beyond deterministic graph queries and a single assistant. No commercial-tool superiority is assumed. Master pp3, 18.

What does it improve, and by roughly how much?

Performance, reliability, scale or cost.

> Proposed targets: impact precision and recall each ≥90%; supported trace precision and recall each ≥95%; detection of ≥18/20 seeded defects with ≤2/20 false alarms on clean controls; 12/12 baseline questions correct in every final AI repeat; and ≥20% lower median review time than deterministic graph review without lower answer accuracy. Report failures and limitations. Master pp18–19.

What part of this is your own original thinking?

> My original work is the proposed BIRE evidence contract, source-to-graph integration, controlled requirement-change experiment and transfer evaluation, building on my digital engineering research. Published standards and third-party tools remain attributed. The presentation reports prototype artifacts and planned functions; it does not establish a complete autonomous engineering system. Master pp2–3.

2 Technical plan 25% of score

How will you build it, and what tools, equipment or facilities do you have access to?

Use Mg/MgPy for SysML v2 modeling and supported extraction; canonical JSON/Python for validation; Neo4j for versioned trace relationships; and Docker Compose for the backend. Develop the aircraft definition in Plane Maker and test X-Plane feasibility before Unreal replay/visualization. Verify BIRE aerodynamic coupling; animation alone is insufficient. A plugin or external solver is conditional.
Resources are proposed, not secured: existing suitable workstation; Mg reserve $400; model API $600; compute $450; storage $150; review $450; Unreal assets $100; discovery $100; X-Plane professional digital license $1,000; contingency $750. Total cash $4,000. Public software carries applicable license obligations. Custom flight-control engineering support is not assumed included with the X-Plane license. Master pp4–14, 21.

What are the biggest unknowns, and how will you resolve each one?

Unknowns and resolution: persistent identity/duplicates—test rename and repeat imports; Mg access/semantic fidelity—validate a supported fixture by December 15; rotating-tail dynamics—document X-Plane limitations and select the justified route by January 15; AI evidence reliability—test stale versions and missing citations; usefulness—compare review quality/effort; transfer—measure a small game adapter. Missing access triggers an explicitly labeled fixture fallback. Master pp5, 7, 10–14, 18, 20.

How will you test, prototype or iterate?

Freeze source revisions, validate imports, run deterministic checks, stage proposals, review changes and preserve results. Test a notional BIRE limit change from ±25° to ±20° using identical synthetic telemetry; a 23° sample changes verdict, while a full run still depends on every sample. Evaluate 12 change tasks, 20 defects, 20 clean controls and 12 version questions. Repeat AI tasks three times. Master pp16–19.

How will you know it works?

Name the success criteria or numbers you'll measure.

Success requires the numerical targets above plus no duplicate accepted revision identities, no invalid published imports, correct historical-baseline retrieval, denied unauthorized agent writes, three deterministic replays and an isolated database restore. Compare manual, scripted/relational, deterministic Neo4j, single-assistant and specialized-agent workflows. Report author bias and small-sample limits. Passing this research benchmark does not establish aircraft safety or certification. Master pp18–19.

3 Execution plan 25% of score

3.1 Milestones

One goal per period, with how you'll know it's done. Progress reports are due at the 2- and 4-month marks; final defenses are Apr 19–23.

Period	Milestone	Done when...
Nov 16–Dec 15, 2026	Freeze sources and environment	50 h. Review rights, hardware, Mg access, identity rules and schema. Configure initial Plane Maker baseline; clarify X-Plane license/support. Done when fixtures import, unsupported constructs are reported and builds/cost assumptions are recorded.

Period	Milestone	Done when...
Dec 16–Jan 15, 2027	X-Plane feasibility and BIRE evidence	50 h. Retain both requirement-version evaluations; prevent repeat-import duplicates; test restore; archive aircraft package, telemetry and dynamics limitations. Done when provenance checks pass and the justified simulation route is recorded. Submit 2-month report by Jan 15, date convention to confirm.
Jan 16–Feb 15, 2027	Agents and Unreal replay	55 h. Integrate bounded agent roles and a small Unreal client after the X-Plane gate. Done when baseline selection, citation, permission and hold tests pass and replay uses tested frame/unit conversions.
Feb 16–Mar 15, 2027	Transfer and user feedback	50 h. Add a small game speed-limit adapter; record core reuse and effort. Seek 3 interviews and 1 supervised evaluation. Done when both domains replay and preliminary findings, including missing feedback, are reported. Submit 4-month report by Mar 15, date convention to confirm.
Mar 16–Apr 23, 2027	Evaluation and final defense	45 h plus 26 h contingency available across the build. Freeze evaluation by Mar 19; report by Apr 9; clean install/restore by Apr 16; defend Apr 19–23. Done when raw results, costs, limitations and evidence package are complete. No readiness claim with unresolved critical failures.

3.2 Business outline

Not a full business plan yet. One or two lines each.

Target customer: who pays for this?

> Initial paying-customer hypothesis: small aerospace/robotics R&D firms and university laboratories needing inspectable requirement-change evidence. Seek three discovery interviews and one supervised evaluation by March 15, 2027. No customer or revenue commitment is claimed. Master p22.

Value: why would they choose it over what they use today?

> Customers would choose a scoped adapter/evidence service if a pilot demonstrates less review work with equal or better accuracy and useful integration into their existing process. Public synthetic examples and reproducible evidence will support evaluation. Value remains to be measured. Master pp18, 22.

Rough economics: price, cost to make or deliver, and how it makes money.

> Proposed pilot: $3,000 including $300 compute; assumed direct cost 30 h × $60 + $300 = $2,100; contribution $900 (30%) before overhead and taxes. Client simulator licenses/custom engineering are quoted separately. Prototype cash estimate $4,000; founder labor 276 h × $60 = $16,560 imputed, unpaid; total economic estimate $20,560. Master p21.

Path to market: how will you reach your first users or customers?

Use existing engineering/academic contacts and TeraOpenScience introductions for discovery, then show a synthetic-data demonstration. Agree data permissions, ownership, scope and acceptance criteria before a supervised paid pilot. Review institutional rights, supplier licenses and platform terms before commercialization. No partnership or institutional endorsement is assumed. Master p22.

4 Capacity to deliver 15% of score

Your background in this field

Degree, work experience or past projects that show you can do this. 2–3 lines.

MS Electrical Engineering, Wright State University (2012); BS Electrical Engineering (2004); current doctoral work in Electrical and Systems Engineering at the University of Dayton. Relevant applicant-reported experience includes Python, embedded systems, modeling, formal-method tools and BIRE digital engineering. Coauthor: Intelligent Autonomous Drones by Cognitive Deep Learning (Apress, 2022). Master p22.

Skills this project needs, and who covers each

Include advisors, mentors, contractors or collaborators if you have them.

Skill area	Who covers it	Their role
Technical / engineering	David Allen Blubaugh	Own models, adapters, graph, containers and simulation integration.
Testing / experiments / data	David Allen Blubaugh	Own fixtures, protocol, tests and results; disclose solo-study bias.
Business / customers	David Allen Blubaugh	Own discovery, cost tracking, pilot scope and rights review.
Other: academic advice	Dr. Thomas Ford, proposed reviewer	Advisor; challenge involvement subject to consent and availability. No commitment assumed.

Time and support for Nov 16 – Apr 23

Hours per week you can commit, plus any lab, school or employer support.

Proposed allocation: 12 h/week for approximately 23 weeks, totaling 276 h (250 planned plus 26 contingency), subject to availability confirmation. David leads all work. Dr. Thomas Ford is the applicant-reported doctoral advisor; challenge-specific review depends on consent and availability. No institutional funding, lab allocation, paid assistance or vendor support is committed. Confirm hardware, cash and access at the first gate. Master pp20–22.

5 Real-world impact 10% of score

What real need does this address, and what evidence do you have?

E.g., people you've talked to, data, published studies.

> Requirement changes can leave obsolete tests and evidence appearing current. The supplied BIRE presentation and established requirements-management documentation support the need for traceability and change review. This project will test a concrete evidence-continuity workflow. No completed customer interviews or measured productivity gains are asserted. Master pp2–3, 22.

What could it lead to?

New applications, markets or research directions.

> Possible outcomes include a reproducible benchmark, engineering-tool adapters, teaching examples and scoped integration services. Aerospace, embedded software, robotics, industrial automation, general software, games and digital twins are potential applications. Only BIRE and one small game subsystem are evaluated here; wider reuse needs domain-specific semantics and tests. Master pp3, 17, 22.

Who benefits?

Health, safety, education, sustainability or equity.

> Engineering reviewers could gain clearer provenance and earlier detection of stale evidence; researchers and students could gain inspectable examples and reusable fixtures. Potential safety benefit is improved evidence review, not demonstrated accident reduction. No quantified health, environmental or equity benefit is claimed. Operational use needs additional assurance and validation. Master p22.

6 Confirmation

> *"All work will follow applicable university, platform, consumer-safety, consent, and privacy requirements."*

Type your full name and today's date to confirm the statement above.

Full name	David Allen Blubaugh
Date	November 6, 2026

> Before sending
> Every template question is answered; monthly completion evidence and resource costs are included.
> Applicant review: confirm proposed hours, $4,000 cash estimate, workstation, source permissions and vendor access.
> The typed name/date accompany the organizer’s confirmation statement; no handwritten signature is represented.
> Send this summary with the supporting Revision 3 master plan by November 7, 2026, 11:59 PM Eastern. The organizer has suggested this format; no jury acceptance or eligibility determination is implied.

MASTER PROJECT PLAN

ADDITIONAL SUPPLEMENTAL INFORMATION

For jurors who require
in-depth technical information

The completed standard-template summary appears first.
The following master plan provides supporting detail
and includes the original presentation appendix.

David Allen Blubaugh | November 6, 2026

Project purpose and jury overview

BIRE Agentic AI MBSE

Traceable Development and Assurance for Safety-Critical Systems and Software

TeraOpenScience Science Challenge | Master project plan | Revision 2 | November 5, 2026

Applicant: David Allen Blubaugh
Contact: davidblubaugh2000@yahoo.com
Build period: November 16, 2026-April 23, 2027

The engineering question

A tail-angle sample reads 23 degrees. One requirement permits it; a revised requirement does not. Which model, configuration, test and decision belong to each judgment? This project will make that answer inspectable.

I propose a research assistant that connects mission needs, SysML v2 models, software, tests and telemetry through a versioned knowledge graph. BIRE provides the aerospace case. Mg and MgPy provide the intended model interface. Neo4j stores engineering relationships. Docker packages the backend. Unreal Engine 5 provides an interactive simulation and evidence-review client. Controlled agents propose changes; people accept baselines.

By April 23, 2027, the planned result is a reproducible BIRE requirement-change demonstration, a small game-subsystem transfer test, and a measured comparison with simpler workflows. The deliverable is research software and evidence, not flight-qualified control software or a certification claim.

Jury criterion	Where the plan answers it
Innovation 25%	Pages 3, 6 and 15: bounded novelty, alternatives and comparative tests.
Technical plan 25%	Pages 4-16: interfaces, graph semantics, deployment, simulation and verification.
Execution 25%	Pages 17-18: monthly gates, hours, supply chain and costs.
Capacity 15%	Page 19: applicant skills, responsibility and support limits.
Impact 10%	Pages 3 and 19: needs, beneficiaries and paths to adoption.

Reading guide: sources and corrections p2; architecture p4; Mg p5; knowledge graph p6; Neo4j p7; Docker p8; recovery p9; Unreal p10-11; agents p12; experiments p13-14; evaluation p15-16; execution p17-19; references and confirmation p20. Appendix A starts at PDF page 21.

Source reconciliation and capability status

The new attachments are byte-identical to the earlier named presentation and first-plan PDFs. The current copies are BIREPROGRAM(3).pdf and bire_Blubaugh_projectplan(2).pdf. Their source hashes are recorded in the embedded provenance manifest. The organizer template copy is ProjectPlan_Template_Science-Challenge-(2).docx. [S1-S5]

Source evidence	Status and consequence for this plan
Presentation 1, 7, 29	The title is BIRE Digital Engineering Ecosystem; slide 7 says Summer 2026 Research Objective and slide 29 describes summer SOCHE learning. PDF creation metadata is October 31, 2026. This supports the supplied summer-work context, not an independently verified delivery date.
Presentation 11-13, 19-22, 25	Architecture, exporter examples and graph mappings are shown. Treat these as reported prototype artifacts pending code, dependency and replay inspection.
Presentation 21	Duplicate detection is reported, but repeated imports can create duplicates. Identity and repeat-import integrity are mandatory early work, not established capabilities.
Presentation 12, 26	Persistent identity, expanded verification, telemetry mapping, runtime assurance, simulation and AI reasoning include future work. They do not establish a complete integrated system.
Presentation 16-17	Verification values and dashboard data are explicitly illustrative; the dashboard is a representative prototype design. They are not benchmark results.
Presentation 23-25, 28, 30	Controller architecture and tool lists are context. Claims of complete safety, universal performance or mutual dependence of SysML and graphs are narrowed to testable hypotheses.

Material corrections to the first plan

The first plan could not inspect the presentation; this revision does. It adds a defined Unreal client and container operations workstream. Its earlier $3,000 pilot wording was ambiguous: the revised price includes the $300 compute allowance. The proposed 276-hour allocation is rebalanced on page 17. The $3,000 prototype ceiling remains conditional, with revised reserves on page 18. [S2 pp5-6, 10-13]

The source text ends an instruction mid-sentence after “Recommend deterministic validation, software testing, an” and contains an ellipsis before a prior response. No missing passage is reconstructed. The earlier drafting prompt is a requirement source only. Logos, contributor names and institutional roles in the slides do not establish present endorsement or challenge staffing. [S1, S3 pp1-3, S5]

Innovation alternatives and application scope

A bounded research contribution

My contribution is the proposed evidence contract, controlled change experiment and cross-domain evaluation built on my BIRE digital engineering work. Traceability, graph storage, retrieval-augmented generation and agent orchestration are established techniques. The experiment asks whether their disciplined combination improves review quality or effort on a small, explicit workload. Novelty will be refined through a first-month literature and product review. [S2 pp2-3; S3 pp6-11, 27]

Alternative	What it already does	Question to test
Relational tables / joins	Represent revisions, links and constraints with explicit schemas.	Does the graph reduce query/adapter effort for changing multi-hop traces?
Documents / spreadsheets	Communicate decisions and maintain small trace matrices.	Does a structured workflow catch stale evidence with less manual reconciliation?
Git / scripted manifests	Track files, commits and repeatable executable checks.	Does graph navigation improve impact discovery beyond the same manifests?
Vector retrieval	Find semantically similar text.	Does baseline filtering plus typed traversal reduce incorrect-context retrieval?
DOORS Next / Jama	Provide requirements, traceability, impact analysis and review facilities. [W12-W13]	Is this small adapter/evidence experiment useful alongside existing tools?

Why explicit relationships are central

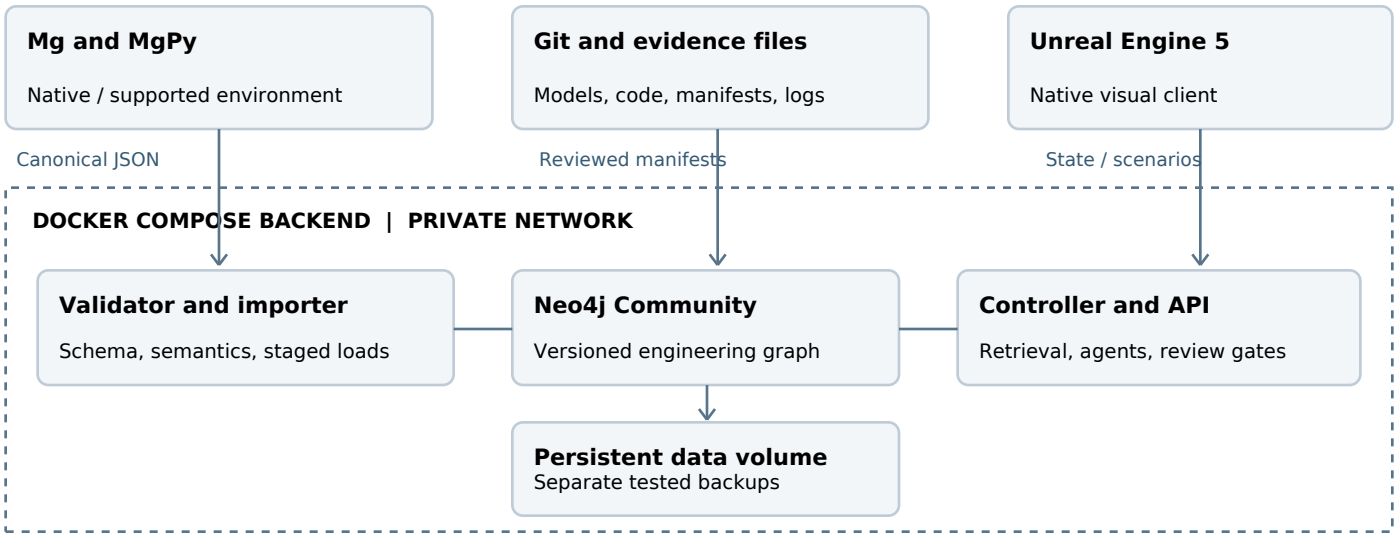
The hypothesis concerns connections between versions, not merely retrieval of similar paragraphs. A requirement-to-test path can identify candidate evidence. The evaluation still needs a valid test, correct data and an applicable baseline. A missing edge can hide an impact; a wrong edge can create one. Neither a graph path nor a cited source proves requirement satisfaction.

Applications and practical limits

Aerospace teams could inspect model and control-configuration changes. Embedded, robotics and industrial-control teams could connect interfaces, interlocks, tests and logs. General software teams could trace dependencies and releases. Games provide a lower-risk transfer example. Digital twins and teaching provide inspectable replay exercises. These eight application areas preserve the posted scope; only BIRE and one small game subsystem are build commitments. [S1; S2 pp7-8]

Targets include at least 90% impact precision and recall and a 20% review-time reduction against the deterministic graph baseline without lower accepted-answer accuracy. These are proposed thresholds, not measured improvements. Graph value and agent value are tested separately on page 15.

System architecture and artifact authority



The native model environment exports a versioned exchange package. A containerized validator checks structure and semantics. The importer stages graph revisions. The controller retrieves approved evidence and orchestrates proposals. Unreal consumes scenario/state messages and reports client observations. Large files remain outside the graph. This is a proposed deployment, not a claim that all interfaces already operate. [S3 pp11, 13, 19-23]

Artifact	Authoritative record and synchronization rule
Requirements / architecture	Reviewed Mg source revision, archived with its export and hash. Graph data are a derived projection; disagreement returns to the source owner.
Code / configuration	Git commit plus immutable configuration artifact. A proposed patch stays on an isolated branch until accepted.
Simulation input / output	Versioned scenario, solver build and run manifest in evidence storage. A recording does not become physical validation.
Raw telemetry / assets	Original files and content hashes in access-controlled file storage. Normalize into a new derived dataset; retain the original.
Accepted baseline / decisions	Versioned manifest plus reviewed decision record. Graph nodes index these records; the graph is not independently authoritative for approval.
Derived graph relationships	Imported links cite exact source revisions. Proposed/inferred links carry separate status and never silently replace authored engineering meaning.

Authority is explicit rather than universal. Model, code and evidence repositories each govern their own artifacts. A baseline manifest binds exact revisions across them. The controller refuses an unknown or conflicting baseline. There is no autonomous synchronization that can overwrite an accepted source record.

Mg extraction and canonical exchange contract

Mg is Mgnite’s SysML v2 environment; MgPy supplies Python-oriented automation and model exploration. Official material documents an IPython-based kernel and Python model access. It does not establish that this project’s exporter preserves every construct or identity across revisions. No undocumented API is assumed. [W1-W2; S3 pp18-20]

Minimum supported model subset

Start with packages, requirement definitions, part definitions/usages, ownership and selected typed references. Preserve units and source locations. Satisfaction, allocation and verification mappings are accepted only where their actual source semantics are checked. Behaviors, expressions and other unsupported constructs remain source records with explicit exclusions. Unsupported content cannot become verified links through inference.

Record	Mandatory fields and meaning
Envelope	schema_version, export_id, project_id, source tool/build, source revision, baseline_id, declared units and construct support, export time, content hash.
Artifact revision	entity_id, revision_id, kind, name, source locator, source revision, content hash, provenance; definition/usage and ownership where relevant.
Relationship assertion	assertion_id, relation type, source/target revision IDs, evidence locator, origin, status and baseline applicability.
Evidence / evaluation	dataset/run/test IDs and hashes, origin, unit/timebase metadata, selected requirement/configuration revisions, verdict, logs and execution context.
Finding / decision	Task/run/agent IDs, claim, exact citations, uncertainty, proposed patch hash, reviewer, disposition, rationale and timestamp.

Integration experiment by December 15

Use a small authorized model with rename, reparent, definition/usage, cross-reference and unit cases. Export it through the installed supported Mg/MgPy route. Compare every supported field to a manually reviewed fixture. Repeat the export and import. Required results: no silent construct loss; resolved identity crosswalk; no dangling accepted links; stable canonical content for unchanged inputs. Record installed versions, errors and permitted redistribution.

Canonicalization specifies key ordering, Unicode normalization, array ordering where meaningful and numeric/unit serialization. Volatile timestamps and execution IDs belong in the envelope, outside the semantic content hash. A stable byte format alone does not establish persistent semantic identity. [S3 p19]

If supported access fails or fidelity is inadequate, use declared standards-based fixtures. That preserves downstream experiments but leaves real Mg extraction and source round-trip fidelity incomplete. Mg/MgPy containerization is optional and subject to supported deployment and license.

Knowledge graph semantics and relationship catalog

The graph represents engineering entities and meaningful relationships. Neo4j stores and queries a labeled property graph; the project supplies its semantics. It is not automatically an RDF/OWL ontology or a formal reasoner. Version selection precedes traversal and any semantic ranking. [S3 pp14, 22, 25, 30]

The core question follows this chain: requirement revision, model constraint, software/configuration revision, test revision, run, dataset, evaluation and review. The chain supports candidate impact discovery and evidence inspection. The small prototype can be implemented another way; Neo4j is a research choice to evaluate, not a universal prerequisite.

Relation and direction	Meaning / permitted endpoints	Origin and scope
HAS_REVISION: Entity → Revision	A durable entity owns an immutable artifact revision. Revision kind identifies requirement, model, code, config or test.	Importer; exact entity/revision key.
SELECTS: Baseline → Revision	Manifest selects one applicable revision for a logical artifact.	Authored manifest; accepted baseline only.
OWNS: ModelRevision → ModelRevision	Source model ownership; preserve definition/usage kinds.	Imported; same declared model context.
TYPED_BY: UsageRevision → DefinitionRevision	A usage references its definition.	Imported; cited source revision.
ALLOCATED_TO / CONSTRAINS: RequirementRevision → ModelRevision	Explicit allocation or constraint; these two meanings are not interchangeable.	Authored/imported; baseline-bound.
IMPLEMENTS: CodeOrConfigRevision → RequirementRevision	Reviewed implementation trace, not a proof of correctness.	Authored; reviewed assertion.
CHECKS: TestRevision → RequirementRevision	Test expresses a scoped verification check.	Authored; requirement revision explicit.
EXECUTES: Run → TestRevision; USES: Run → Revision	Run executes a test and uses exact configuration/model/code revisions.	Runner manifest; baseline explicit.
PRODUCES: Run → Dataset; EVALUATES: Evaluation → Dataset	Run provenance and dataset examined by a recorded evaluation.	Runner/checker; immutable IDs/hashes.
AGAINST: Evaluation → RequirementRevision; BASED_ON: Decision → Evaluation	Requirement applied and evidence used for disposition.	Checker/reviewer; exact revisions.
CITES: AssertionOrFinding → SourceRecord	Claim or relationship points to exact source location.	Importer/agent; provenance recorded.

Each assertion also has an entity-independent assertion_id, source locator, origin and status. Reify assertion metadata as nodes if edge properties cannot express the required history. AI-proposed links stay in proposal records. Review creates a new accepted assertion without erasing the proposal. Additional relationship types require schema review.

Neo4j integrity and baseline publication

Proposed deployment: single-node Neo4j Community Edition on Linux amd64 using the official image. Candidate tag: neo4j:2026.10.1, documented by the vendor at preparation. At Gate G1, verify compatibility and security, select the exact supported patch, resolve its digest and freeze it with the Python driver and schema version. It is not installed or tested by this plan. Existing 5.x research stores must not be opened in a newer server without an isolated compatibility/migration test. [W3; S3 p21]

Project-owned identity

Use a project namespace plus persistent entity_id, immutable revision_id and source-system ID crosswalk. A display name or Neo4j internal ID is not the durable cross-tool key. A rename retains identity only with source support or a reviewed mapping. Ambiguous identity enters quarantine. A content hash detects content change; two distinct entities may still have identical content.

Validation and transactions

Validate JSON structure, field presence, type semantics, units and endpoint existence before import. Apply uniqueness constraints to project entity, revision and assertion keys; index baseline/status/kind fields according to measured queries. Community-supported uniqueness is useful, while other constraint types depend on edition. Required-field and endpoint rules remain application checks. Use parameterized values and allowlisted labels/types. [W4]

MERGE uses explicit reviewed keys. It cannot repair a wrong key, missing uniqueness rule or incorrect relationship meaning. Repeat imports must preserve logical counts and revision content. Reject conflicting content under the same revision ID. Small imports use one transaction. Larger imports use a batch_id with STAGED records and checkpoints; readers select only a published manifest. Validate the complete batch before atomically publishing its manifest pointer.

Immutable history and permissions

The application rejects edits to accepted revisions and creates new revisions for changes. Git/evidence manifests and external decision logs preserve the approved record. Neo4j does not automatically provide tamper-proof engineering history. A privileged operator can still alter the store; detect discrepancies against independently retained manifests.

For this Community prototype, agents receive only a restricted application API, no database credential and no raw Cypher execution. The importer and human publication path are separate logical capabilities enforced by application code and process/network configuration. Do not claim Enterprise fine-grained database roles, clustering or audit facilities. Shared-host administrators remain trusted operators. [W5]

A migration exports known manifests, dumps the store, restores into isolation, runs identity and query tests, then records a reviewed cutover. Failure retains the old environment. No blind in-place upgrade is planned.

Docker services readiness and resource boundaries

Docker packages the backend and its dependencies. Compose describes one local application, not a large microservice estate. Baseline host: Ubuntu 24.04 LTS amd64 with Docker Engine and Compose plugin, exact versions frozen at G1. A Windows/WSL2 deployment is a separately tested option. Unreal remains a native graphics client; Mg uses its supported environment. [W6-W7]

Service	Access and persistence	Initial resource budget
neo4j	Private backend network; /data named volume. No public Bolt/HTTP ports in normal mode; optional operator loopback binding.	2 CPU; 4 GiB limit; initial 1 GiB heap and 1 GiB page cache, adjusted after measurement.
validator/importer	Project image; read-only frozen input mount; staged-write capability through reviewed importer.	1 CPU; 1 GiB; short-lived batch job.
controller/API	Project image; allowlisted endpoints; no arbitrary shell, Cypher or host access. Optional loopback client endpoint.	2 CPU; 2 GiB; bounded agent/tool calls.
test runner	Same locked code base; isolated fixture namespace and temporary volumes. No production credentials.	1 CPU; 1 GiB; disposable execution.

Build and connection contract

Pin official base images by tag and digest; lock Python dependencies and record build inputs. Use non-root application users where supported. Do not override vendor database ownership assumptions without testing volume permissions. Set memory/CPU limits and confirm the actual Compose implementation enforces them. Cap log size and rotation; keep engineering evidence outside disposable logs.

A bounded authenticated database query is the readiness check. Compose service_healthy may delay dependent startup; application retries with backoff still handle later disconnections. A running process or open TCP socket alone is insufficient. Readiness does not prove schema initialization or accepted-baseline availability. [W6]

Credentials and network boundaries

Grant secret files only to services that need them; use the selected image’s documented bootstrap method. Compose secrets are a delivery mechanism, not automatic encrypted host storage. Keep source secret files outside Git, restrict host permissions and rotate application credentials through tested procedures. Bootstrap authentication settings do not necessarily rotate an existing store credential. [W7]

No ordinary application container mounts the Docker socket or protected research directories. Agent egress is limited to approved model-provider requests with authorized or synthetic data. The graph is not an aircraft control bus. Containers reduce dependency drift; host kernels, drivers, GPUs and external services still affect behavior and security.

Persistence backup recovery and restart

Object	What it preserves and what it does not
Image	Packaged software and build layers; it is not the live database.
Container / writable layer	A running instance and disposable changes; removal can lose unmounted data.
Named volume	Database data across container replacement when reused; volume deletion, corruption and host loss remain risks.
Consistent database dump	A database recovery artifact from an approved procedure; needs compatible restore software and a successful restore test.
Canonical export / manifests	Portable engineering records and hashes; may omit database users, configuration, indexes or operational state unless separately captured.

Proposed Community recovery procedure

Pause imports and publication. Stop the database cleanly. Use the selected version’s supported offline neo4j-admin database dump procedure with exclusive store access. Capture the dump, configuration, schema, image digest, driver/application versions, evidence manifests and source commits. Preserve required security settings separately and protect any credential material. Never treat a live file copy as a generally consistent database backup. [W8]

Store one encrypted copy on separate media or a separately controlled storage destination. Hash each file and record a manifest. Proposed retention: daily snapshots for 7 days, weekly for 4 weeks, monthly for 6 months, plus accepted milestone packages. Capacity and permissions must be confirmed before implementation. Back up referenced external evidence files too; restoring the graph alone may leave broken evidence links.

Restore and restart acceptance

Restore into a new isolated volume using compatible documented tools. Verify checksums, schema, logical identities, relationship endpoints, accepted manifests and reference-query answers. Run the clean replay tests before any cutover. Record operator, commands, time, failure log and disposition. Leave the original store untouched. Container replacement with the same volume tests persistence; an isolated restored volume tests recovery. They are separate checks.

Proposed recovery point: at most 24 hours of unaccepted work lost; accepted milestones are archived immediately. Proposed recovery time: 2 hours for the small test dataset after required images and backups are locally available. Measure both during the restore exercise. These are targets, not demonstrated service guarantees.

On restart, compare source commits, manifests, graph publication marker and backup state. Any mismatch enters HOLD until reconciled. Image pinning does not fix mutable model services or nondeterministic graphics. Recovery demonstrations use copies rather than the protected original research environment. [S2 pp9-12]

Unreal simulation scope and dynamics authority

Unreal Engine 5 will provide the BIRE scene, aircraft visual model, rotating-tail animation, scenario controls and telemetry review. The presentation explicitly places Unreal in future visualization and simulation work. This plan makes a small client a scheduled deliverable; it does not claim an existing validated Unreal BIRE model. [S3 pp26, 28]

Two simulation levels

Mandatory level: a deterministic tail-actuator/saturation software fixture and recorded trajectory replay. Unreal displays its states and provenance. This is an executable integration demonstration; a kinematic trajectory is labeled as such and does not establish aircraft stability.

Conditional research level: connect an external six-degree-of-freedom solver, preferably a documented JSBSim interface if an authorized BIRE aerodynamic model is available. A custom solver component is an alternative after effort review. The solver owns aircraft state integration and applied forces/moments. Unreal does not simultaneously integrate a second authoritative rigid body.

What a credible BIRE dynamics model needs

Inventory aerodynamic coefficient sources and validity ranges, mass/inertia, reference geometry, control definitions, trim conditions, actuator dynamics and passive-hinge behavior where applicable. Record tail rotation as a state or controlled geometry parameter and ensure its effect enters the aerodynamic model. Mesh rotation alone changes neither the solver’s coefficients nor its force application. Public USU material identifies a BIRE variant of the F-16 and related simulation research; it does not provide all required project model data here. [W9]

The flight-control diagram includes PID outer loops, LQR inner loops, gain scheduling, allocation and a safety-monitor concept. These are architecture context, not a commitment to synthesize and validate all controllers within this challenge. Any controller used in simulation requires its own version, assumptions and test evidence. AI development agents cannot command the solver’s control loop. [S3 pp23-24]

Fidelity gate and exclusions

By December 15, decide whether authorized dynamics data and an executable solver can be reproduced. If not, deliver the mandatory fixture/replay level and report incomplete aerodynamic integration. No flight hardware, operational deployment, flight-qualified runtime assurance, autonomous flight testing or high-fidelity envelope claim is included.

The client uses a simple aircraft asset with separate tail pivot and an on-screen indication of commanded angle, simulated actual angle and displayed pose. Asset geometry is visual evidence only. Record all asset licenses and coordinate conventions. Keep the scene small enough to fit the effort and hardware budgets.

Unreal interfaces time and verification

Select a supported Unreal Engine 5 patch at G1 and record it with plugin versions and project commit. Render locally on a suitable GPU host. The proposed client uses a documented project-owned message bridge; no off-the-shelf Mg-to-Unreal or Neo4j-to-flight-solver plugin is assumed.

Interface	Proposed contract
Scenario request	run_id, baseline_id, scenario/configuration hashes, solver build, seed, step size and requested duration. Acceptance returns immutable run context.
State frame	schema_version, run_id, sequence, sim_time_s, position_m, attitude quaternion with declared ordering/frame, velocity frame/units, commanded and actual tail_angle_rad, validity.
Client acknowledgment	Last consumed sequence, display time, dropped/reordered count, engine/project versions and client warning status. Display statistics are not solver verdicts.
Evidence output	Solver logs/dataset hashes, test/evaluation records and client capture references. Videos are supplementary evidence, not numerical ground truth.

Frame and unit conversion

Define solver coordinates as right-handed local North-East-Down, in meters, and body Forward-Right-Down. Unreal uses a left-handed Z-up frame; its position units default to centimeters. In a local aligned world, position maps to (100N, 100E, -100D). For attitude, convert the complete rotation matrix with the declared world/body basis transforms; do not guess Euler-angle sign changes. Validate identity and 90-degree basis cases before visualization. Verify each API’s actual units rather than assuming every physics quantity uses centimeters. [W10-W11]

Time and transport

The solver owns simulation time. Initial fixture step: 0.01 s; state publication: 50 Hz; rendering target: 30 frames/s. These are proposed workload settings, not stability or latency guarantees. Time-step sensitivity tests must justify any six-degree-of-freedom integration choice. Unreal interpolates between states and never feeds display interpolation back as measured physics.

Start with offline dataset replay. Add a local TCP or equivalent framed-message bridge only after replay passes. Include length/schema validation, sequence checks, bounded queues and reconnect behavior. Record gaps, duplicates and out-of-order frames. Stale data show an explicit warning; a disconnected display cannot silently retain a current-state claim.

Acceptance evidence

Replay a 60-second fixture with correct tail sign and declared units. Basis tests must pass, gap counters must match seeded transport faults and rendered states must be checked against logged samples. Measure target 30 fps at 1080p and p95 display lag below 100 ms on the declared hardware. Missing these targets is reported honestly. Numerical solver replay uses logged states and tolerances; bit-identical Unreal rendering across GPUs is not required.

Agents review states and assurance boundaries

Logical role	Inputs and output	Permission / completion
Requirements	Selected requirement/context → ambiguity and impact findings.	Read approved evidence; cite exact revisions or mark unresolved.
Model consistency	Supported model subset → mapping and structural findings.	No accepted-model write. Deterministic validators decide structural validity.
Software support	Reviewed impact set → staged patch and rationale.	Isolated branch only; no merge, deployment or arbitrary shell.
Verification	Requirement/patch/tests → scoped test plan and execution requests.	Allowlisted runner; report actual logs and unsupported properties.
Telemetry	Selected dataset/quality metadata → interpretation.	Read evidence; deterministic checker computes numeric verdicts.
Assurance review	Findings/manifests/logs → completeness recommendation.	No self-approval. Conflicts and missing evidence must be explicit.

Controller and permission model

A deterministic controller sets baseline, task scope, retry/time/cost limits and permitted tools. Roles may use the same model provider; they are not independent experts. Retrieval first enforces project access and baseline membership. Optional vector ranking operates only on that eligible set. Every material claim includes source revision and exact location, or an explicit unsupported status.

States are DRAFT, VALIDATED, REVIEW_PENDING, ACCEPTED, REJECTED and HOLD. Missing evidence, failed mandatory checks or conflicting applicable requirements cause HOLD. Human acceptance records proposal hash, passing check references, reviewer and rationale. Reversal creates a new baseline; it does not erase a previous accepted evaluation.

Imported text is untrusted content, not authority to execute instructions. Tool requests pass schema and permission checks. Agents cannot receive database write credentials, alter gold labels or modify approved manifests. In the solo prototype David may be both author and reviewer; that is not independent assurance. [S2 pp8-9]

Verification and formal methods

Use schema checks, graph integrity checks, pytest, boundary/property tests and application permission tests for explicit properties. A bounded TLA+ workflow model is optional: acceptance requires recorded review and mandatory checks; accepted revision contents remain unchanged. Model checking concerns the modeled state space, not automatic proof of implementation correctness.

SPIN, CBMC and other tools listed in the presentation are potential methods, not delivered proofs. Agreement among agents, passing samples or Monte Carlo testing cannot establish general safety. Runtime assurance for a real aircraft requires separately validated monitoring, control authority and timing; it is outside the agent’s development-time role. [S3 pp23-24, 26, 28]

Experiment A BIRE angle requirement change

Use DEMO-BIRE-ANGLE-001 as a notional inclusive bound on sampled actual tail rotation. Version 1 permits |delta_b| ≤25 degrees. Version 2 tightens it to ≤20 degrees. The presentation illustrates a 25-degree value, but its example does not establish an approved aircraft envelope or interchangeability of active-tail and passive-hinge limits. [S3 pp16, 18, 25; S2 p11]

Step	Action and retained evidence
1 Freeze B1	Select requirement v1, model M1, config C1 (25-degree command saturation), test T1 and synthetic S1. Record commits, hashes, units and timestamp rules.
2 Propose B2	Create requirement v2 with rationale. Traverse accepted graph traces to model constraint, configuration, unit conversion, tests, verification case and reports.
3 Stage changes	Propose M2/C2/T2 with a 20-degree command cap. Run structural, identity, boundary and permission checks on an isolated branch.
4 Reevaluate S1	Use the same dataset hash under v1 and v2. Preserve both evaluations. Do not retroactively replace the B1 result.
5 Generate S2	Run the revised saturation fixture and optionally the separately qualified solver scenario. Assign a new dataset/run ID. Do not relabel output as physical telemetry.
6 Review	A human accepts, rejects or holds B2 with test/evidence references. A failed mandatory check prevents acceptance. Retain complete audit history.

An answer that can be checked by hand

S1 is a synthetic degree-valued sequence at declared monotonic times: [0, 10, 19.9, 20, 20.1, 23, 25, 25.1, -20, -20.1, -25, -25.1]. The 23-degree sample passes v1 and fails v2. The entire sequence fails both bounds because it contains values beyond 25 degrees. This distinction avoids claiming a whole-run pass from one passing sample.

For the separate saturation fixture, y = min(L, max(-L, u)); L is the selected command limit. This checks a software command clamp, not actual actuator tracking or aircraft response. S2 is new fixture evidence. A six-degree-of-freedom result, if available, remains simulated evidence with stated validity limits.

Boundary and quality cases

Test ±20, immediately inside/outside the bound, ±25, radians conversion, NaN/Inf, missing units, invalid signal identity, duplicate/reversed timestamps and missing baseline. Invalid input produces INVALID or INDETERMINATE, never PASS. Specify numeric representation and any tolerance before running; do not silently relax the requirement. The assistant must explain every verdict using the selected version, data hash and test definition.

Experiment B game subsystem transfer

Use a simple Unreal vehicle actor in a flat test scene. The example is a game rule, not an aircraft model: DEMO-GAME-SPEED-001 limits commanded forward speed. Version 1 permits 12 m/s; version 2 permits 10 m/s. Physics-dependent actual speed, collisions and rendering are distinct observations. State the controlled variable precisely.

Executable transfer sequence

Create a small design requirement and state description, a C++ or Blueprint implementation with an explicit configuration asset, an automated rule test and a runtime log. Freeze game baseline G1. Propose the limit change in G2. Traverse graph traces to the configuration asset, movement-command logic, test cases, HUD label and replay/report artifacts. Stage the patch and retain reviewed decisions.

Evaluate synthetic commands [0, 9.9, 10, 10.1, 11, 12, 12.1] against both requirement versions. An 11 m/s command passes the older bound and fails the newer bound. Run the revised command clamp separately and retain its output as new evidence. Add pause/resume and configuration-reload tests so an old runtime asset cannot silently use the revised label.

Core reused without domain edits	Domain-specific adapter or check
Identity/revision and baseline manifests	Unreal project, asset and source identifiers; cooked/runtime configuration provenance.
Canonical validation and Neo4j publication	Mapping game requirements, movement logic and automated-test output.
Finding, citation and review contracts	Game terminology, speed units, command versus actual speed.
Agent controller and report generator	Scene launch, runtime logging and game-rule tests.
Backup, import and replay checks	Unreal build/platform capture and asset-license records.

Transfer acceptance

Require correct version-specific verdicts, no duplicate accepted revisions, citations resolving to exact game artifacts, a retained G1/G2 decision history and a reproducible fixture replay. Report adapter development hours and every core module changed. Target at least 70% reuse of declared core modules without domain-specific edits and at most 40 hours adapter work, with the module list frozen before implementation.

A visual play session supports understanding; its video alone does not prove the rule. Compare runtime configuration and logged values with automated results. Report limitations if the gameplay framework or build process prevents reliable provenance. Success would show one useful transfer; it would not establish a universal software-development assistant. [S1; S2 pp3, 12]

Evaluation graph value and agent value

Freeze the study before final runs

Prepare at least 30 requirements, 60 additional model/software/test artifacts and 120 reviewed reference links. This is a small authorized/synthetic corpus, not an aircraft completeness claim. Use separate development fixtures. Freeze 12 change tasks, 20 seeded defects, 20 clean controls and 12 conflicting-baseline questions. Keep gold labels outside agent retrieval. [S2 pp4-5, 11-12]

Condition	Purpose and fair comparison
A Manual review	Read the same artifacts and manifests without AI. Record review time and final answer quality.
B Scripted / relational	Use equivalent entity/link tables, fixed scripts and the same source information. Avoid deliberately weakening this baseline.
C Neo4j deterministic	Use the same links and checks as B. Compare query/adapter effort, correctness and review time to test graph value.
D Single assistant	Use C evidence plus one assistant and the same permitted tools.
E Specialized roles	Use the proposed controller/roles with C evidence. Compare E with C and D to isolate agent contribution.

Counterbalance order and use matched variants. David’s familiarity with authored tasks can bias results; with no independent reviewer, timing remains exploratory. Record active review separately from compute waiting, setup and retries. Run D and E three times per change task (36 outputs each); repeat baseline questions across those runs. Record provider/model IDs, prompts, settings, tools, costs and all failed attempts.

Metric	Proposed threshold and denominator
Trace / impact	Supported trace precision and recall each ≥95%; impact precision TP/(TP+FP) and recall TP/(TP+FN) each ≥90%. Report micro totals and each task; zero denominators are N/A.
Defect / false alarm	Detect ≥18/20 seeded defects with correct reason/source; false alarms ≤2/20 clean controls. Four defects each: units, stale evidence, broken traces, configuration conflict, identity/mapping.
Baseline / claims	12/12 version questions resolved correctly in every final AI repeat. Unsupported or citation-mismatched material claims ≤5% of all audited material claims; audit all final task outputs.
Effort / transfer	Target ≥20% lower median active review time for E vs C, with accepted-answer accuracy no lower. Report D too. Transfer targets are defined on p14.

Predefine accepted-answer rubrics and abstention counting. Abstention is preferable to invented evidence but is not a correctly resolved answer. A missed threshold is a negative result. Report task variation and sample limits; three AI repeats are not independent population-level trials.

Operational acceptance and release evidence

Test	Evidence required before a readiness claim
1 Clean host startup	Documented supported host; frozen images/config; stack starts without unpublished manual fixes.
2 Readiness	Authenticated query, schema checks and baseline availability; demonstrate retry after database outage.
3 Frozen import	Exact expected identities, endpoint kinds, relationships and manifest membership match reviewed fixtures.
4 Repeat import	Three imports leave logical record counts and immutable revision content unchanged.
5 Invalid input	Selected schema, units, identity and endpoint faults reject/quarantine; no invalid accepted evidence.
6 Interrupted import	Terminate a staged batch; readers never see it as published; resume or discard safely.
7 Container replacement	Replace database container on the intended volume; exact accepted reference queries remain correct.
8 Isolated restore	Restore a consistent dump into a new volume; hashes/manifests and queries pass; measure RPO/RTO.
9 Unauthorized write	Every allowlisted agent endpoint denies acceptance, raw Cypher, arbitrary shell and protected-file writes.
10 Historical retrieval	Select two baselines with conflicting requirements; no mixed-revision judgment or stale citation is accepted.
11 Version capture	Record database/driver/app/schema/image/host/Unreal/Mg builds and dependency inventory.
12 Workload measure	On the frozen corpus, measure 100 local reference queries after warm-up; target p95 <1 s excluding model calls. Record cold timing, RAM, storage and hardware.

Release gates and reproducibility

Three clean deterministic fixture replays must produce identical verdicts and canonical semantic hashes. Execution timestamps and run IDs are compared separately. Solver numerical replay uses a declared tolerance and platform; rendered pixels and language-model wording need not be bit-identical. Record any provider or dependency change as a new experiment revision.

Required gates cover provenance, invalid-input handling, baseline isolation, identity, publication permissions and recovery. A critical unresolved defect prevents a readiness claim. A limited demonstration may still be released with explicit failed gates. Meeting operational gates does not automatically meet the scientific benefit thresholds on page 15.

Evidence package: schema/mappings, fixture sources, manifests, expected answers, test definitions, raw logs, scores, environment lock, restore record, costs and limitations. No claimed test is marked passed until an execution record exists. No stack installation or benchmark execution is claimed by this planning document.

Monthly milestones effort and decisions

The proposed allocation remains 12 hours/week for about 23 weeks: 276 hours. It is an applicant-review assumption, not confirmed availability. Mandatory work is 250 hours plus 26 contingency. Unreal work is limited to one small scene and replay/fixture integration; high-fidelity aircraft modeling remains conditional. [S2 pp4-6]

Period / hours	One milestone	Done when / decision gate
Nov 16-Dec 15 50 h	G1: freeze sources, contracts and environment	Inventory rights and hardware; review supported Mg export; choose exact builds/digests; freeze schema, corpus plan and identity policy. Demonstrate minimal import; decide dynamics access and cash feasibility.
Dec 16-Jan 15 50 h	G2: deterministic BIRE evidence path	Repeat imports cleanly; retain v1/v2 evaluations; stage/publish safely; package backend and perform initial restore. Submit 2-month report by Jan 15, subject to organizer date convention.
Jan 16-Feb 15 55 h	G3: controlled agents and Unreal replay	Roles cite exact baselines; permission/hold tests pass. Native client replays fixture, converts frames and shows tail states. Freeze development prompts after exploratory ablations.
Feb 16-Mar 15 50 h	G4: transfer and external feedback	Small game adapter runs, core reuse is measured; seek 3 discovery interviews and 1 supervised evaluation. Submit 4-month report by Mar 15; report missing participation honestly.
Mar 16-Apr 23 45 h	G5: frozen evaluation and defense	Freeze final protocol by Mar 19; run comparisons; report by Apr 9; clean install/restore by Apr 16; defend Apr 19-23. No unplanned major feature build.
Across periods 26 h	Contingency	Use for documented access, integration or recovery problems. Cut optional physics/model breadth before integrity, validation or evidence controls.

Workstream reconciliation

The 250 planned hours allocate 40 to source/model contracts and Mg, 40 to graph/import integrity, 25 to containers/recovery, 35 to Unreal BIRE scene/replay, 30 to controller/agents, 25 to game transfer, 35 to evaluation/tests, and 20 to discovery/reporting/defense. These totals avoid adding simulation work without reducing other scope. Optional solver and formal-model work use only available contingency or a reviewed scope substitution.

If G1 access, time or funding fails, reduce the model subset and use explicit fixtures. If G3 quality fails, demonstrate deterministic evidence with bounded assistant output. Do not lower acceptance thresholds after seeing final results; publish a limitation or negative result instead.

Resources supply chain and project economics

All USD amounts are planning allowances, not supplier quotes. No award, institutional computing allocation or vendor assistance is assumed secured. The revised $3,000 ceiling reallocates the first plan’s reserves to account for Unreal development. [S2 p10]

Resource / official source	Cash allowance	Access and fallback
Existing workstation / applicant	0	Target 32 GB RAM, ≥100 GB free SSD, GPU with ≥8 GB VRAM for small UE scene; inventory at G1. Confirm availability, do not imply ownership.
Mg / MgPy, Mgnite [W1]	400	Verify access, extraction route and actual terms. Fixture fallback leaves real export integration incomplete.
Model-provider API	600	Development 300; evaluation 300. Approved/synthetic data only; record usage and cap requests.
Optional compute / GPU rental	450	Quote at G1; short sessions for UE if existing hardware fails. If inadequate, reduce scene complexity or revise funding gate.
Backup media / separate storage	150	Reuse suitable existing storage first; maintain independent recovery copy.
Review / evaluation assistance	450	Optional 7.5 h at assumed 60/h. No contractor or independent reviewer secured.
Unreal assets / plugins [W14]	100	Prefer a simple original/permitted asset; no paid flight-model plugin dependency.
Discovery / remote demo support	100	Remote sessions; no travel assumed.
Contingency	750	Recorded access, hardware or recovery overrun. Do not hide labor funding here.
Total cash	3,000	2,250 planned plus 750 contingency; confirm funding by Dec 15.

Python, pytest, Git, JSON Schema and local Neo4j Community carry a $0 incremental license allowance subject to actual license compliance. Linux Docker Engine is the proposed route; Docker Desktop has separate subscription conditions, especially for government use. Unreal and Mg eligibility/terms must be checked for the actual user and deployment. No free commercial entitlement is presumed. [W15-W16]

Labor and unit economics

Founder effort: 276 h × $60/h imputed value = $16,560, unpaid in this plan. Total economic prototype estimate = $19,560 including cash. Proposed pilot price: $3,000 including a $300 compute allowance; delivery cost 30 h × $60 + $300 = $2,100; contribution $900 or 30% before overhead, support, taxes and acquisition. One pilot does not recover prototype development cost.

David maintains the supplier/dependency register: version, license, price, lead time, permitted data use, alternative and restore status. Confirm cash and hardware before purchase; pause optional work at the cap. Retain portable exports if vendor access changes.

Capacity commercialization impact and risk

Capacity and accountable responsibilities

I hold an MS in Electrical Engineering (Wright State University, 2012) and a BS in Electrical Engineering (2004), and pursue Electrical and Systems Engineering doctoral work at the University of Dayton. My relevant background includes Python, embedded systems, modeling and formal-method tools; I coauthored Intelligent Autonomous Drones by Cognitive Deep Learning (Apress, 2022). These are applicant-reported qualifications. [S2 p6; S3 pp2-3, 29]

David owns architecture, implementation, fixtures, evaluation, costs and discovery. Dr. Thomas Ford is the applicant-reported doctoral advisor; challenge review is subject to consent and availability. Other names and affiliations in the historical deck are not committed challenge staff. No university, employer, government or Mgnite funding/support is represented as secured.

Customer need and adoption

Initial customer hypothesis: small aerospace/robotics R&D; groups and laboratories struggling to reconcile requirement changes with code, tests and evidence. The posted project and presentation describe that problem; commercial requirements tools confirm that traceability/change review is an established need. They do not validate demand for this prototype. [S1; S3 pp6-9; W12-W13]

Seek three interviews and one supervised evaluation by March 15 through existing contacts and TeraOpenScience introductions. Ask about actual review effort, integration constraints, buying authority and data rights. Use consented anonymized notes. Offer a tightly scoped paid pilot only after the parties agree data, ownership, acceptance criteria and price. No interview, customer or revenue is claimed completed. Assess integration services before assuming a subscription business.

Benefits with measurable limits

Reviewers could gain clearer provenance and less stale evidence; researchers could gain a reproducible benchmark; students could inspect a complete change history. Broader applications are listed on page 3. Potential safety benefit is earlier detection of evidence problems, not demonstrated accident reduction. No quantified environmental, equity or health benefit is asserted. Operational aerospace adoption requires program-specific validation, security, assurance and tool assessment.

Risk trigger	Owner response
Mg fidelity / source rights fail	David quarantines content, narrows support and discloses fixture-only limitations.
Stale AI claim / failed baseline check	HOLD; retain output, repair retrieval or validator, rerun only as a recorded experiment revision.
Hours / cost / GPU overrun	Cut optional solver, graphics detail and role breadth; preserve core integrity and evidence tests.
No external reviewer / customer	Report solo-study bias and incomplete discovery; do not invent validation.
Ownership / license ambiguity	Exclude affected material from public release and third-party model processing until resolved.

A public synthetic subset is possible after rights review. Preserve confidential/vendor data outside public releases. Confirm platform commercialization terms and institutional ownership before any license or sale.

References questionnaire coverage and confirmation

Source keys. S1: Pasted text(20261106-030252).txt. S2: 13-page bire_Blubaugh_projectplan.pdf (current identical copy ends (2)). S3: 30-page BIREPROGRAM(1).pdf (current identical copy ends (3)). S4: organizer Word template (current copy ends (2)). S5: Pasted text(20261106-031105).txt, drafting instructions only. The source text and original first plan are embedded for preservation; no missing text is reconstructed.

Official technical references checked November 5, 2026. These establish background/tool capabilities, not execution of this project. Version-dependent details are rechecked at G1.

W1 Mgnite Mg platform; W2 Mgnite Python model-access release note.
W3 Neo4j Docker deployment; W4 Neo4j constraints; W5 Neo4j authentication/authorization.
W6 Compose startup/readiness; W7 Compose secrets; W8 Neo4j offline dump procedure.
W9 USU AFRL BIRE research; W10 Epic coordinate conventions; W11 Epic measurement units.
W12 IBM DOORS Next overview; W13 Jama impact analysis.
W14 Epic licensing; W15 Docker Desktop terms; W16 Neo4j open-source licensing; W17 OMG SysML 2.0 specification.

Organizer question group	Master-plan location
Summary / posted title / name / contact	p1; the exact posted title is stated below.
New idea / alternatives / improvement / original contribution	pp3, 6, 15; measured benefits remain targets.
Build / tools / access / unknowns / tests / success criteria	pp4-16; access and fidelity gates pp5, 10, 17.
Milestones / resources / supply chain / business outline	pp17-19; customers, value, economics and market path.
Background / skills / roles / time / support	pp17-19; proposed 12 h/week and support limits.
Need / evidence / future applications / beneficiaries	pp3, 19; public sources and applicant context.
Confirmation / full name / date	Statement below; applicant reviews assumptions before submission.

Posted title: BIRE Agentic AI–MBSE: Traceable Development and Assurance for Safety-Critical Systems and Software.
Project post: TeraOpenScience project 11767. Platform category is Space Technology; the build case is aerospace aircraft research.

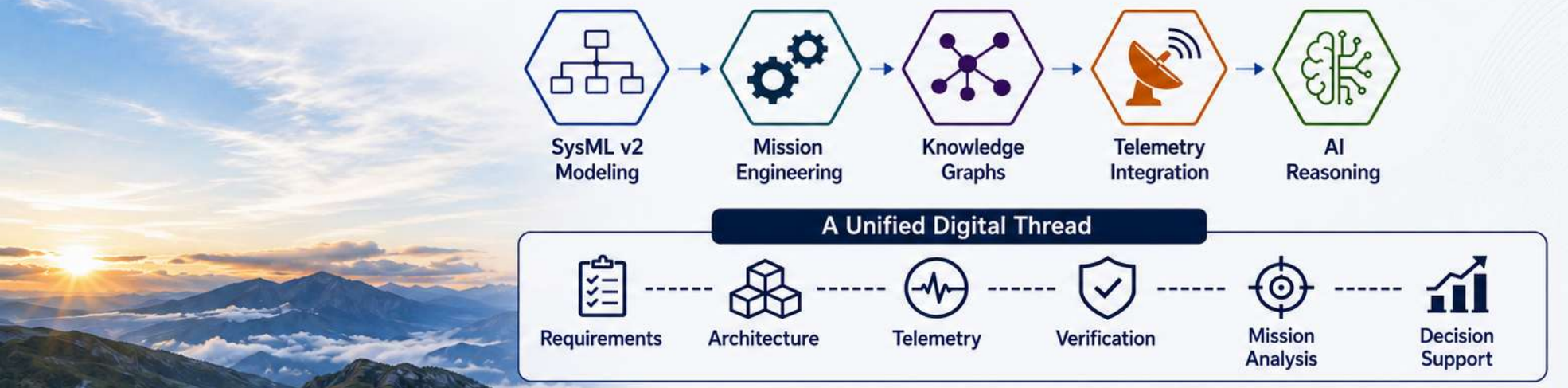
Confirmation: “All work will follow applicable university, platform, consumer-safety, consent, and privacy requirements.”
Typed name: David Allen Blubaugh. **Date:** November 5, 2026. Prepared for applicant review; no handwritten signature or submission is represented.

Deadline: Patrick’s supplied Nov 2/5 emails extend the template’s Nov 6 deadline to Saturday, Nov 7, 2026, 11:59 PM Eastern. Report dates Jan 15/Mar 15 are planning conventions to confirm. Review hours, cash, hardware, rights and collaborator availability before sending. Appendix A follows: 30 supplied pages, unchanged. Historical slide claims remain qualified by this plan.

BIRE Digital Engineering Ecosystem

SysML v2 + Mission Engineering + Knowledge Graphs + Telemetry + AI Reasoning

Current Status and Future Research Roadmap



BIRE
Bio-Inspired Rotating Empennage

Author
David A. Blubaugh
University of Dayton Ph.D. Candidate
SOCHE Student Contractor
AFRL–DAGSI Research

UNIVERSITY of DAYTON

AFIT
AIR FORCE INSTITUTE OF TECHNOLOGY

AIR FORCE RESEARCH LABORATORY

David Allen Blubaugh

Engineer • Researcher



PHD ENDEAVORS

Mission Engineering for Assured Autonomy Using SysML v2, AI, and Knowledge Graphs

- Integrating Mission Engineering, Systems Engineering, and Digital Engineering into a unified digital thread.
- Combining SysML v2, STAMP/CAST, and Formal Methods for safety, assurance, and verification.
- Developing a Digital Engineering Knowledge Graph (DEKG) to connect models, telemetry, requirements, and verification evidence.
- Focused on creating trustworthy, autonomous aerospace systems through advanced digital engineering.

WORK WITH SINCLAIR COMMUNITY COLLEGE (UNMANNED AERIAL SYSTEMS PROGRAM)

- Supporting the Unmanned Aerial Systems (UAS) program at Sinclair Community College.
- Involved in curriculum development, technical guidance, and hands-on systems integration.
- Helping students build real-world skills in UAS design, engineering, and operations.
- Committed to advancing the next generation of UAS professionals.



SINCLAIR COMMUNITY COLLEGE

KEY RESEARCH AREAS

- Knowledge Graphs & DEKG
- Runtime Assurance
- SysML v2 & Digital Thread
- AI Agents & Reasoning
- Formal Verification & Traceability

PILOT GOALS

- Actively pursuing training to become both a Remote Pilot (for UAS operations) and a Private Pilot (for manned aircraft).
- Committed to aviation safety, proficiency, and mission-ready operational skills.



Personal / Research Context

Building a Mission Engineering Digital Thread

Academic & Research Background

University of Dayton

- Ph.D. Candidate – Electrical and Systems Engineering
- Student Contractor via SOCHE
- David Blubaugh is a SOCHE PhD student

AFRL Collaboration (RQVC)

- Collaborating with Dr. Rick Graves (AFRL/RQVC) on assured autonomy and digital engineering research for future aerospace systems.

David Blubaugh's Wright State Background

- B.S. in Electrical Engineering
- M.S. in Electrical Engineering

Collaborating Organizations

AFRL – THE AIR FORCE RESEARCH LABORATORY

- Research Sponsor
- Assured Autonomy
- Digital Engineering

AFRL representative Cale Zeune provided his UAF 1.3 model of Coronet / BIRE Project.

UtahState University

- Flight Test Partner
- Telemetry Collection
- Test Operations

UNIVERSITY of DAYTON

- Ph.D. Program
- Research & Innovation
- Academic Support

Dissertation Advisor

Dr. Thomas Ford
Professor of Electrical and Computer Engineering
University of Dayton
Ph.D. Dissertation Advisor

Dissertation Co-Chair Advisors

Dr. Rick Graves
Senior Aerospace Engineer
AFRL / RQVC
Research Sponsor & Technical Lead
Ph.D. Dissertation Co-Chair

Dr. Michael Findler
Professor of Systems Engineering
Arizona State University
Ph.D. Dissertation Co-Chair

Dissertation Committee

- Dr. Thomas Ford (Advisor)
- Dr. Rick Graves (Co-Chair)
- Dr. Michael Findler (Co-Chair)
- Additional Committee Members (UD, AFIT, and ASU)

Agenda

From Mission Need to Mission Success through a Traceable Digital Thread

1	2	3	4	5	6	7	8	9
								
Mission Problem & Need	**Mission Context**	**Mission Architecture**	**Mission SoS (System of Systems)**	**Mission Engineering Approach**	**Systems Engineering Framework**	**Digital Thread & KM**	**Verification & Validation**	**BIRE as Demonstration Platform**
Define the need and objectives.	Understand the operational context.	Define actors, assets, and relationships.	Model the mission as a system of systems.	Apply DoD ME principles and pystesses.	Use SysML v2 and UAF 1.3 standards.	Enable traceability and knowledge management.	Verify models and validate performance.	Use BIRE to validate the ecosystem.



This agenda presents the end–to–end mission engineering journey—connecting mission objectives, systems engineering, verification, and knowledge to achieve **"MISSION SUCCESS."**

What is BIRE?

Bio-Inspired Rotating Empennage

BIRE uses a rotating tail assembly to provide aligned control and stability across all flight regimes, inspired by nature and evaluated for mission effectiveness.

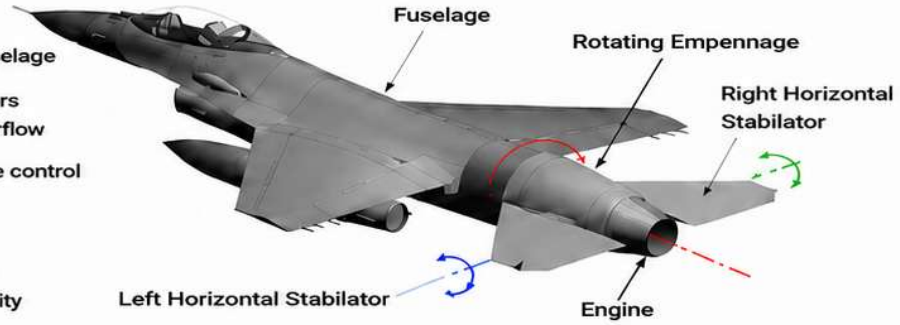
Conventional Aircraft Tail

- Fixed horizontal stabilators
- Performance varies with angle of attack and sideslip
- Control surfaces must work harder at high maneuvers



BIRE: Rotating Empennage

- Tail rotates with fuselage
- Horizontal stabilators stay aligned with airflow
- Intended to improve control authority across selected regimes
- Reduced trim drag and improved stability



Potential Advantages Under Evaluation

Maneuverability

- Maintains control authority at high angles of attack and sideslip
- Enables tighter turns and rapid attitude changes

Trim Efficiency

- Tail alignment reduces trim drag
- Less control surface deflection required

Survivability

- Lower drag and better control reduce energy loss
- Enhanced handling in degraded or hostile conditions

Control Authority

- Intended to improve control authority across selected flight regimes
- Improved responsiveness across the flight envelope

Mission Flexibility

- Supports diverse missions (refueling, escort)

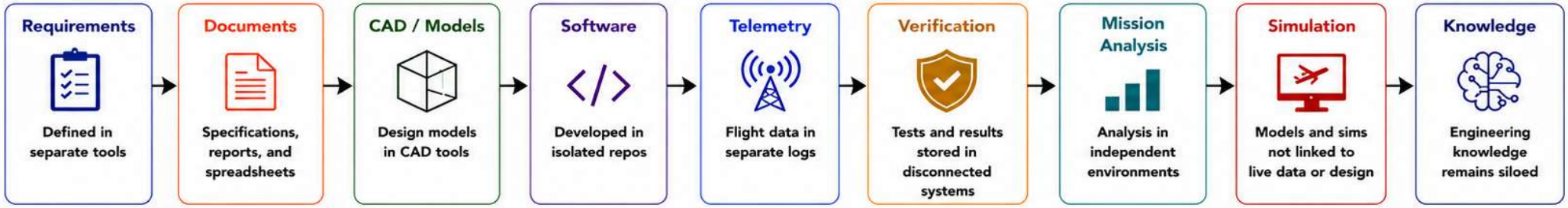
BIRE is the research case study and demonstration platform for building a Mission Engineering Digital Thread.

Research Problem

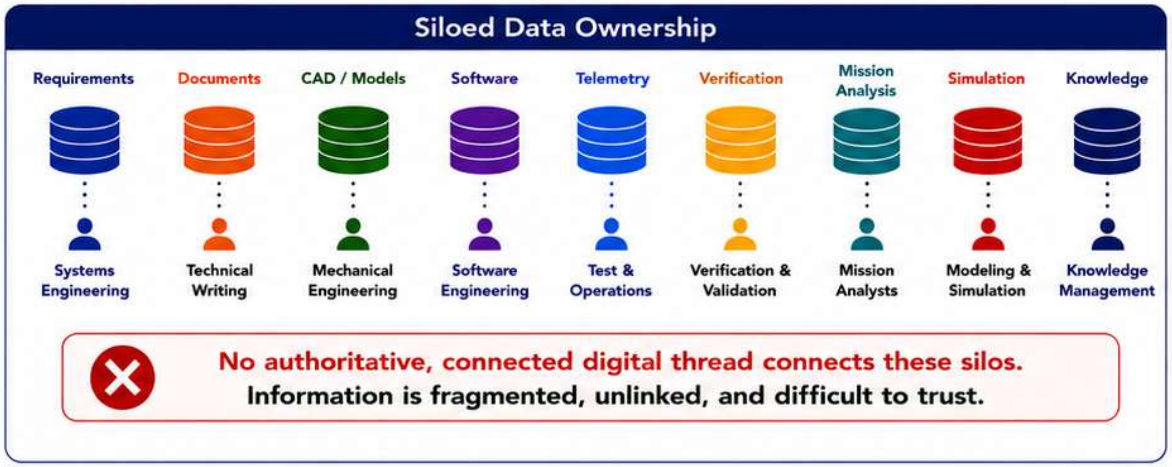
Today Engineering Information is Disconnected

Engineering data and knowledge are siloed across the lifecycle.
Information flows in fragments, not as a continuous digital thread.
No authoritative, connected digital thread.

Disjointed Engineering Information Flow (Today)



Siloed Data Ownership



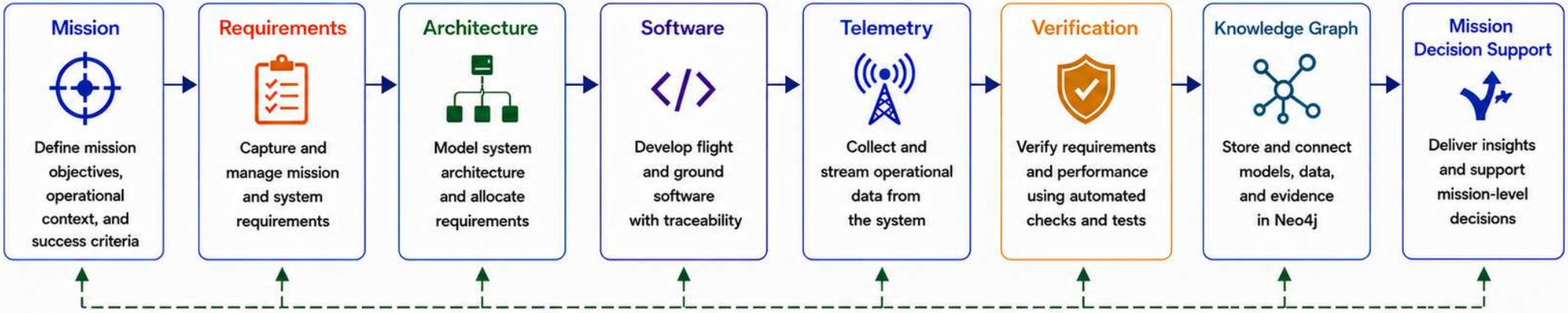
No authoritative, connected digital thread connects these silos.
Information is fragmented, unlinked, and difficult to trust.

Summer 2026 Research Objective

Create a Unified and Connected Engineering Ecosystem

Objective: Develop and demonstrate an **AI-enabled Mission Engineering** Digital Thread that connects mission objectives, models, telemetry, verification, and engineering knowledge into a **continuously traceable digital ecosystem** for **mission decision support.**

End-to-End Digital Thread (Target State)



Continuous Feedback and Traceability

Information flows both forward and backward across the lifecycle.
Changes, insights, and evidence propagate continuously.

- Every engineering artifact is connected.
- Every connection is traceable.
- Every decision is informed by evidence.

Why Mission Engineering?

Mission Engineering Aligns Engineering with Mission-Effectiveness Analysis

Mission Engineering is the disciplined approach to developing, integrating, and operating systems-of-systems to support mission-effectiveness analysis.

What is Mission Engineering?

A mission-focused approach that starts with mission objectives and drives all engineering decisions.

VS.

Traditional Engineering
Optimizes components, subsystems, or performance in isolation.

Mission Engineering
Optimizes the entire system-of-systems to support mission-effectiveness analysis in the operational context.

It's not about building the best parts.
It's about supporting mission-effectiveness analysis to inform better decisions.

Mission Engineering Flow

Mission
Define mission objectives, operational context, and success criteria.

System-of-Systems
Integrate people, processes, and technologies to accomplish the mission.

Architecture
Design the system architecture and allocate functions and resources.

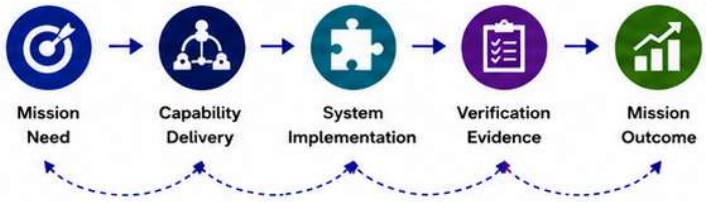
Requirements
Capture and trace capabilities, constraints, and performance needs.

Verification
Verify that the system satisfies requirements and mission needs.

Mission-Effectiveness Analysis
Assess alternatives, trade-offs, and outcomes to inform decisions.

Mission Threads — Connecting the Flow

Mission threads trace how a capability is realized from mission need to mission outcome.



Threads are traceable, measurable, and iterative across the entire lifecycle.

Key Outcomes

- **Mission Focused:** All engineering is driven by mission objectives and operational context.
- **Integrated View:** Considers the full system-of-systems and operational context.
- **Traceable & Measurable:** Mission threads provide end-to-end visibility and accountability.
- **Supports Mission-Effectiveness Analysis:** Enables assessment of alternatives and trade-offs to inform better mission decisions.

Mission Engineering Case Study

BIRE as the Operational Demonstratiom Platform

BIRE

BIRE (Bio-Inspired Rotating Empennage) provides the operational example for applying **Mission Engineering** to real aerospace missions.

Note: BIRE is the demonstration platform, not the full mission system.

Representative Mission Contexts

Aerial Refueling (Buddy / Tanker)
BIRE can operate as the receiver (buddy) or support tanking operations while not providing fuel offload.

Coronet Operations
BIRE participates in formation flight exercises to evaluate handling qualities, control authority, and autonomy.

GPS-Denied Navigation
Demonstrates navigation and autonomy in contested or degraded GPS environments.

Electronic Warfare
Operates in contested electromagnetic environments to test resilience and autonomy.

Long-Range Escort / Force Protection
Supports escort mission profiles to evaluate range, survivability, and system performance.

Example Mission Thread (One End-to-End Mission Thread)

A mission thread connects the mission need to the mission outcome through all engineering and operational elements.

Mission
Define mission objectives and operational context.

Aircraft


BIRE-enabled aircraft platform and system configuration.

Controller
Flight control and guidance algorithms execute the mission.

Telemetry
Operational data collected from aircraft sensors and systems.

Verification
Verify requirements and performance using data, tests, and analysis.

Mission Outcome
Achieve mission objectives and measure mission effectiveness.

Each link in the thread is traceable, measurable, and contributes to mission-effectiveness analysis.

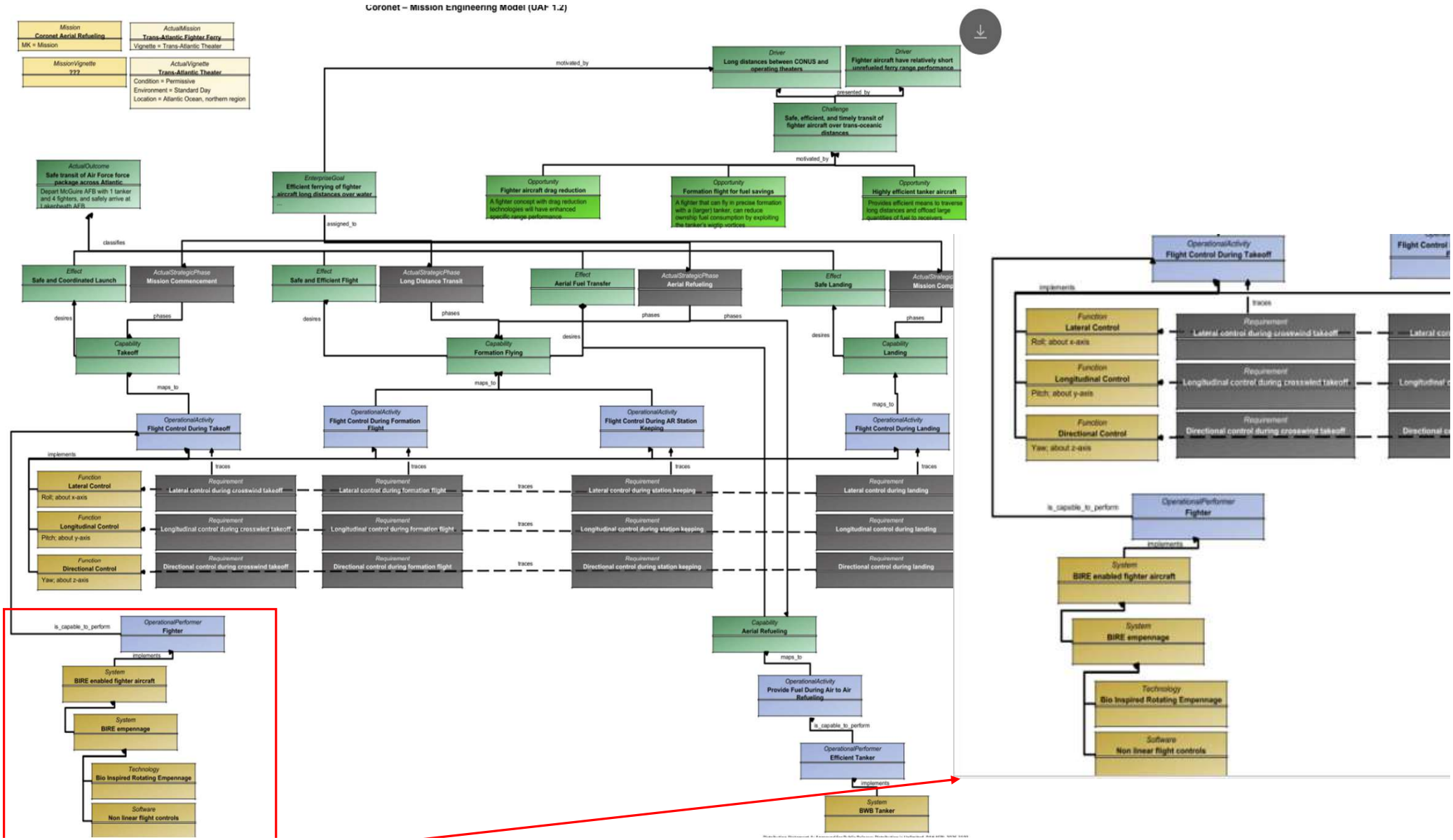
Mission Engineering in Action

By following the mission thread, we ensure that every engineering decision, design choice, and operational action is aligned with mission objectives and validated with real data.

- Mission Focused
- End-to-End Traceability
- Data Driven
- Outcome Oriented

Key Benefit

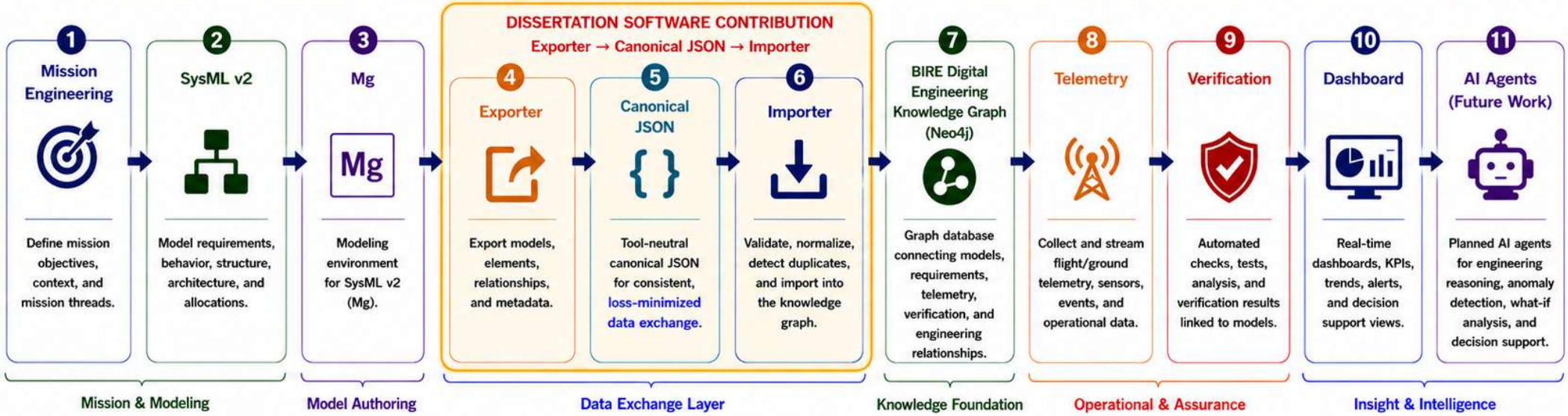
Mission Engineering drives systems-of-systems to deliver measurable mission-effectiveness analysis, not just optimized components.


Coronet – Mission Engineering Model (UAF 1.2)
Mission
Coronet Aerial Refueling
MK = Mission
ActualMission
Trans-Atlantic Fighter Ferry
Vignette = Trans-Atlantic Theater
MissionVignette
???
ActualVignette
Trans-Atlantic Theater
Condition = Permissive
Environment = Standard Day
Location = Atlantic Ocean, northern region
motivated_by
Driver
Long distances between CONUS and operating theaters
Driver
Fighter aircraft have relatively short unrefueled ferry range performance
presented_by
Challenge
Safe, efficient, and timely transit of fighter aircraft over trans-oceanic distances
motivated_by
ActualOutcome
Safe transit of Air Force force package across Atlantic
Depart McGuire AFB with 1 tanker and 4 fighters, and safely arrive at Lakenheath AFB
EnterpriseGoal
Efficient ferrying of fighter aircraft long distances over water
Opportunity
Fighter aircraft drag reduction
A fighter concept with drag reduction technologies will have enhanced specific range performance
Opportunity
Formation flight for fuel savings
A fighter that can fly in precise formation with a (larger) tanker, can reduce ownship fuel consumption by exploiting the tanker's wigtip vortices
Opportunity
Highly efficient tanker aircraft
Provides efficient means to traverse long distances and offload large quantities of fuel to receivers
assigned_to
classifies
Effect
Safe and Coordinated Launch
ActualStrategicPhase
Mission Commencement
Effect
Safe and Efficient Flight
ActualStrategicPhase
Long Distance Transit
Effect
Aerial Fuel Transfer
ActualStrategicPhase
Aerial Refueling
Effect
Safe Landing
ActualStrategicPhase
Mission Comp
desires
phases
Capability
Takeoff
Capability
Formation Flying
Capability
Landing
maps_to
OperationalActivity
Flight Control During Takeoff
OperationalActivity
Flight Control During Formation Flight
OperationalActivity
Flight Control During AR Station Keeping
OperationalActivity
Flight Control During Landing
implements
traces
Function
Lateral Control
Roll; about x-axis
Function
Longitudinal Control
Pitch; about y-axis
Function
Directional Control
Yaw; about z-axis
Requirement
Lateral control during crosswind takeoff
Requirement
Lateral control during formation flight
Requirement
Lateral control during station keeping
Requirement
Lateral control during landing
Requirement
Longitudinal control during crosswind takeoff
Requirement
Longitudinal control during formation flight
Requirement
Longitudinal control during station keeping
Requirement
Longitudinal control during landing
Requirement
Directional control during crosswind takeoff
Requirement
Directional control during formation flight
Requirement
Directional control during station keeping
Requirement
Directional control during landing
is_capable_to_perform
OperationalPerformer
Fighter
System
BIRE enabled fighter aircraft
System
BIRE empennage
Technology
Bio Inspired Rotating Empennage
Software
Non linear flight controls
Capability
Aerial Refueling
OperationalActivity
Provide Fuel During Air to Air Refueling
OperationalPerformer
Efficient Tanker
System
BWB Tanker
OperationalActivity
Flight Control During Takeoff
Function
Lateral Control
Roll; about x-axis
Function
Longitudinal Control
Pitch; about y-axis
Function
Directional Control
Yaw; about z-axis
Requirement
Lateral control during crosswind takeoff
Requirement
Longitudinal control during crosswind takeoff
Requirement
Directional control during crosswind takeoff
OperationalPerformer
Fighter
System
BIRE enabled fighter aircraft
System
BIRE empennage
Technology
Bio Inspired Rotating Empennage
Software
Non linear flight controls

Overall Digital Engineering Architecture

Master Architecture for the BIRE Digital Engineering Ecosystem

This master architecture connects mission objectives, models, data, verification, and knowledge into a unified digital ecosystem that enables **traceability**, **assurance**, and **mission-driven decisions**.



Information Flows Both Directions

→ Data and knowledge flow forward to inform decisions and operations.

← Insights and changes flow backward to refine models, requirements, and system design.

This architecture is the reference model used throughout the dissertation to ensure a **unified**, **traceable**, and **AI-enabled** digital engineering ecosystem.

Project Evolution

From Foundation to a Unified, Knowledge-Centric Engineering Ecosystem

The project evolved in a series of deliberate phases, each building capability, increasing integration, and maturing the digital engineering ecosystem toward a canonical, traceable, and persistent architecture.

Phase 1 — Repository

Establish the source control repository and project infrastructure.

Phase 2 — Neo4j

Introduce Neo4j graph database as the knowledge foundation.

Phase 3 — Telemetry

Ingest telemetry data and connect operational data to the graph.

Phase 4 — Verification

Integrate verification artifacts, tests, and results into the ecosystem.

Phase 5 — Dashboard

Build interactive dashboards for insights, KPIs, and monitoring.

Phase 6–10 — Canonical Architecture

Design and implement the canonical architecture and data exchange layer (Exporter → Canonical JSON → Importer).

★ **CORE ARCHITECTURE**
Established in this phase

Phase 11 — Traceability

Enable end-to-end traceability across requirements, models, code, data, and verification.

Phase 12 — Persistent Identity *(Planned / Next Phase)*

Planned: implement persistent identity for all artifacts, entities, and relationships.

Engineering Maturity Over Time

Each phase adds capability, reduces risk, and increases integration, resulting in a robust, scalable, and mission-ready digital engineering ecosystem.

- Foundation
- Knowledge Foundation
- Operational Data
- Assurance
- Visibility
- Architecture Consolidation
- Traceable Engineering
- Persistent Ecosystem *(Planned)*

Repository Architecture

Organized for Clarity, Modularity, and Maintainability

The repository is organized into modular directories, each with a clear purpose.
This structure supports **development, integration, verification**, and **long-term sustainability.**

REPOSITORY STRUCTURE

BIRE-Digital-Thread/

python/	Core Python source code including exporters, importers, utilities, and shared libraries.
model/	SysML v2 and related models, schemas, and model resources.
dashboard/	Dashboard application code, visualizations, and web assets.
neo4j/	Knowledge graph scripts, schemas, constraints, and database initialization.
telemetry/	Telemetry ingestion tools, parsers, data schemas, and processing pipelines.
docs/	Documentation, architecture diagrams, user guides, and design references.
LICENSE	Open-source license for the project.

Note: Only implemented components are shown.
Additional capabilities are planned for future phases.

FOLDER PURPOSES

python/
- Exporter and importer implementation
- Canonical JSON handling
- Utility functions and helpers
- Configuration and environment setup

model/
- SysML v2 models and packages
- Requirements, interfaces, and constraints
- Model libraries and reference data
- Diagrams and model artifacts

dashboard/
- Web dashboard (UI/UX)
- KPI panels and visual components
- API clients and data services
- Static assets (HTML, CSS, JS, images)

neo4j/
- Neo4j initialization and setup scripts
- Ontologies, labels, and relationships
- Constraints, indexes, and procedures
- Backup and maintenance scripts

telemetry/
- Telemetry connectors and listeners
- Data parsers and normalizers
- Message schemas and mappings
- Storage and preprocessing pipelines

docs/
- Architecture and design documentation
- User guides and tutorials
- API references and examples
- Research notes and standards

KEY BENEFITS

Modular — Separation of concerns for clean development.

Maintainable — Clear structure simplifies updates and debugging.

Scalable — Supports growth without disrupting existing modules.

Reusable — Components can be reused across projects.

Collaborative — Easy for teams to navigate and contribute.

FUTURE ENHANCEMENTS (PLANNED)

Planned future capabilities include:
- Advanced CI/CD pipelines
- Automated verification and test suites
- Extended analytics and AI-assisted tools

These capabilities are part of the ongoing project roadmap and will be integrated in upcoming phases.

Neo4j Foundation

Building a Knowledge-Centric Engineering Ecosystem

Neo4j provides the graph database foundation for a connected, semantic, and traceable digital ecosystem.
It unifies models, data, verification, and mission context into a **connected engineering knowledge layer**.

WHAT IS A KNOWLEDGE GRAPH?

A knowledge graph represents entities (nodes) and their relationships (edges) to capture context, semantics, and traceability across the entire digital engineering lifecycle.

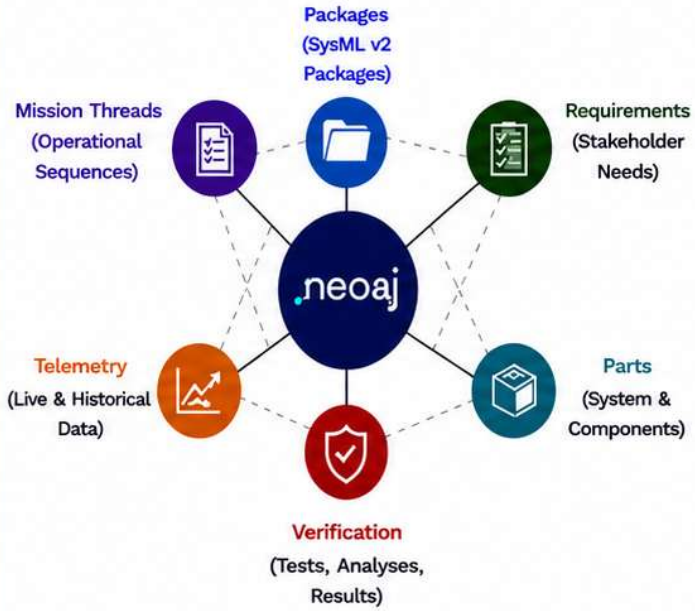
Node (Entity) —[:RELATIONSHIP_TYPE] (Edge)→ Node (Entity)

Planned Relationship Labels (Examples)

- BIREAngleLimitRequirement –[:CONSTRAINS]→ BIREMechanism
- BIRERequirement –[:SATISFIED_BY]→ BIRETestCase
- BIRERequirement –[:ALLOCATED_TO]→ BIRERole
- BIRETestCase –[:VERIFIES]→ BIRERequirement
- BIREMechanism –[:PART_OF]→ BIREAssembly
- BIRETelemetryStream –[:PROVIDES_DATA_FOR]→ BIRERequirement
- BIREMissionThread –[:HAS_SEQUENCE]→ BIREEvent

Note: Additional labels are planned for future phases.

CORE NODE CATEGORIES



WHY GRAPHS ARE POWERFUL

Captures Complex Relationships
Models many-to-many relationships that traditional databases struggle with.

Enables Semantic Queries
Answer complex questions across domains in real time.

Supports Traceability
Links mission → requirements → design → code → tests → telemetry.

Handles Time & Evolution
Stores versioned data and time-series information naturally.

Improves Insight & Decision Making
Reveals patterns, dependencies, and risks to drive better engineering decisions.

Foundation for AI & Automation
Provides structured, connected knowledge for AI agents and automation.

Telemetry Pipeline

From Flight Data to Insight and Decision Support

This pipeline transforms raw flight data into **trusted, connected knowledge** that is traceable, queryable, and actionable across engineering, verification, mission analysis, and operations.

1 Flight



Aircraft generates telemetry during flight operations.

- Sensors
- Actuators
- States
- Events

2 CSV



Telemetry is logged to CSV files on the ground system.

- Timestamped rows
- Channel columns
- Raw values
- Metadata headers

3 Python ETL



Python ETL scripts parse, clean, map, and normalize data.

- Parse CSV
- Unit conversion
- Rename channels
- Handle missing
- Type normalization

4 Validation



Data is validated for quality, completeness, and physical limits.

- Range checks
- Rate-of-change
- Completeness
- Consistency
- Anomaly flags

5 Neo4j



Validated data is stored in Neo4j as time-series nodes and relationships.

- Time entities
- Channel nodes
- Values as nodes
- Relationships
- Indexes

6 Simulation



Data is aligned with simulation results for comparison, tuning, and verification.

- Time alignment
- Metrics extraction
- Residuals
- Scenario tagging
- Model correlation

7 Telemetry Channels



Standardized channel definitions provide context and semantics.

- Channel dictionary
- Units & limits
- Hierarchy
- Descriptions
- Source mapping

8 Dashboard



Data is visualized in real time for operators, engineers, and analysts.

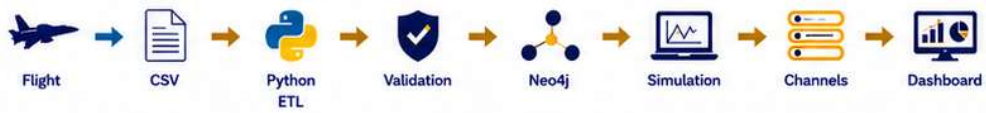
- Live plots
- KPIs & alerts
- Drill-down
- Reports
- Export

WHAT HAPPENS AT EACH STEP?							
	Sensors and systems on the aircraft produce high-rate telemetry. It is transmitted and recorded on the ground.	Data is saved in a simple, portable format for easy transfer and initial inspection.	Ensures the data is trustworthy before it enters the knowledge data. Bad data is flagged or rejected.	Data becomes part of the digital engineering ecosystem—connected, queryable, and traceable.	Enables comparison between predicted and actual behavior to support verification and model improvement.	Provides meaning and structure to every data stream so it can be used consistently across tools.	Turns data into situational awareness and engineering insight for fast, informed decisions.

KEY BENEFITS

- **Data Integrity** — Validated and trustworthy telemetry.
- **Traceability** — Every value is connected to time, source, and context.
- **Interoperability** — Standardized channels and semantics.
- **Insight** — Enables analytics, correlation, and root cause analysis.
- **Decision Support** — Empowers operators and engineers with real-time awareness.

DATA FLOW SUMMARY



Verification Pipeline

Representative Verification Workflow

This pipeline **automatically verifies** requirements against flight telemetry and simulation data, producing **objective evidence, traceable results**, and **connected knowledge**.

1 Requirement

A formal requirement with measurable criteria and thresholds.

- Unique ID
- Description
- Metric / Parameter
- Thresholds & Limits
- Verification Method

2 Telemetry

Relevant telemetry data is retrieved and aligned in time and context.

- Time-synchronized
- Channel mapping
- Units normalized
- Quality screened
- Context filtered

3 Checker

Automated checker compares data against the requirement.

- Rule evaluation
- Calculations
- Tolerance handling
- Edge cases
- Exception handling

4 PASS / FAIL

The result is determined with margins, statistics, and confidence.

- PASS or FAIL
- Margin to limit
- Confidence level
- Worst-case value
- Evaluation time

5 Verification Result

A structured result is created with evidence and traceability.

- Result ID
- Requirement ID
- Data reference
- Metrics & margins
- Timestamp & tool

6 Knowledge Graph

The result is stored in Neo4j and linked to all related entities.

- Nodes & relationships
- Bidirectional links
- Versioned history
- Queryable context
- Impact propagation

7 Closed-Loop Insight

Results drive dashboards, alerts, decisions, and continuous improvement.

- Dashboards
- Alerts / Notifications
- Trend Analysis
- Root Cause Discovery
- Requirements Update

WHAT HAPPENS AT EACH STEP?

1	2	3	4	5	6	7
The requirement defines what must be true, how it is measured, and the acceptable bounds.	The system pulls the right data for the right time window and conditions.	The system compares observed values to the requirement using defined rules and tolerances.	A formal result is recorded with all evidence, context, and traceability.	The result is connected in the knowledge graph to enable traceability and reasoning.	Insights are used to improve the system, update requirements, and support decisions.	

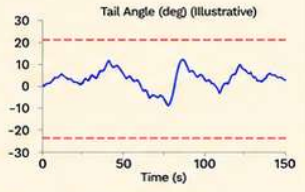
AUTOMATIC VERIFICATION CAPABILITIES (CURRENT & PLANNED)

- **Continuous Evaluation (Planned)** — Real-time verification as new data arrives.
- **Reduced Manual Inspection** — Minimizes human error and inconsistency.
- **Quantitative Evidence** — Provides margins, statistics, and confidence.
- **Scalable & Repeatable** — Works across thousands of requirements.

EXAMPLE REQUIREMENT CHECK (ILLUSTRATIVE)

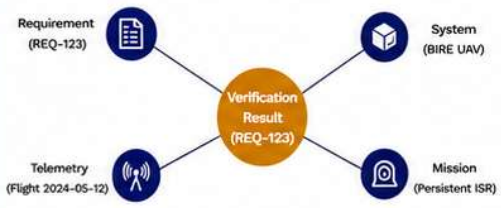
REQ-123: **BIRE Active Tail Angle Limit**

Requirement: $|\delta a| \leq 25°$
Telemetry: tail_angle_deg
Result: PASS
Worst Case: 23.7° (illustrative)
Margin: 1.3° (illustrative)
Confidence: 99.4% (illustrative)



Note: Values are illustrative unless generated by the system.

KNOWLEDGE GRAPH CONNECTION (EXAMPLE)



Legend
- Result
- Entity
- Relationship

Stored As Relationships
- RESULT_OF
- EVALUATES
- USES_DATA
- ASSOCIATED_WITH

Dashboard (Prototype)

Streamlit Dashboard Prototype – Representative Design

This Streamlit dashboard prototype provides visibility into verification results, telemetry health, and system performance—empowering engineers to identify issues and make data-driven decisions.

USER WORKFLOW

1. **Log In** — User authenticates and selects a run.
2. **Filter & Explore** — Apply filters to focus on requirements, violations, or time ranges.
3. **Analyze** — Review plots, metrics, and violation details to understand system behavior.
4. **Investigate** — Drill down into specific violations or telemetry events for root cause.
5. **Teport & Export** — Export data, generate reports, or share insights.

Role-Based Views

Engineers, Analysts, and Managers see what matters most to their mission.

BIRE Digital Thread Dashboard (Prototype)

Mock Data | Run: 2025-05-12_15-30-22 (Illustrative) | Refresh

FILTERS (Prototype)

- Run: 2025-05-12_15-30-22 (Illust.)
- Time Range: Last 2 Hours (Illustrative)
- Requirement ID: All Requirements
- Requirement Status: All
- Violation Severity: All
- Subsystem: All
- Telemetry Channel: All

Apply Filters

Clear Filters

RUN STATUS (Illustrative)

SUCCESS

Started: 2025-05-12 15:30:22 (Illust.)

Duration: 02:15:44 (Illustrative)

Data Points: 1.2M (Illustrative)

REQUIREMENTS (Illustrative)

532

Total Requirements

478	38	16
PASS	WARN	FAIL

Coverage: 94.7% (Illustrative)

VIOLATION COUNTS (Illustrative)

54

Total Violations

12	27	15
Critical	Major	Minor

Last Update: 10s ago (Illustrative)

DATA HEALTH (Illustrative)

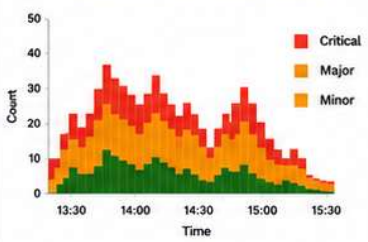
GOOD

No Data Gaps

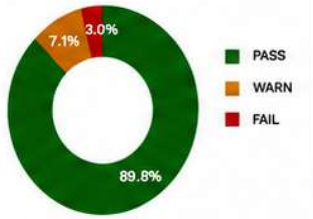
Latency: 1.2s (Illustrative)

Quality Score: 98.6% (Illustrative)

VIOLATIONS OVER TIME (Illustrative)



REQUIREMENT STATUS DISTRIBUTION (Illustrative)

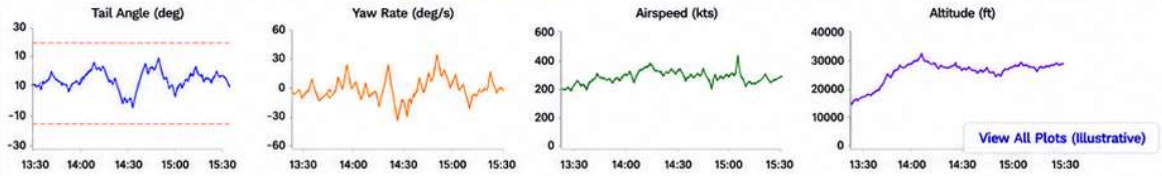


TOP VIOLATIONS (Illustrative)

Requirement ID	Description	Severity	Count
REQ-123	BIRE Tail Angle Limit	Critical	18
REQ-087	Yaw Rate Limit	Major	9
REQ-210	Roll Angle Limit	Major	7
REQ-145	Sideslip Limit	Minor	6
REQ-305	Control Surface Rate	Minor	5

View All Violations (Illustrative)

TELEMETRY PLOTS (Illustrative)



View All Plots (Illustrative)

KEY CAPABILITIES (Prototype)

- **Run Status (Illustrative)** — Overview of pipeline execution, runtime, and data quality.
- **Requirement Status (Illustrative)** — Visibility into PASS, WARN, FAIL counts and coverage.
- **Violation Counts (Illustrative)** — Quick assessment of critical, major, and minor violations.
- **Interactive Plots (Illustrative)** — Explore telemetry trends with synchronized, zoomable plots.
- **Filtering & Drill-Down (Prototype)** — Filter by time, requirement, subsystem, severity, and more.
- **Export & Reporting (Prototype)** — Export data, plots, and reports for documentation and decision making.

Why SysML v2?

The Foundation for Model-Based Systems Engineering

SysML v2 enables a precise, formal, and integrated way to model systems, requirements, behavior, interfaces, states, and allocations—creating a **single, connected model** throughout the lifecycle.

NOTE

SysML v2 is **adopted** as the MBSE foundation; tooling maturity is still **evolving.**

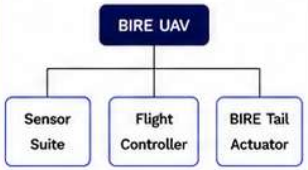
Core Modeling Concepts in SysML v2

Parts

Represent physical or logical components of the system.

Example

BIRE UAV, Flight Controller, Sensors, Actuators

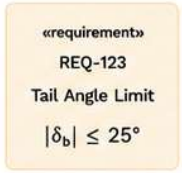


Requirements

Capture what the system must do and the constraints it must satisfy.

Example

REQ-123: Tail angle limit shall not exceed ±25°



Actions

Model operational behavior and transformations the system performs.

Example

Execute Control Law, Read Sensor, Log Telemetry

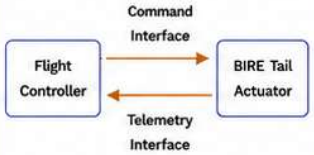


Interfaces

Define interaction points between parts and their exchanged information.

Example

Command Interface, Telemetry Interface

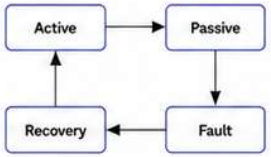


States

Capture conditions and modes the system can be in.

Example

Active, Passive, Fault, Recovery (BIRE Tail)

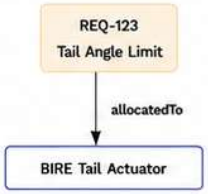


Allocations

Link requirements, behaviors, and data to the parts that realize them.

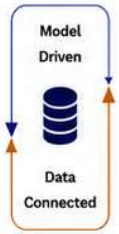
Example

Allocate REQ-123 to BIRE Tail Actuator



Why Model-Based Systems Engineering?

- Systems are too complex for document-based engineering.
- Models provide a precise, unambiguous source of truth.
- Enables early verification, impact analysis, and traceability.
- Supports simulation, analysis, and automated tools.
- Ensures consistency across the entire digital thread.

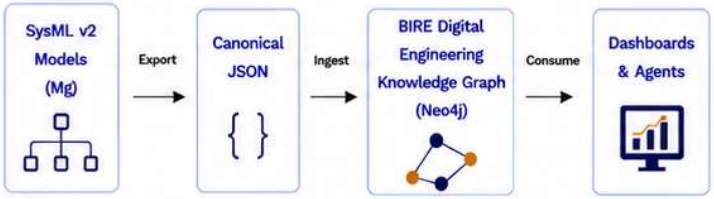


Why Mg Modeling Environment was Selected?

- Native support for SysML v2.
- Open source and extensible.
- Excellent SysML v2 diagram support.
- Model repository with Git integration.
- Lightweight, fast, and developer friendly.
- Active community and rapid evolution.

★ Mg provides the right balance of capability, openness, and performance for research and long-term use.

How It Fits in the Digital Ecosystem



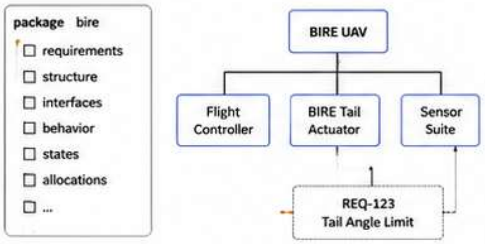
- Models are the authortvative source.
- Models are connected to data and verification.
- Enables closed-loop, knowledge-driven engineering.

Mg Export Adapter

Transforming Models into Interoperable Knowledge

The Mg Export Adapter extracts SysML v2 models and transforms them into **Canonical JSON**—
a **deterministic export format** for reliable ingestion into the BIRE Digital Engineering Knowledge Graph.

1 SysML v2 Model



Model elements include packages, parts, requirements, interfaces, actions, states, and allocations.

2 Mg Exporter

- Traverses the model using the SysML v2 API
- Resolves elements, relationships, and properties
- Applies deterministic ordering and stable IDs within the export
- Normalizes data for consistent export

Produces a complete, loss-minimized representation of the model.

3 Python Transformer

- Reads Mg export data (in-memory)
- Normalizes names, types, and units
- Expands references and resolves IDs
- Prepares data for canonical structure
- Validates schema and constraints

Ensures consistency, schema compliance, and quality.

4 Canonical JSON (Deterministic Export)

- Deterministic export format
- Order-consistent and whitespace-free
- Self-describing and schema-aligned
- Optimized for graph ingestion

Ready for ingestion into the BIRE Digital Engineering Knowledge Graph.

Why Canonical JSON is Critical

Deterministic & Reproducible	Deterministic export ensures the same input model produces the same JSON output.
Interoperable	Language- and platform-agnostic. Easily consumed by Neo4j, tools, and AI agents.
Efficient & Lightweight	Compact, structured, and optimized for transport and storage.
Traceable & Auditable	Every element, relationship, and property is explicit, queryable, and traceable.
Future-Proof	Stable schema enables long-term preservation and evolution of the digital thread.

Exporter Design Principles

1. **Complete** — Capture all relevant model elements and relationships.
2. **Deterministic** — Stable ordering, IDs, and timestamps.
3. **Loss-Minimized** — Preserve semantics and intent.
4. **Schema-Aligned** — Conform to the Canonical JSON schema.
5. **Validated** — Enforce rules and constraints before export.
6. **Extensible** — Support new model constructs and attributes.
7. **Traceable** — Maintain source references and provenance.

Canonical JSON Example (Actual Export Output)

```
{
  "source": {
    "tool": "Mg",
    "version": "0.5.2",
    "model": "BIRE",
    "exportedAt": "2025-05-12T15:30:222"
  },
  "elements": [
    {
      "id": "bire.req.REQ-123",
      "type": "Requirement",
      "name": "Tail Angle Limit",
      "text": "The BIRE tail angle shall not exceed ±25 degrees.",
      "attributes": {
        "limit_deg": 25,
        "unit": "deg"
    } }
  ],
  "relationships": [
    {
      "sourceId": "bire.req.REQ-123",
      "type": "allocatedTo",
      "targetId": "bire.part.BIRE_Tail_Actuator"
    }
} }
```

- Deterministic IDs within the export (not persistent across tools/systems).
- Explicit types and attributes.
- Relationships are fully represented.
- Provenance and traceability included.

The Mg Export Adapter ensures that SysML v2 models become **deterministic**, **interoperable**, and **knowledge-ready**—
forming a reliable bridge between design, verification, and the BIRE Digital Engineering Knowledge Graph.

NOTE: SysML v2 is adopted as the MBSE foundation; tooling maturity is still evolving.

Canonical JSON

The Interoperable Contract for Digital Engineering Data

Canonical JSON is a **deterministic**, **schema-aligned** representation of engineering knowledge.
It enables **consistent exchange**, **traceability**, and **automated reasoning** across tools and systems.

CANONICAL JSON SCHEMA OVERVIEW (CURRENT IMPORT TYPES)

PACKAGES
Organizational containers for grouping model elements.

PART DEFINITIONS
Reusable definitions of physical or logical parts.

PART USAGES
Context-specific instances of part definitions.

REQUIREMENT DEFINITIONS
Engineering requirements with attributes and text.

FUTURE TYPES (NOT CURRENTLY IMPORTED)

ACTIONS
Behavior elements and activities.

INTERFACES
Interaction points between elements.

STATES
State machine states and transitions.

ALLOCATIONS
Allocation relationships between model elements.

These types are planned for future releases.

CANONICAL JSON STRUCTURE (HIGH LEVEL)

```
{
  "schema": "bire://schemas/canonical-json/v1/schema.json",
  "model": {
    "id": "model.f2fe84a9e-9b2d-4b9c-8a7e-0d5c1f2e4001",
    "name": "BIRE UAV Model",
    "version": "0.5.2",
    "source": {
      "tool": "Mg",
      "toolVersion": "0.5.2",
      "exportedAt": "2025-05-12T15:30:22Z"
    },
  },
  "elements": [ { ...} ],
  "relationships": [ {...} ],
  "metadata": { ... },,
  "documentation": { ... }
}
```

- Schema reference (contract)
- Model metadata
- Elements (nodes)
- Relationships (edges)
- Metadata
- Documentation

EXAMPLE ELEMENT (REQUIREMENT DEFINITION)

```
{
  "id": "bire.req.REQ-123",
  "type": "RequirementDefinition",
  "name": "Tail Angle Limit",
  "text": "The BIRE tail angle shall not exceed
       ±25 degrees.",
  "attributes": {
    "limit_deg": 25,
    "unit": "deg"
  },
  "owner": "bire.pkg.Requirements",
  "createdAt": "2025-05-12T10:15:00Z",
  "createdBy": "Mg Export Adapter",
  "status": "Baseline"
}
```

EXAMPLE RELATIONSHIP (IMPLEMENTED: CONSTRAINTS)

```
{
  "id": "rel.9a7c3e11-0b4a-4c1b-9f2e-3d9c7f4b8a6e",
  "type": "CONSTRAINTS",
  "sourceId": "bire.req.REQ-123",
  "sourceType": "RequirementDefinition",
  "targetId": "bire-part.BIRE_Tail_Actuator",
  "targetType": "PartDefinition",
  "constraint": "BIRE tail actuator satisfies tail angle limit",
  "createdAt": "2025-05-12T10:16:00Z"
}
```

WHY CANONICAL JSON MATTERS

INTEROPERABLE
A single, consistent format works across tools and platforms.

DETERMINISTIC
Stable ordering, IDs, and timestamps ensure the same model always produces the same JSON.

TRACEABLE
Every element and relationship is explicitly, queryable, and auditable.

AUTOMATABLE
Machines can parse, validate, and reason without custom adapters.

SCALABLE
Structured for large models, incremental updates, and version management.

FUTURE-PROOF
Schema versioned and extensible without breaking existing integrations.

THE CONTRACT BETWEEN TOOLS

Canonical JSON is the contract that ensures every tool (modeler, verifier, simulator, analytics, dashboards, AI agents, and the knowledge graph) interprets the same data in the same way—**eliminating ambiguity, enabling automation**, and **ensuring trust**.

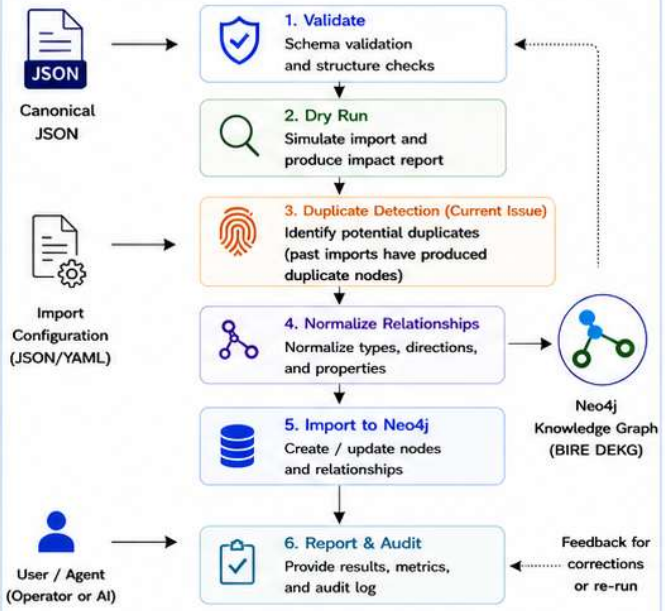
Canonical JSON provides a **stable**, **unambiguous**, and **extensible contract** that connects SysML v2 models to verification, analytics, and the **BIRE Digital Engineering Knowledge Graph**.

Importer

Reliable Ingestion into the BIRE Digital Engineering Knowledge Graph

The Importer **validates**, **normalizes**, and **ingests** Canonical JSON into Neo4j.
It ensures data quality, manages duplicates (current issue), and preserves relationships and provenance.

IMPORTER ARCHITECTURE



IMPORTER CAPABILITIES

Validation
- JSON schema validation
- Required fields and types
- Semantic rules and constraints
- Referenced IDs must exist

Dry Run
- Simulate import without writing to Neo4j
- Show counts: create / update / skip / delete
- Highlight errors and warnings
- Generate impact summary

Duplicate Detection (Current Issue)
- Detect duplicates by qualifiedName and type
- Identify nodes/relationships that would duplicate
- Report potential duplicates before import
- Note: Repeated imports have produced duplicates in Neo4j. Deduplication will be addressed in Phase 12.

Phase 12 target: MERGE by qualifiedName

```
Planned deduplication using unique constraints on
(qualifiedName, type).
```

Relationship Normalization
- Normalize type names (case, aliases)
- Enforce direction conventions (source → target)
- Normalize property names and units
- Merge equivalent relationships (within batch)

IMPORTER WORKFLOW (DETAIL)

1	Receive Input	Importer receives Canonical JSON and configuration (mappings, rules, options).
2	Validate	Perform schema and semantic validation. Reject on errors; warn on issues.
3	Dry Run	Simulate import and produce an impact report. No data is written.
4	Duplicate Detection	Identify potential duplicates. Report for review. Duplicates are not prevented (current issue). Deduplication planned for Phase 12.
5	Normalize	Normalize relationships and properties to canonical forms.
6	Import	Write to Neo4j using MERGE semantics. Create or update as appropriate.
7	Report & Audit	Return results, metrics, and audit log (what was created, updated, skipped).
8	Monitor & Feedback	Monitor health and performance. Feedback loop for corrections or re-run.

IMPORTER CONFIGURATION (EXAMPLES)

- ID mapping rules (UUID, [illegible])
- Property mappings
- Units normalization
- Required fields
- Relationship direction rules
- Namespace handling
- Import options (merge/update)
- Validation level (strict/lenient)

QUALITY GUARANTEE

Accurate
Only valid data is imported

Consistent
No duplicates within import batch (but not across runs)

Reliable
Safe imports with dry run and rollback on failure

Traceable
Full audit and provenance

KEY METRICS REPORTED

- Nodes created / updated / skipped
- Relationships created / updated / skipped
- Potential duplicates detected
- Errors / warnings
- Import duration
- Data quality score

IMPORTER TECHNOLOGY STACK

Python 3.12+

Neo4j 5.x+

Canonical JSON (Contract Format)

SysML Mapping

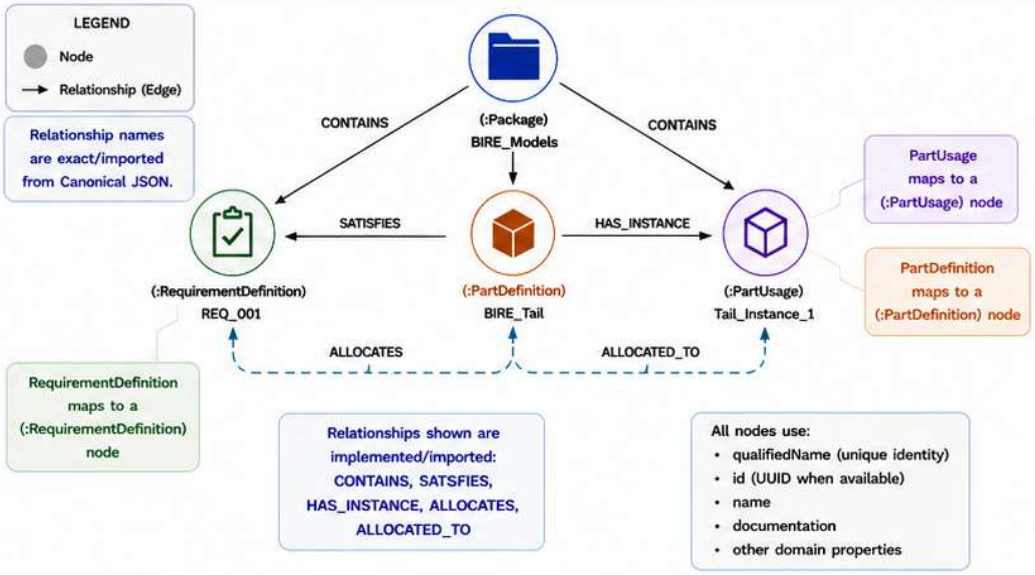
Mapping SysML v2 Concepts to Neo4j Graph Elements

SysML v2 models are exported to **Canonical JSON** and imported into **Neo4j**.
Each SysML concept is mapped to a graph pattern of **nodes** and **relationships**.

SYSML v2 TO NEO4J MAPPING

SysML v2 Concept	Description	Neo4j Label (Node)	Node / Edge	Properties / Key Fields (examples)
Package	A container that organizes model elements.	(:Package)	●	• qualifiedName • name • packageType • documentation
RequirementDefinition	A requirement that captures a need or constraint.	(:RequirementDefinition)	●	• qualifiedName • id • text • requirementType • verificationMethod
PartDefinition	A defined component or physical part in the system.	(:PartDefinition)	●	• qualifiedName • name • partType • mass • documentation
PartUsage	An instance or usage of a part within a structure.	(:PartUsage)	●	• qualifiedName • role • multiplicity • owner • partRef
Relationship	A semantic relationship between elements.	(Relationship / Edge)	→	• type • source • target • properties • context

EXAMPLE IN NEO4J GRAPH (USING IMPLEMENTED RELATIONSHIPS)



MAPPING PRINCIPLES

Deterministic Mapping
Same SysML concept always maps to the same Neo4j label and properties.

Identity Preservation
qualifiedName provides stable identity across tools and re-imports.

Relationship Fidelity
Semantic relationships preserve meaning and direction as defined in SysML v2.

WHY THIS MAPPING MATTERS

- Preserves engineering meaning and intent.
- Enables powerful queries and reasoning in Neo4j.
- Supports requirements traceability and impact analysis.
- Provides a foundation for AI agents and analytics.

EXAMPLE QUERY (CYPHER)

```
// Find all requirements allocated to a component
MATCH (req:RequirementDefinition)-[:ALLOCATES]->
(comp:PartDefinition)
RETURN req.qualifiedName,
       comp.qualifiedName,
       req.text
```

neo4j

RELATIONSHIPS (EXAMPLES)

Implemented / Imported

- CONTAINS
- SATSIFIES
- HAS_INSTANCE
- ALLOCATES
- ALLOCATED_TO
- CONSTRAINS

Planned (Future)

Additional semantic relationships will be imported in future phases (e.g., Phase 12).

Canonical Architecture

Complete BIRE Digital Engineering Ecosystem

A unified, model-based, knowledge-centric architecture that connects Mission, Engineering, Operations, and Verification through a digital thread and the BIRE Digital Engineering Knowledge Graph.

1. MISSION

- Mission Objectives
- Mission Threads
- MOEs / MOPs
- Operational Scenarios
- Environments & Threats
- CooOps



2. FLIGHT CONTROL

Outer Loops
- Altitude → Pitch
- Airspeed → Throttle
- Bank → Roll Rate
- Sideslip β → Yaw Rate

Inner Loops (LQR)
- p, q, r Rates
- δ_a (BIRE Angle)
- δ_e (BIRE Rate)

Control Laws
PID + LQR + MPC Supervision
Gain Scheduling

3. RUNTIME ASSURANCE

Envelope Protection
α, β, q, n, Airspeed, Rates, δ_a

Safety Monitor
Constraints & Limit Enforcement

Fault Detection
Actuator, Sensor, Estimator

Mode Management
Normal / Degraded / Recovery

4. TELEMETRY

Telemetry Acquisition
Sensors, Controllers, Systems

Time Synchronization
UTC / PTP

Preprocessing
Filtering, Validation, Compression

Streaming / Storage
MQTT / Kafka / Files / DB

Ingestion Pipeline
Canonical JSON Export

5. REQUIREMENTS

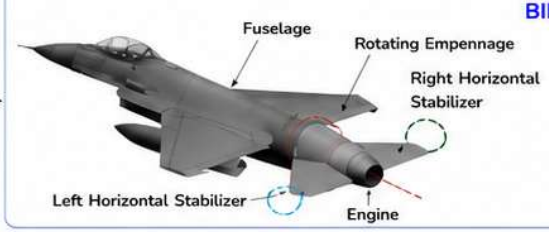
- Mission Requirements
- System Requirements
- Flight Control Requirements
- Safety Requirements
- Verification Requirements
- Traceability Relationships

Managed in SysML v2
Exported as Canonical JSON

6. SENSORS

- IMU / INS
- Air Data (Pitot / AoA / AoS)
- GPS / GNSS
- BIRE Angle Encoder
- Rate Gyros
- Health Sensors

BIRE AIRCRAFT (Physical System)



Sensors → Signal Conditioning → State Estimation → Control Computer → Actuator Drivers → Actuators

7. ACTUATORS

- BIRE Actuator (Active)
- BIRE Ringe (Passive)
- Stabilators
- Throttle
- Control Surfaces
- Servos / Drives

8. VERIFICATION (PLANNED)

- Model Verification (SysML v2)
- Simulation (6-DOF, JSBSim)
- Software Verification (CMC)
- Requirements Traceability
- Test Plans & Procedures
- Results & Evidence

Formal Verification (PLANNED)
CBMC / SPIN model checking
(Planned for Phase 12+)

- Mission
- Requirements
- Architecture (Parts)
- Flight Control
- Runtime Assurance
- Telemetry
- Sensors
- Actuators
- Verification

9. BIRE DIGITAL ENGINEERING KNOWLEDGE GRAPH (Neo4j)



Unified Graph
(Nodes + Relationships)

Graph Capabilities
- Traceability End-to-End
- Impact Analysis
- Change Propagation
- Causal Reasoning
- AI Agent Queries
- Live Operational Context

Storage
Neo4j Graph Database

Digital Thread

Every element is connected across the lifecycle:

Requirements
↕
Architecture
↕
Implementation
↕
Verification (Planned)
↕
Operations

Flight Control

Complete BIRE Flight Control Architecture

A layered, closed-loop flight-control architecture that ensures **stability**, **performance**, and **safety** for the Bio-Inspired Rotating Empennage (BIRE) aircraft.

1 — RC Pilot / Autopilot
Human commands or autonomous mission commands.

2 — Envelope Protection
Limits on airspeed, altitude, angle of attack, sideslip, rates, load factor, and actuator deflection.

3 — Trim Manager
Computes optimal trim states for steady and maneuvering flight.

4 — Runtime Assurance (Supervisory Gate)
Monitors commands and system state, then blocks or modifies unsafe commands before actuation.

5 — PID Controller (Outer Loops)
Manages slower dynamics: altitude, airspeed, bank, heading, sideslip.

6 — LQR Controller (Inner Loops)
Controls fast dynamics: p, q, r rates and BIRE angle (α_B) / rate.

7 — Gain Scheduling
Adjusts controller gains based on flight condition (airspeed, altitude, Mach, angle of attack, etc.).

8 — Control Allocation
Distributes control demands to available effectors (BIRE, stabilators, throttle, etc.).

9 — BIRE Actuator
Drives the rotating empennage (active or passive) with stabilators.

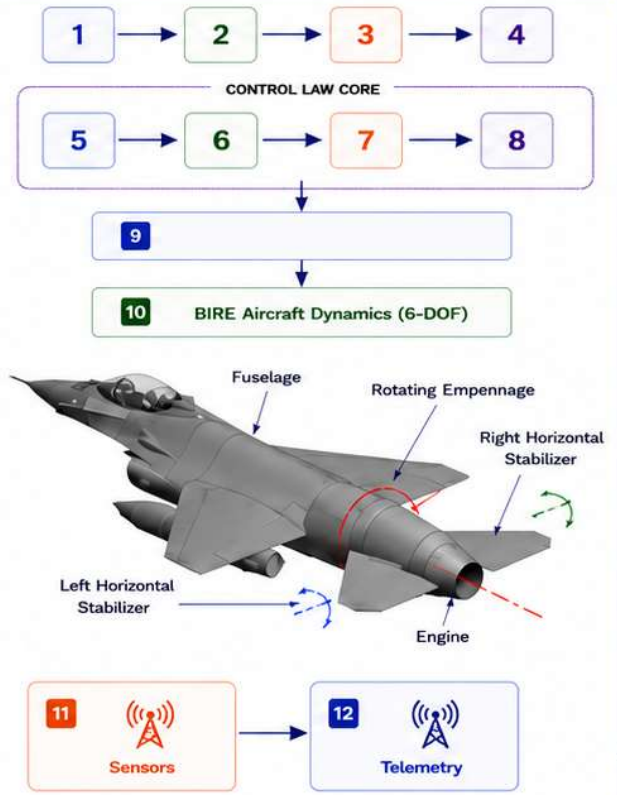
10 — BIRE Aircraft Dynamics (6-DOF)
6-DOF aircraft response to control actions and external disturbances.

11 — Sensors
IMU, GPS, air data, control surface position, engine, BIRE angle/rate, etc.

12 — Telemetry
High-rate data transmission to GCS and digital engineering ecosystem.

#	LAYER	WHAT IT DOES (BRIEF)
1	RC Pilot / Autopilot	Provides pilot inputs or autonomous guidance commands.
2	Envelope Protection	Prevents the aircraft from entering unsafe flight conditions; enforces hard and soft limits.
3	Trim Manager	Computes and maintains trim to reduce control effort and improve efficiency.
4	Runtime Assurance (Supervisory Gate)	Monitors commands and system state, then blocks or modifies unsafe commands before actuation.
5	PID (Outer Loops)	Regulates mission objectives: altitude, speed, bank, heading, and sideslip.
6	LQR (Inner Loops)	Stabilizes fast dynamics: roll rate (p), pitch rate (q), yaw rate (r), BIRE angle (α_B) and rate ($\dot{\alpha}_B$).
7	Gain Scheduling	Adapts controller gains to current flight condition for robustness and performance.
8	Control Allocation	Solves for actuator commands that achieve desired moments/forces while respecting limits.
9	BIRE Actuator	Executes commands to move the rotating empennage; includes rate limits and position feedback.
10	BIRE Aircraft Dynamics (6-DOF)	Aircraft responds to control inputs and external disturbances (wind, turbulence, maneuvers).
11	Sensors	Measure aircraft states, environment, and system health; provide feedback for control and monitoring.
12	Telemetry	Streams data to Ground Control Station and the Digital Engineering Knowledge Graph for storage, analysis, and AI.

CLOSED-LOOP FLIGHT CONTROL ARCHITECTURE



Requirements

Canonical Requirements and Traceability

Requirements **drive** everything. They flow from mission needs to telemetry and back, creating a closed-loop, **traceable** digital thread.

REQUIREMENTS FLOW (CANONICAL)

1. **Operational Requirements** — High-level mission needs and operational objectives. What the system must accomplish.
2. **SysML Requirements** — Formalized in SysML RequirementDefinition elements with unique identifiers.
3. **Architecture** — Allocated to system architecture elements (components, interfaces, functions).
4. **Verification** — Verified through analysis, simulation, test, and formal methods.
5. **Telemetry** — Execution produces data that demonstrates requirement satisfaction.
6. **Mission** — Validated performance improves mission effectiveness.

CANONICAL REQUIREMENTS HIERARCHY (EXAMPLE)

- **Operational Requirement (OR)** — Example: BIRE aircraft must remain controllable under maximum sideslip.
- **SysML Requirement (REQ)** — Example: BIRE passive hinge shall allow ±25° rotation.
  - **Architecture Allocation** — Example: Allocated to BIREMechanism passive hinge component.
  - **Verification Method** — Example: Verified via 6-DOF simulation and wind tunnel test.
  - **Telemetry Parameter** — Example: Hinge angle telemetry confirms compliance.
- **Mission Satisfaction** — Example: Maintained controllability supports mission success.

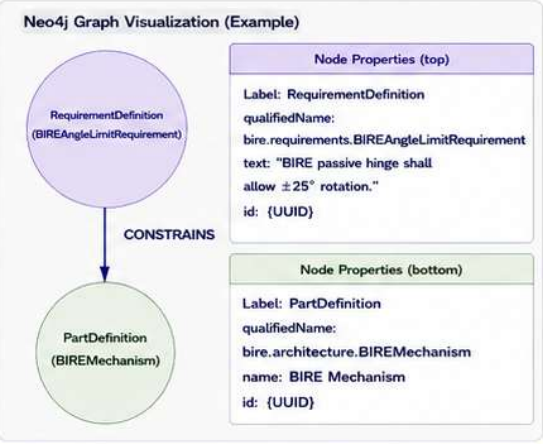
PHASE 11 TRACEABILITY WORK

Phase 11 establishes canonical traceability links between requirements and architecture using the BIRE ontology in Neo4j.

FIRST IMPLEMENTED RELATIONSHIP

BIREAngleLimitRequirement (RequirementDefinition) —CONSTRAINS→ BIREMechanism (PartDefinition)

Neo4j Graph Visualization (Example)



This demonstrates that requirements are no longer isolated text—they are linked directly to the architecture.

TRACEABILITY EXAMPLE (END-TO-END DIGITAL THREAD)

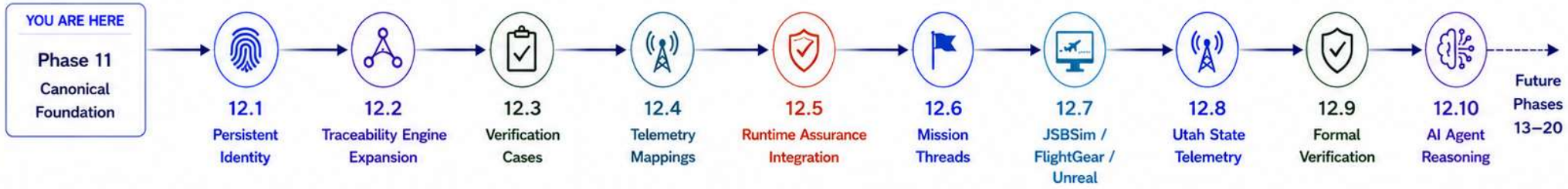
Operational Requirement — Maintain controllability in high-sideslip conditions.
→ **SysML Requirement** — BIRE passive hinge shall allow ±25° rotation.
→ **Architecture Allocation** — Allocated to BIREMechanism (passive hinge).
→ **Verification** — Verified in 6-DOF simulation and wind tunnel test.
→ **Telemetry** — Hinge angle telemetry must be within ±25°.
→ **Mission** — Ensures safe, stable, and mission-effective flight.

Technical Roadmap / Remaining Work

Phase 12–20 Research and Implementation Roadmap

Future phases (12–20) extend the canonical foundation to deliver persistent identity, end-to-end traceability, strong verification, real-time assurance, integrated mission threads, high-fidelity simulation, Utah State telemetry, formal verification, and AI agent reasoning—advancing the BIRE digital engineering ecosystem.

PHASE 12–20 ROADMAP OVERVIEW



12.1 Persistent Identity

- Global, stable IDs for all entities (requirements, architecture, tests, telemetry, artifacts).
- Identity services and registries with lifecycle management.

12.2 Traceability Engine Expansion

- Automated, end-to-end traceability across all digital thread elements.
- Impact analysis and change propagation.
- Bidirectional traceability queries at scale.

12.3 Verification Cases

- Model verification cases for all requirements.
- Link methods, tests, simulations, and evidence.
- Coverage tracking and gaps identification.

12.4 Telemetry Mappings

- Map telemetry signals to system elements and requirements.
- Units, rates, calibration, and quality metadata.
- Enable automated validation pipelines.

12.5 Runtime Assurance Integration

- Integrate Runtime Assurance as a supervisory gate in the digital thread.
- Policy authoring and enforcement traceability.
- Health monitoring, alerts, and mitigations linked to requirements.

12.6 Mission Threads

- Model missions as first-class objects.
- Link objectives, scenarios, assets, constraints, and success criteria.
- End-to-end traceability from mission to verification.

12.7 JSBSim / FlightGear / Unreal

- High-fidelity simulation integration.
- Closed-loop testing and what-if analysis.
- Scenario replay and visualization.
- Benchmarking and performance evaluation.

12.8 Utah State Telemetry

- Ingest Utah State flight test telemetry,
- Time sync, alignment, and storage in Neo4j.
- Real-world data for validation and calibration.

12.9 Formal Verification

- CBMC / SPIN model checking and proofs.
- Safety properties and invariants.
- Evidence linked to verification cases.
- Assurance case generation.

12.10 AI Agent Reasoning

- AI agents for requirements analysis, impact reasoning, and design guidance.
- Natural language queries over the knowledge graph.
- Decision support and automation.

Dissertation Central Contribution

An integrated, AI-enabled Mission Engineering Digital Thread that connects mission objectives, model-based systems engineering, operational telemetry, verification evidence, and engineering knowledge into a continuously traceable digital ecosystem. **BIRE is the validation platform; the contribution is the reusable digital engineering architecture for this ecosystem across future aerospace systems.**

TOOLS & PROGRAMMING SYSTEMS

All Tools and Software Systems Used in This Endeavor So Far

A comprehensive ecosystem supporting mission engineering, systems engineering, modeling, simulation, autonomous systems, verification, and digital engineering.

ENGINEER · MODEL · SIMULATE



MODELING & SYSTEMS ENGINEERING

- **SysML v2 Pilot Implementation** (Eclipse Modeling Tools)
- **Eclipse Modeling Tools** (EMT)
- **Acceleo** Model-to-Text / Model-to-Model Transformation (M2T/M2M)
- **Mg (Mgnite)** Modeling with SysML v2 (v0.5.2 / SysML v2 v0.4.9)

- Requirement modeling
- Structural & behavioral modeling
- Parametric modeling
- Model transformation & code generation
- MBSE & digital thread foundation

SIMULATION & FLIGHT DYNAMICS

- **JSBSim** Flight dynamics modeling & simulation
- **FlightGear** Open-source flight simulator (visualization & simulation)
- **MATLAB / Simulink** Modeling, simulation, analysis & control design

- 6-DOF aircraft dynamics
- Trim analysis
- Linearization
- Control system design
- Flight simulation & visualization

GAME ENGINE & VISUALIZATION (FUTURE)

- **Unreal Engine 5** High-fidelity visualization, real-time simulation, digital twin integration

- Real-time 3D environments
- Sensor & telemetry visualization
- Digital twin & mission rehearsal
- Human-in-the-loop simulation

PROGRAMMING LANGUAGES

- **C / C++** Embedded systems, performance-critical code
- **Python** Scripting, automation, data analysis, AI integration
- **MATLAB Language** Numerical computing, modeling, analysis
- **C#** .NET development, tooling, automation interfaces
- **Verilog / VHDL** Hardware description & FPGA design

- Embedded firmware development
- Simulation scripting
- Data processing & AI/ML workflows
- Hardware design & verification

VERSION CONTROL & DEVELOPMENT TOOLS

- **Git** Distributed version control
- **GitHub** Code hosting, collaboration, CI/CD
- **Visual Studio Code** Code editor, extensions, integrated terminal
- **GDB** Debugger
- **Qt** GUI development

- Source code management
- Team collaboration
- Debugging & testing
- Cross-platform development
- User interface development

VERIFICATION & FORMAL METHODS

- **SPIN (Promela)** Model checking (concurrency verification)
- **CBMC** Bounded model checking for C/C++
- **Frama-C** Static analysis of C code
- **ACL2** Theorem proving (formal verification)
- **Isabelle/HOL** Theorem proving (higher-order logic)
- **Coq** Proof assistant
- **nuXmv** Model checking (state-based systems)
- **KeYmaera X** Hybrid systems verification

- Formal verification
- Static & dynamic analysis
- Safety & correctness assurance
- Mathematical proof of properties

DATA, TELEMETRY & KNOWLEDGE SYSTEMS

- **Neo4j** Graph database for knowledge graphs (DEKG)
- **Python (Data Stack)** Pandas, NumPy, SciPy, Matplotlib Data analysis & visualization

- Telemetry ingestion & storage
- Knowledge graph modeling
- Data analytics & visualization
- AI/ML integration

OPERATING SYSTEMS & PLATFORMS

- **Windows** Primary development environment
- **Linux (Ubuntu)** Development, simulation, automation

- Cross-platform development
- Simulation execution
- Tool integration
- Automation & scripting

COMMUNICATION & NETWORKING

- **Wireshark** Network protocol analyzer
- **MAVLink** UAS communication protocol

- Telemetry analysis
- Data link monitoring
- UAS communication development

DOCUMENTATION & COLLABORATION

- **Confluence** Documentation & knowledge management
- **Microsoft 365** Word, PowerPoint, Excel, OneNote

- Technical documentation
- Requirements traceability
- Reporting & presentations
- Collaboration

INTEGRATING THE ECOSYSTEM

These tools work together across the digital engineering lifecycle — from requirements and modeling, through simulation and verification, to deployment, telemetry, and continuous improvement of autonomous aerospace systems.

Model → Simulate → Verify → Deploy → Monitor → Improve

What I Learned as a Summer SOCHE Intern

How It Advances My Ph.D. Research

Stronger foundation and tools for my Ph.D. research:
an integrated, traceable, AI-enabled mission engineering digital thread.

1. Mission Engineering First

- Start with mission needs.
- Define objectives.
- Align systems to effects.

Advances My Ph.D.
Builds mission-centric architectures.

2. Digital Engineering in Practice

- Model systems with SysML v2.
- Use models as authority.
- Maintain end-to-end traceability.

Advances My Ph.D.
Strengthens SysML v2 modeling and digital thread implementation.

3. Knowledge Graphs Connect Everything

- Connect requirements, design, and data.
- Enable powerful queries and reasoning.
- Visualize relationships and impact.

Advances My Ph.D.
Supports DEKG and traceability in Neo4j.

4. Telemetry and Real Data

- Collect and understand telemetry.
- Map data to requirements and models.
- Use data for evidence and insight.

Advances My Ph.D.
Enables verification, RTA integration, and telemetry mapping.

5. Verification and Assurance

- Verify models and code.
- Use formal and simulation methods.
- Ensure safety and assurance.

Advances My Ph.D.
Drives verification and runtime assurance research.

6. Collaboration and Communication

- Work in multi-discipline teams.
- Communicate clearly.
- Deliver results that matter.

Advances My Ph.D.
Improves leadership, teamwork, and impact.

Key Takeaway
Real-world experience that strengthens my Ph.D. research and prepares me to deliver a reusable digital engineering ecosystem for assured autonomy.

SysML v2 defines meaning. | Knowledge Graphs connect meaning. | Together: traceable, scalable, intelligent, mission-ready.

Why SysML v2 and Knowledge Graphs Are Critically Important for MBSE

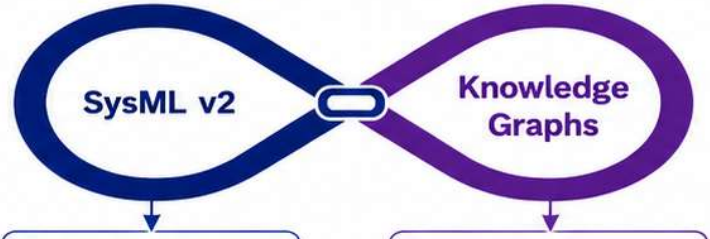
Two Foundations—Critically Dependent on Each Other

SysML v2: precise, **standardized** semantics and structure.
Knowledge Graphs: **connected**, **queryable**, **scalable** foundation.
Together: semantics come to life across the digital thread.

What SysML v2 Provides

- **Standardized Semantics**
  Common language for systems.
- **Rich, Typed Modeling**
  Precise definitions and constraints.
- **Consistency & Validation**
  Built-in rules ensure correctness.
- **Lifecycle Traceability**
  Links from needs to operation.
- **Interoperability**
  Open standard for secure exchange.

Critically Dependent on Each Other



SysML v2 gives meaning, structure, and standards.

Knowledge Graphs give connectivity, context, and computational power.

Together: a **living**, **connected**, **intelligent** ecosystem that scales from models to mission.

What Knowledge Graphs Provide

- **Global Connectivity**
  Connects all data and relationships.
- **Powerful Query & Reasoning**
  Complex queries and what-if analysis.
- **Scalability & Performance**
  Handles large, connected data.
- **Cross-Domain Integration**
  Links models, requirements, telemetry.
- **Enables AI & Automation**
  Structured context for AI/ML.

SysML v2 Alone

- Data siloed.
- Limited queries.
- Manual integration.

Knowledge Graphs Alone

- No formal semantics.
- Inconsistent data.
- Unreliable inferences.

SysML v2 defines meaning. | **Knowledge Graphs connect meaning.** | **Together: traceable, scalable, intelligent, mission-ready.**