Enabling AI-Augmented MBSE with the Model Context Protocol

Systems Engineering Capstone Project

Andrew Dunn Greg Pappas Dr. Stephen Rapp

2026-02-17

Table of contents

Chapter 1

Overview

Download as PDF View Presentation Presentation PDF

1.1 Executive Summary

This document outlines the systems engineering plan for developing an open source SysML v2 Model Context Protocol (MCP) server. The project serves dual purposes:

1. **Open Source Contribution**: Provide standalone tooling for AI-augmented Model-Based Systems Engineering (MBSE) workflows, filling a gap between commercial platforms and DIY scripting
2. **Academic Capstone**: Demonstrate INCOSE systems engineering principles [1] for a Wayne State University masters engineering capstone project

Commercial tools like SysGit provide mature, licensed Git-based MBSE platforms with comprehensive SysML v2 support. This project offers a lightweight open source alternative focused specifically on AI/LLM integration via the Model Context Protocol.

1.1.1 Key Deliverables

Software:

- **tree-sitter-sysml**: Standalone SysML v2 grammar for tree-sitter (100% training file coverage)
- **kebnf-to-tree-sitter**: Automated converter from OMG KEBNF specifications to tree-sitter grammars
- **open-mcp-sysml**: Rust MCP server with Git integration (GitLab as reference) and SysML v2 support

Publications:

- NDIA GVSETS 2026: AI-Augmented MBSE via MCP (draft Mar 23, final Jun 5, presentation Aug 11)
- INCOSE/SysEng Journal: Grammar Transposition Methodology (kebnf-to-tree-sitter)
- INCOSE 2027: SE Benchmark for LLMs (future work)

Academic:

- Capstone SE documentation (SEP, SyRS, ADD, VVP, RTM)

1.1.2 Timeline

- **Initial Research**: Early January 2026 (SysML v2 specifications and prior art)
- **Concept Phase Start**: January 12, 2026 (Week 1)
- **Capstone Delivery**: April 25, 2026 (Week 15)
- **Duration**: 15 weeks

1.1.3 Project Status (Feb 14, 2026)

Current Status

Repositories:

Repository	Status	Next Step
tree-sitter-sysml	99.6% coverage (274/275 files), 125/125 tests	Pre-release cleanup (queries, CHANGELOG)
kebnf-to-tree-sitter	640 rules parsed, 335+ conflicts	Resolve DD-001 architecture decision
open-mcp-sysml	Phase 1 Complete (5 MCP tools, 22 tests)	Execute benchmark vignettes, Phase 2 token strategies
sysml-grammar-benchmark	Scaffolded (0% functional)	Add corpus submodules, implement adapter
gvsets	~7 pages, quantitative claims unvalidated	Execute V1/V4/V5, replace TODO placeholders
capstone	SRR + PDR complete (Feb 14)	VVP prose, conclusions rewrite

Publications:

Paper	Status	Due
GVSETS 2026	Drafted, evaluation section placeholder	Draft Mar 23, Final Jun 5
Grammar Transposition	Conflict resolution in progress	Q3-Q4 2026
INCOSE 2027 Benchmark	8 vignettes defined (Section C.1)	Q3 2027

tree-sitter-sysml (Brute-Force Grammar): **PRODUCTION READY**

- 125/125 corpus tests passing
- 99.6% coverage across 275 external files (OMG, GfSE, Advent)
- Context-sensitive definition bodies implemented
- 6 language bindings (C, Rust, Go, Python, Node.js, Swift)
- Pre-release cleanup: ~18-25 hours to 1.0.0

kebnf-to-tree-sitter (Spec-Driven Grammar):

- KEBNF parser complete (640/640 rules)
- Grammar generation produces 335+ conflicts (vs 54 in brute-force)
- 4 critical path decisions open (DD-001, DD-008, DD-009, DD-020)

open-mcp-sysml (MCP Server): **PHASE 1 COMPLETE**

- 3 crates: sysml-parser, repo-client, mcp-server
- 5 MCP tools: sysml_parse, sysml_validate, sysml_list_definitions, repo_list_files, repo_get_file
- L0/L1/L2 detail levels implemented (80-97% token reduction)
- 22 tests (unit, integration, MCP protocol compliance)
- Phase 2 PRD ready: 7 token reduction strategies (including overflow detection)

sysml-grammar-benchmark:

- Repository scaffolded with PRD, CI, Quarto dashboard
- Python runner script complete, no adapters or corpora yet
- Placeholder data in dashboard

GVSETS Paper:

3-condition experiment designed with benchmark vignettes V1, V4, V5 (Section C.1):

1. **Baseline**: All files concatenated (naive)
2. **Vanilla MCP**: Simple tool calls
3. **Optimized MCP**: Cache ID + Summary pattern

Next Priority: Execute benchmark vignettes V1/V4/V5 to validate GVSETS quantitative claims

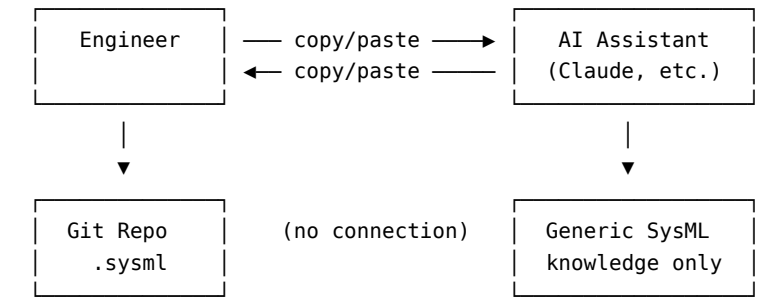
1.2 Problem Statement

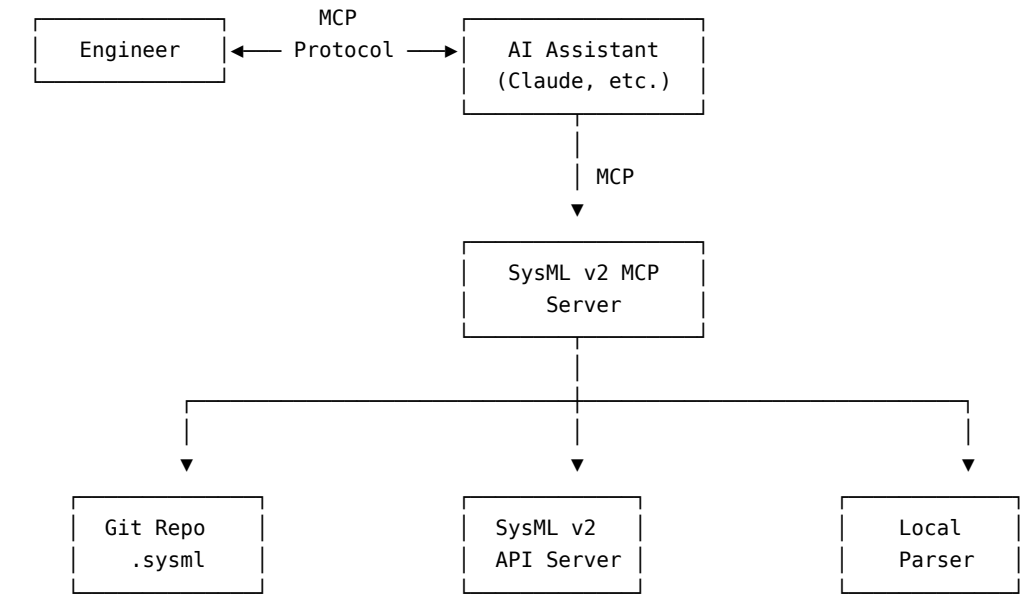
The Model Context Protocol [2] ecosystem has 75,000+ GitHub stars and 10+ official SDKs, while SysML v2 [3] achieved OMG adoption in July 2025. Yet their intersection remains unexplored. Defense and aerospace organizations need:

- Standardized AI-tool integration for MBSE workflows
- Lightweight programmatic access to SysML v2 models
- CI/CD integration for model validation
- Open source alternatives to proprietary vendor lock-in

1.3 MCP for SysML Context

The Model Context Protocol [2] standardizes how AI applications access external data and tools. An MCP server bridges AI assistants and domain-specific systems—in our case, SysML v2 models stored in Git repositories.





Without MCP	With MCP Server
AI sees pasted snippets	AI reads entire project
No model validation	Validates against SysML v2 spec
Manual copy/paste workflow	Direct Git repository integration
Generic SysML knowledge	Structured element queries
Context lost between sessions	Project state persists

This transforms the AI from a "SysML syntax helper" into an "MBSE collaborator" that understands actual project state and can take actions within it. For detailed MCP architecture and server design, see Section 4.1.

1.4 Project Objectives

1. Develop an open source MCP server for SysML v2
2. Integrate with Git providers for model persistence (GitLab as reference implementation)
3. Connect to SysML v2 API Services for validation
4. Demonstrate AI-augmented MBSE workflows using GitLab Duo
5. Publish findings at NDIA GVSETS

1.5 Central Thesis: The Harness Matters

This MCP server is one component of a larger **harness** for leveraging LLMs in MBSE workflows. The thesis of this project is that harness design—how context is selected, structured, and presented to LLMs—may matter more than raw model capability.

This thesis draws inspiration from emerging practitioner frameworks, particularly Dex Horthy's **12-Factor Agents** [4] and the concept of **Context Engineering**—the discipline of optimizing what information reaches an LLM and how it's structured. As Horthy notes: "Everything is context engineering. LLMs are stateless functions that turn inputs into outputs. To get the best outputs, you need to give them the best inputs."

OpenAI's "Harness Engineering" report [5] provides compelling industry validation: a team built a million-line production product with zero manually-written code by investing in environment design rather than direct coding. Their central finding—"the primary job of our engineering team became enabling the agents to do useful work"—directly parallels this project's thesis. Their experience with progressive disclosure ("give Codex a map, not a 1,000-page instruction manual"), repository-local knowledge stores, and mechanical enforcement of architectural invariants confirms that harness quality determines agent effectiveness, independent of model capability.

1.5.1 The Context Window Problem

Large language models exhibit measurable performance degradation when operating in the back half of their context windows. Even frontier models with 200K+ token limits show reasoning quality drops as context length increases. SysML v2 models exacerbate this:

- Enterprise systems contain thousands of elements across hundreds of files
- Naive "load everything" approaches exhaust token budgets before work begins
- Relevant elements become obscured within structural boilerplate
- Model relationships span files in ways that defeat simple truncation

A 40,000-token "project awareness" overhead (as observed in this document's own development) leaves limited budget for actual reasoning about complex models.

1.5.2 Intelligent Context Management

The value proposition extends beyond "MCP server provides model access" to **"MCP server enables selective context presentation"**:

Anti-Pattern	Harness-Aware Approach
Return entire `.sysml` files	Return specific elements by query
Dump full element hierarchies	Return element + immediate relationships
Include all metadata	Filter to semantically relevant properties
Load model into context upfront	Lazy-load via iterative tool calls
Single monolithic prompt	Decompose across agents/iterations

The parser/grammar provides the foundation: structured access to model elements. The MCP server provides the interface: tools that can be designed for minimal, targeted context injection. The harness design determines whether this pipeline produces meaningful LLM contributions or context-stuffed hallucinations.

1.5.3 Research Questions

This thesis motivates several questions addressed in the literature review (Section 3.1):

1. What **context management strategies** do existing AI+MBSE systems employ?
2. How much context is **sufficient** for meaningful LLM reasoning over models?
3. What **decomposition patterns** (multi-agent, iterative refinement) reduce per-call context?
4. How do we **measure** "meaningful performance" for LLM-MBSE interactions?

The architecture (Section 10.1) is designed to enable experimentation with these questions through configurable tool granularity and optional SysML v2 API integration for server-side query resolution.

1.5.4 Research-Plan-Implement Cycles

An emerging pattern in effective LLM agent design is the **iterative research-before-planning** approach: rather than planning upfront and executing linearly, high-quality agent workflows interleave targeted research slices with incremental planning. Each cycle:

1. **Research**: Gather just enough context relevant to the immediate decision
2. **Plan**: Make a focused plan for the next concrete step
3. **Implement**: Execute the step, capturing results
4. **Repeat**: Use implementation results to inform the next research slice

This pattern appears across multiple sources—Anthropic's "Building Effective Agents" [6] describes the **evaluator-optimizer workflow** where "one LLM call generates a response while another provides evaluation and feedback in a loop." The 12-Factor Agents framework emphasizes **small, focused agents** that "own their context window" rather than attempting monolithic operations.

For SysML v2 models, this suggests MCP tools should support incremental exploration: query a subsystem, analyze its interfaces, decide what adjacent context is needed, fetch that context, then proceed—rather than loading an entire model upfront. The grammar and parser provide the foundation for these surgical context extractions.

1.6 Scope

1.6.1 In Scope

Grammar Development (Dual-Path):

- **tree-sitter-sysml**: Brute-force grammar with 100% training file coverage
- **kebnf-to-tree-sitter**: Spec-driven grammar converter for formal traceability

MCP Server:

- **open-mcp-sysml** (Rust): Consumes tree-sitter-sysml via bindings
- Git provider file read/write operations (GitLab as reference)
- SysML v2 API client integration
- stdio and HTTP transport mechanisms
- Container deployment

Publications:

- GVSETS 2026 paper on MCP architecture
- Grammar transposition methodology paper
- SE documentation (SEP, SyRS, ADD, VVP)

1.6.2 Out of Scope (Future Work)

- **sysml.rs**: Full SysML v2 semantic analysis in Rust — a research instrument for PhD work enabling import resolution, type checking, and constraint evaluation beyond tree-sitter's syntax-only capabilities
- AI benchmarking framework (INCOSE 2027 paper topic)
- Multi-agent architectures
- Commercial integrations

1.7 Document Structure

This book contains the complete systems engineering documentation:

- **Chapter 1**: Foundation (SysML v2 background)
- **Chapter 3**: Literature Review (AI + MBSE research, prior art)
- **Chapter 4**: Model Context Protocol
- **Chapter 5**: Tooling Ecosystem
- **Chapter 6**: Systems Engineering Plan (SEP)
- **Chapter 7**: Work Breakdown Structure (WBS)
- **Chapter 8**: Stakeholder Analysis
- **Chapter 9**: System Requirements Specification (SyRS)
- **Chapter 10**: Architecture Design Description (ADD)
- **Chapter 11**: Verification & Validation Plan (VVP)
- **Chapter 12**: Implementation
- **Chapter 13**: Conclusions

Appendices include glossary, references, traceability matrix, publication strategy, and benchmark vignettes.

Chapter 2

Foundation

2.1 From Documentation to Computation

SysML v2 [3] fundamentally transforms Model-Based Systems Engineering from a documentation paradigm to a computational one. Where SysML v1 served primarily as a specification language with ambiguous semantics requiring external tools for analysis, v2 provides formal first-order logic semantics, a comprehensive expression language, and standardized APIs that enable automated verification, simulation, and design space exploration directly from models. The July 2025 OMG adoption marks the culmination of seven years of development by 80+ organizations addressing v1's core limitation: the inability to compute.

This transformation directly enables the MCP server we're building (see Section 2.7). SysML v2's textual notation means models can be stored in Git, processed by AI agents, and validated through CI/CD pipelines—the exact workflow this capstone demonstrates.

2.2 Why This Matters for AI-Augmented MBSE

MBSE has struggled with the "model-reality gap"—system architectures that couldn't be validated, simulated, or traced to requirements without extensive manual effort and custom tooling. SysML v2's formal foundation, built on the Kernel Modeling Language (KerML) [7] rather than UML, establishes precise execution semantics that tools can implement consistently.

For AI integration specifically, SysML v2 enables:

- **Textual models as code**: LLMs can read, generate, and modify SysML v2 text directly
- **Evaluable requirements**: Constraints return true/false, enabling automated verification
- **Standardized APIs**: The Systems Modeling API provides consistent programmatic access
- **Git-native workflows**: Models diff, merge, and branch like source code

This is why an MCP server for SysML v2 is tractable now when it wasn't before—the language finally supports computational interaction.

2.3 SysML v1's Inherited Limitations

SysML v1's computational limitations trace directly to its architecture as a UML profile. Three gaps prevented automation:

1. **No standardized expression language**: Practitioners had to use UML's Object Constraint Language (OCL), designed for software and ill-suited for engineering calculations with physical quantities
2. **No standardized action language**: Behavioral effects lacked specification, leaving semantics interpretation-dependent
3. **No textual control structures**: Complex behaviors required graphical syntax that "can quickly become quite large and difficult to oversee and maintain"

XMI (XML Metadata Interchange) failed in practice because "every tool supports UML differently and exports XMI differently." This interchange failure is precisely what the SysML v2 API specification addresses, and why our MCP server can rely on standardized REST endpoints rather than proprietary tool integrations.

2.4 KerML: The Formal Foundation

SysML v2's computational capability rests on KerML (Kernel Modeling Language), an entirely new application-independent foundation replacing UML dependency. KerML provides syntactic and semantic foundations through three layers:

- **Root layer**: Elements, relationships, and namespaces
- **Core layer**: Types, classifiers, and features
- **Kernel layer**: Specialized constructs

The formal semantics are specified as first-order logic (FOL), enabling mathematical precision unprecedented in systems modeling.

2.4.1 Textual Notation

The textual representation enables computational workflows:

```
part def Vehicle {
    attribute mass :> ISQBase::mass;

    part engine : Engine {
        attribute mass :> ISQBase::mass = 200 [kg];
    }

    part transmission : Transmission {
        attribute mass :> ISQBase::mass = 100 [kg];
    }

    attribute totalMass :> ISQBase::mass = engine.mass + transmission.mass;
}
```

This notation enables version control through Git, programmatic access through standard parsing, CI/CD pipeline integration, and AI-assisted modeling through LLM processing.

2.4.2 Evaluable Requirements

Requirements transform from text fields to evaluable constraints:

```
requirement def MassRequirement {
    subject vehicle : Vehicle;
    attribute massActual : ISQ::MassValue;
    attribute massLimit : ISQ::MassValue;

    require constraint { massActual <= massLimit }
}

satisfy MassRequirement by vehicle;
```

The `require` constraint evaluates to true or false. The `satisfy` relationship explicitly binds design elements to requirements, creating traceable verification.

2.5 The Systems Modeling API

The Systems Modeling API and Services specification [8] standardizes programmatic model access through REST/HTTP, replacing v1's broken XMI interchange. Tools implementing the API provide consistent endpoints

for element creation and querying, relationship navigation, version control operations (projects, branches, commits), and ad-hoc and saved queries.

This API-first architecture enables the computational workflows our MCP server exposes: automated validation, CI/CD integration, cross-tool workflows, and AI-assisted modeling.

2.6 Upstream Repositories

The SysML v2 reference implementations are hosted at github.com/Systems-Modeling:

Repository	Purpose	License
SysML-v2-Release	Latest incremental releases (start here)	LGPL-3.0
SysML-v2-Pilot-Implementation	Parser, Eclipse IDE, Jupyter kernel	LGPL-3.0 / GPL-3.0
SysML-v2-API-Services	REST/HTTP API reference server	LGPL-3.0 / GPL-3.0
SysML-v2-API-Python-Client	Generated Python client	LGPL-3.0
SysML-v2-API-Cookbook	Jupyter notebook examples	N/A

2.6.1 Parser Technology

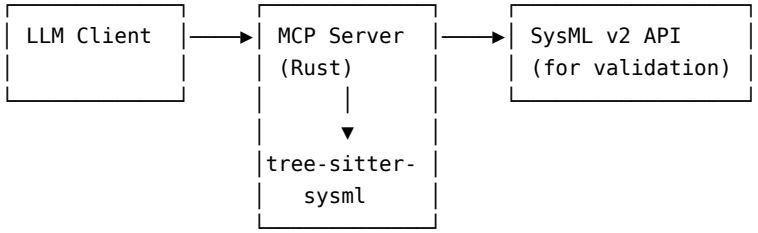
Component	Technology
Language	Java 21+ (Eclipse 2025-03)
Framework	Xtext (generates parser from grammar)
Metamodel	Eclipse EMF (Ecore)
Build	Maven with Tycho (for Eclipse plugins)

Key finding: Full SysML v2 parsing requires JVM (Xtext/EMF). Our tree-sitter grammar provides a documented subset with error recovery suitable for AI integration workflows.

2.7 Architecture Selection

Evaluated three integration options:

Option	Approach	Pros	Cons
A	Pure API Proxy	Simple, no JVM	Requires running server
B	Embedded JVM Parser	Offline, full AST	Large footprint
C	**Hybrid (selected)**	Single binary + validation	Grammar maintenance



Rationale: Balances deployment simplicity (single Rust binary) with validation capability (API server for full SysML v2 compliance). Tree-sitter grammar handles parsing with error recovery; API server handles complex semantic validation when available.

2.8 Tool Ecosystem

As of early 2026:

Commercial tools with SysML v2 support: CATIA Magic (Dassault), IBM Rhapsody, PTC Modeler, Ansys SAM, Sparx EA

Open source alternatives: Eclipse SysON (web-based graphical), Syside (VS Code extension), OMG Pilot Implementation (reference)

Key gap: No lightweight tooling exists for AI integration with SysML v2. This project fills that gap with a tree-sitter grammar and MCP server.

2.9 Industry Context

[9] argues that agile hardware engineering requires Git-based revision control for system models—not just agile tactics layered on legacy PLM tools. SysML v2's textual notation enables branching/merging workflows approximating software engineering agility.

Our position: This capstone follows INCOSE processes for academic rigor, while the MCP server aligns with the agile vision—enabling AI tools to interact with SysML models in Git repositories. Commercial tools like SysGit provide mature platforms for this workflow; our open source MCP server provides a lightweight alternative focused on AI integration.

2.10 Key Findings

1. **Parser complexity**: Full SysML v2 parsing requires JVM (Xtext/EMF). Tree-sitter provides documented subset with error recovery.
2. **API maturity**: The REST API spec is stable and well-documented. OpenAPI clients available.
3. **JSON-LD format**: All API responses use JSON-LD with `@id`, `@type` conventions.
4. **Git-native workflows**: Industry momentum toward storing SysML v2 models in Git repositories.

Chapter 3

Literature Review

3.1 Overview

This chapter surveys emerging research at the intersection of AI/LLMs and Model-Based Systems Engineering (MBSE). The literature review serves three purposes:

1. **Inform design decisions** for the MCP server architecture
2. **Validate project relevance** by identifying gaps in current tooling
3. **Position contributions** within the academic landscape

3.2 AI + SysML v2 Research

3.2.1 Li et al. (2025) - LLM-Assisted Semantic Alignment for SysML v2

Attribute	Value
Citation	[10]
Venue	IEEE ISSE 2025

Key Contribution: Proposes a prompt-driven approach for LLM-assisted semantic alignment of SysML v2 models across organizations.

Summary

7-stage iterative process with human-in-the-loop verification for cross-organizational model alignment. Key context management insights:

- **Staged decomposition**: Process split into discrete stages (preparation, extraction, matching, verification, generation, consistency check, export) with explicit user confirmation gates
- **JSON intermediate representation**: SysML v2 textual models converted to structured JSON before LLM processing
- **Confidence scoring**: All outputs include confidence metadata for verification
- **Coverage checking**: LLM explicitly confirms all elements processed to prevent silent omissions

On context windows: Paper explicitly acknowledges "attention degradation and token limits" as concerns. They balance prompt completeness vs. brevity by using detailed prompts for extraction stages, lighter prompts for validation.

MCP tool implications: Suggests tools like `extract_model_elements` → `suggest_alignments` → `verify_alignment` with JSON intermediate format.

3.2.2 Hendricks & Cicirello (2025) - Text to Model via SysML

Attribute	Value
Citation	[11]
Venue	arXiv preprint

Key Contribution: Five-step NLP pipeline converting natural language text to SysML diagrams (BDD), then to computational models via code generation.

Summary

5-step pipeline (preprocessing → knowledge graph → BDD → code → simulation) that deliberately minimizes LLM usage. Context management approach: **avoidance**.

- **LLMs used only for sentence-level attribute extraction**—never sees full documents
- Traditional NLP (TF-IDF, coreference resolution, OpenIE) handles document-level processing
- Per-sentence prompts sidestep context window limitations entirely
- Intermediate outputs at each step enable human inspection

Performance: Higher recall than GPT-4o zero-shot on key phrase extraction; end-to-end validation on simple pendulum where Copilot hallucinated parameter values.

The "harness matters" lens: Demonstrates that reasonable results are achievable by *avoiding* context management—limiting LLM to tiny, well-structured prompts.

3.2.3 Darm et al. (2025) - Inference-Time Intervention for Requirement Verification

Attribute	Value
Citation	[12]
Venue	arXiv preprint

Key Contribution: Uses intervention techniques on specific LLM attention heads to verify requirements against Capella SysML models. Achieves perfect precision on requirement fulfillment checking.

Summary

Inference-time intervention modifies 1-3 attention heads to achieve 100% precision on requirement verification. Key context management insight: **progressive context narrowing**.

Graph-based model representation:

- Capella models extracted to triple format: `|Entity| |Relation| |Entity|`
- Semantic similarity filtering finds top-k relevant components
- LLM re-ranking narrows to top-1
- Breadth-first traversal extracts adjacent components from starting point
- Chain-of-thought prompt with constrained output ("Final Answer: Yes/No")

Key finding: LLMs exhibit "overconfidence" on requirement fulfillment—they default to "yes." The intervention shifts toward conservative (higher precision) outputs.

MCP relevance: The subgraph extraction pattern (similarity → re-rank → BFS traversal) is directly applicable. The *context management* patterns transfer even when intervention techniques don't.

3.2.4 Otten et al. (2026) - Generative AI in Systems Engineering: LLM Risk Assessment

Attribute	Value
Citation	[13]

Attribute	Value
Venue	IEEE SysCon 2026

Key Contribution: Introduces LLM Risk Assessment Framework (LRF) for evaluating LLM use in systems engineering across autonomy and impact dimensions.

Summary

LLM Risk Assessment Framework (LRF): 2D matrix classifying LLM applications by **autonomy** (4 levels: Assisted → Fully Automated) and **impact** (Low/Medium/High).
Autonomy levels (inspired by SAE driving automation):

- Level 0 - Assisted: Human in charge, LLM provides support
- Level 1 - Guided: LLM suggests, human approves
- Level 2 - Supervised: AI executes under monitoring
- Level 3 - Fully Automated: AI acts independently

MCP tool classification implications:

- Query/exploration tools → Level 0-1, Low impact → Minimal risk
- Model modification tools → Level 1-2, Medium-High impact → Medium risk
- Automated design generation → Level 2-3, High impact → High risk (requires safeguards)

3.2.5 Bouamra et al. (2025) - SysTemp: Template-Based SysML v2 Generation

Attribute	Value
Citation	[14]
Venue	arXiv preprint

Key Contribution: Multi-agent system using template-based structuring to generate SysML v2 from natural language, addressing corpus scarcity and complex syntax.

Summary

Multi-agent template generator that decomposes SysML v2 generation into structured template selection and population. Context management via **template-mediated structuring**—the template constrains LLM output to valid patterns.
Architecture:

- Template generator agent identifies appropriate SysML v2 structural patterns
- Population agent fills templates from natural language specifications
- Templates encode syntactic constraints, reducing hallucination risk

Key insight: Corpus scarcity is the central obstacle for SysML v2 generation. Templates compensate by providing structural scaffolding that LLMs cannot learn from limited examples.
MCP relevance: Template-based approach aligns with tool-mediated generation. An MCP server providing parse/validate feedback enables the same constraint-first pattern without hardcoded templates. Validates our grammar-first architecture: structural knowledge must come from tooling, not LLM training data.

3.2.6 Jin et al. (2025) - SysMBench: Benchmarking LLMs on System Model Generation

Attribute	Value
Citation	[15]
Venue	arXiv preprint

Key Contribution: First benchmark for evaluating LLM-generated system models. 151 curated scenarios across 17 LLMs demonstrate that raw LLM capability is insufficient—best BLEU score is 4%.

Summary

151 human-curated scenarios spanning multiple domains and difficulty levels, evaluated with SysMEval (semantic-aware metric) and traditional metrics (BLEU, CodeBLEU). Strongest quantitative evidence that LLMs cannot reliably generate system models without external support.
Benchmark design:

- Each scenario: NL requirements → model description language → visualized diagram
- Evaluation: BLEU, CodeBLEU, and custom SysMEval-F1 metric
- Three enhancement strategies tested: direct prompting, few-shot, chain-of-thought

Key findings:

- Best BLEU: 4% (across all 17 LLMs tested)
- Best SysMEval-F1: 62%
- Enhancement strategies provide marginal improvement
- Model description language syntax is a primary failure mode

MCP relevance: Provides the strongest quantitative argument for our thesis—harness design matters more than model capability. If the best LLMs achieve only 4% BLEU on system models, external tooling (parsing, validation, structural feedback) is not optional but essential. Validates the need for grammar-aware MCP tools that provide syntactic scaffolding.

3.3 AI + UML/General Modeling

3.3.1 Giannouris & Ananiadou (2025) - NOMAD: Multi-Agent UML Generation

Attribute	Value
Citation	[16]
Venue	arXiv preprint

Key Contribution: Cognitively-inspired multi-agent framework decomposing UML class diagram generation into entity extraction, relationship classification, and diagram synthesis.

Summary

Cognitively-inspired multi-agent framework decomposing UML generation into specialized subtasks. Context management via **pipeline partitioning**—each agent sees only what it needs.
Agent architecture:

Agent	Input	Output
Concept Extractor	NL requirements	Classes + attributes
Relationship Comprehender	Requirements + entities	Typed relationships
Model Integrator	Entities + relationships	JSON intermediate
Code Articulator	JSON	PlantUML

Key insight: JSON intermediate representation acts as "context checkpoint"—formalizing output before next stage removes NL ambiguity.
Performance: F1 improves 0.66 → 0.70 (breadth), 0.74 → 0.84 (depth). Relationship modeling sees largest gains (0.52 → 0.92).
MCP relevance: Pipeline pattern with intermediate representations directly applicable. Schema constraints align with MCP tool design.

3.3.2 Ferrari et al. (2024) - Model Generation with LLMs: Requirements to UML

Attribute	Value
Citation	[17]
Venue	arXiv preprint

Key Contribution: Evaluates ChatGPT generating UML sequence diagrams from 28 requirements documents. Identifies challenges with requirements smells.

Summary

First systematic study of GPT-3.5 generating UML sequence diagrams from 28 real-world requirements documents (87 variants). Qualitative analysis by 3 experts identified 23 categories of issues.
Performance (5-point scale, mean=3):

- Standard adherence: 4.54
- Terminological alignment: 4.49
- Understandability: 4.37
- Completeness: 3.63
- **Correctness: 3.22** (NOT significantly above mean—critical weakness)

Key failure modes:

- Requirements smells (ambiguity/inconsistency) cause LLM to "hide" conflicts by abstracting
- Session memory pollution causes hallucinations
- Cross-reference handling fails when context unavailable

MCP implications: Single-shot generation inadequate; need iterative refinement. Session isolation essential.

3.4 Foundational Industry Work

3.4.1 Bader et al. (2024) - User-Centric MBSE Using Generative AI

Attribute	Value
Citation	[18]
Venue	MODELSWARD 2024

Key Contribution: LLM-assisted model understanding patterns. Explicitly recommends SysML v2 as more LLM-friendly than XMI.

Summary

Fine-tuned GPT-3.5 on UML component diagrams (XMI format) to generate models from natural language. Identifies three critical obstacles and **explicitly recommends SysML v2** as more LLM-friendly.
Critical findings:

- Context window severely limiting: 16K tokens exceeded generating ~8 elements
- XMI verbosity exacerbates context issues (~30% reduction via pre-processing still insufficient)
- Moving context window prevents referencing past elements

SysML v2 validation: Paper explicitly recommends SysML v2's textual notation as superior for LLM interaction—validates our project direction.
MCP implications: Context chunking critical even for simple models. Server should abstract/manage element IDs. Post-processing required—implement syntax checking.

3.4.2 Neema et al. (2025) - Evaluating Engineering AGI

Attribute	Value
Citation	[19]
Venue	arXiv preprint

Key Contribution: Bloom's taxonomy-based framework for evaluating engineering AGI capabilities, including CAD/SysML model evaluation criteria.

Summary

Bloom's taxonomy-based framework for evaluating engineering AGI agents across cognitive levels from factual recall to meta-reasoning.
6 cognitive levels:

1. Remember - Factual recall (equations, standards)
2. Understand - Interpret design structure/function
3. Apply - Predict performance, invoke simulation tools
4. Analyze - Complete partial designs, detect errors
5. Create - Synthesize full designs from requirements
6. Reflect - Critique decisions, recognize limitations

MCP relevance:

- Level 3+ expects agents to "invoke external tools such as solvers, simulators"—aligns with MCP tool-calling
- Structured artifact I/O assumes automated validation—MCP could provide these tools
- Metadata-driven test generation enables domain-specific benchmarks

3.4.3 Lopopolo (2026) - Harness Engineering: Agent-First Software Development

Attribute	Value
Citation	[5]
Venue	OpenAI Engineering Blog

Key Contribution: Documents building a million-line production product with zero manually-written code, identifying environment design as the primary engineering activity in agent-first workflows.

Summary

Five-month experiment building an internal beta product entirely via Codex agents (~1,500 PRs, 3-7 engineers). Key findings directly relevant to MCP server design:

- **Progressive disclosure over monolithic context**: "One big AGENTS.md" failed; replaced with structured `docs/` directory where a short map (~100 lines) points to deeper sources of truth. Validates L0/L1/L2 tiered loading patterns
- **Repository knowledge as system of record**: Context that lives outside the repository (chat threads, documents, tribal knowledge) is effectively invisible to agents. All architectural decisions, plans, and quality standards must be versioned and co-located with code
- **Mechanical enforcement**: Custom linters and structural tests enforce architectural invariants (dependency directions, naming conventions, file size limits). Error messages inject remediation instructions into agent context
- **"Golden principles" and garbage collection**: Recurring agent tasks scan for pattern drift, update quality grades, and open targeted refactoring PRs—continuous entropy management rather than periodic cleanup

On throughput: Average 3.5 PRs per engineer per day. Agent runs regularly exceed 6 hours on single tasks. Minimal blocking merge gates because corrections are cheap at high throughput.
MCP relevance: The article's framing of "harness engineering" as a discipline—designing environments, feedback loops, and control systems for agents—validates this project's central thesis (Section 1.5). Their

progressive disclosure patterns map directly to our L0/L1/L2 context budgeting approach. Their observation that "boring technologies" are easier for agents to model supports our tree-sitter and MCP protocol choices.

3.5 Additional References

Paper	Key Finding	Domain
Jerry et al. (2026) [20]	SysML v2 as semantic backbone for WCAG-compliant UI generation	Healthcare
Erikstad (2024) [21]	CrewAI multi-agent + MBSE for ship design optimization	Marine
Rouabhia & Hadjadj (2025) [22]	9-LLM benchmark on UML method generation; 100% syntactic validity	Benchmarking
Mao et al. (2025) [23]	Data dependency inference improves code gen +11.66%	Code generation
Trendowicz et al. (2026) [24]	GPT-4o requirements quality assessment validated by experts	Agile RE
Crabb & Jones (2024) [25]	"Draft materials" workflow (LLM generates, engineer refines) effective	Industry practice

3.6 Research Gaps & Our Contribution

Gap Identified	Evidence	Our Response
No MCP-based MBSE integration	0 MCP servers for SysML in 7,364+ public repos	`open-mcp-sysml` MCP server
Limited open source SysML v2 + AI tooling	Sensmetry sysml-2ls archived Oct 2025	`tree-sitter-sysml` grammar (MIT)
Raw LLM capability insufficient	SysMBench: 4% BLEU across 17 LLMs	Grammar-aware MCP tools provide structural feedback
Validation approaches still emerging	Papers focus on specific interventions	Hybrid architecture: local parse + API validation
No standardized AI-MBSE interface	Each paper implements custom integration	MCP protocol standardization

3.7 Synthesis

Analysis of the literature reveals consistent themes directly applicable to MCP server design.

3.7.1 Context Management Strategies

Strategy	Papers	Trade-off
Avoidance	Hendricks	Limit LLM to sentence-level; sidesteps problem but limits capability
Staged decomposition	Li, NOMAD	Pipeline with intermediate representations; adds latency but controls context
Template-mediated structuring	SysTemp	Templates constrain output to valid patterns; requires template library
Progressive narrowing	Darm	Similarity→re-rank→traversal; requires graph structure

Strategy	Papers	Trade-off
Multi-agent partitioning	NOMAD, Erikstad, SysTemp	Each agent sees only what it needs; adds orchestration complexity
Progressive disclosure	Lopopolo (OpenAI)	Map-first entry point with structured deep dives; scales to 1M LOC

3.7.2 Design Patterns for MCP Tools

1. **JSON intermediate representation** (Li, NOMAD): Convert SysML v2 to structured JSON before LLM processing
2. **Confidence scoring** (Li): Tool responses should include confidence metadata
3. **Reachability-based pruning** (Darm, Mao): Graph traversal to select relevant context
4. **Quality model injection** (Trendowicz): Provide explicit evaluation criteria in prompt

3.7.3 Validated Claims

- **SysML v2 textual notation is LLM-friendly** (Bader explicitly recommends it over XMI)
- **Raw LLM capability is insufficient for system model generation** (SysMBench: best BLEU 4%, best SysMEval-F1 62% across 17 LLMs)
- **Corpus scarcity demands structural tooling** (SysTemp: templates compensate for lack of training data; SysMBench: enhancement strategies provide only marginal improvement)
- **Draft-then-refine workflow works** (Crabb: engineer maintains control)
- **Single-shot generation is insufficient** (Ferrari: correctness not significantly above baseline)
- **Session isolation essential** (Ferrari: memory pollution causes hallucinations)
- **Environment design outweighs direct coding** (Lopopolo: 1M LOC product built with zero manually-written code; engineering effort shifts to harness design)

3.7.4 Implications for Tool Architecture

The literature supports our central thesis (Section 1.5): harness design matters more than model capability. SysMBench [15] provides the strongest quantitative evidence—when the best available LLMs achieve only 4% BLEU on system model generation, external tooling is not optional but essential. Key architectural implications:

- **Granular tools over monolithic operations**: Enable staged workflows with verification gates
- **Server-side context selection**: Use parser/grammar to extract relevant subgraphs before LLM query
- **Iterative refinement support**: Design for multi-turn interactions, not single-shot generation
- **Validation hooks**: Syntax checking essential; LLMs cannot reliably self-correct parsing errors

Chapter 4

Model Context Protocol

4.1 Overview

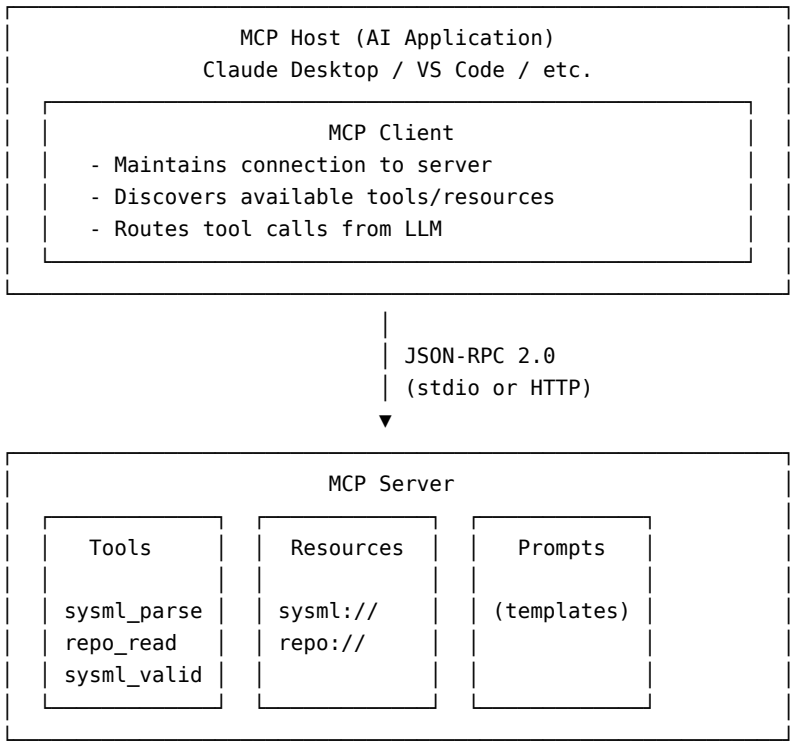
The Model Context Protocol (MCP) [2] is an open standard for connecting AI applications to external systems. Released by Anthropic in November 2024, MCP provides a standardized way for AI assistants to access data sources, execute tools, and interact with domain-specific systems.

Think of MCP as a "USB-C port for AI applications"—a universal interface that allows any MCP-compatible AI host (Claude Desktop, VS Code, custom applications) to connect to any MCP server providing specialized capabilities.

4.2 Architecture

MCP follows a client-server architecture with three key participants:

- **MCP Host**: The AI application (Claude Desktop, VS Code) that coordinates connections
- **MCP Client**: A component within the host that maintains a connection to one MCP server
- **MCP Server**: A program that provides context (tools, resources) to clients



4.2.1 Transport Mechanisms

MCP supports two transport layers:

Transport	Use Case	Characteristics
stdio	Local processes	Claude Desktop, VS Code; no network overhead
HTTP	Remote/CI deployment	Team servers, CI pipelines (e.g., GitLab CI)

The open-mcp-sysml server currently supports stdio transport for local development with Claude Desktop. HTTP transport for remote/CI deployment is planned for Phase 2.

4.2.2 Protocol Flow

1. **Initialize**: Client and server negotiate capabilities
2. **Discover**: Client lists available tools and resources
3. **Execute**: Client calls tools or reads resources as needed
4. **Notify**: Server sends real-time updates when state changes

4.3 Model Context Protocol (MCP) Primitives

MCP defines three core primitives that servers expose to clients:

4.3.1 Tools

Tools are executable functions that AI applications can invoke. Each tool has:

- **Name**: Unique identifier (e.g., `sysml_parse`)
- **Description**: What the tool does
- **Input Schema**: JSON Schema defining expected parameters
- **Output**: Structured response (text, JSON, errors)

Tools enable AI assistants to take actions—reading files, validating models, committing changes—rather than just providing information.

4.3.2 Resources

Resources are read-only data sources accessed via URI patterns. They provide contextual information without side effects:

- `sysml://examples/vehicle` — bundled example model
- `repo://myorg/project/file/model.sysml` — file from Git repository

Resources let AI assistants browse and read project content without executing operations.

4.3.3 Prompts

Prompts are reusable interaction templates that help structure LLM conversations. While MCP supports prompts, our SysML v2 server does not implement them—the tools and resources provide sufficient capability for MBSE workflows.

4.4 SysML v2 Server Design

The SysML v2 MCP server exposes tools and resources tailored for AI-augmented MBSE workflows. Design aligns with requirements in Section 9.3.1.

4.4.1 Tool Definitions

Implemented (Phase 1):

Tool	Purpose	Inputs	Output
`sysml_parse`	Extract elements from SysML text	`source`, `detail_level`	Element list (JSON, L0/L1/L2)
`sysml_validate`	Validate SysML v2 syntax	`source`	Parse diagnostics
`sysml_list_definitions`	List definitions in SysML text	`source`	Definition names and types
`repo_list_files`	List files in Git repository	`project`, `path`, `ref`, `sysml_only`	File list
`repo_get_file`	Read file from Git repository	`project`, `path`, `ref`	File content

Planned (Phase 2+):

Tool	Purpose	Status
`sysml_query`	Query elements by type/properties	Planned
`repo_commit`	Commit file changes	Planned

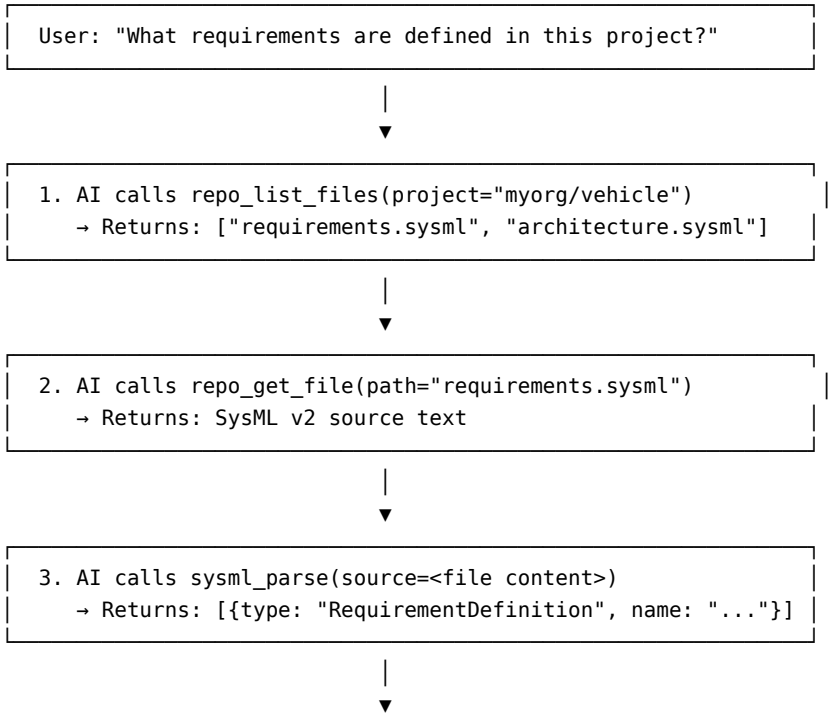
4.4.2 Resource URIs (Planned)

Resource URIs are not yet implemented; the current server exposes tools only. Planned resource patterns:

Pattern	Example	Description
`sysml://examples/{name}`	`sysml://examples/vehicle`	Bundled example models
`repo://{project}/file/{path}`	`repo://myorg/models/file/vehicle.sysml`	Git repository file

4.4.3 Typical Workflow

A systems engineer asks their AI assistant about requirements in a SysML project:



```
AI: "This project defines 12 requirements including..."
```

The AI can continue the conversation—suggesting improvements, drafting new requirements, validating changes—all while maintaining full project context through the MCP server.

4.5 Implementation Considerations

4.5.1 Error Handling

The server handles degraded conditions gracefully:

Condition	Behavior
Invalid SysML syntax	Return parse errors with line numbers via tree-sitter diagnostics
Git provider authentication failure	Return clear error with remediation steps
File not found	Return structured error with available paths
SysML v2 API unavailable	Planned: fall back to local parsing (API integration is Phase 2+)

4.5.2 Security

- Git provider tokens passed via environment variable (e.g., `GITLAB_TOKEN`, `GITHUB_TOKEN`)
- Tokens never logged or included in error messages
- Input validation prevents injection attacks
- HTTP transport supports TLS for remote deployment

4.5.3 Deployment Modes

Mode	Transport	Configuration	Use Case
Local	stdio	Claude Desktop config	Individual engineer
Team	HTTP	Docker/Podman	Shared team server
CI/CD	HTTP	CI service container	Automated validation

See Section 10.13 for detailed deployment architecture.

Chapter 5

Tooling Ecosystem

5.1 Ecosystem Overview

Following SysML v2's OMG adoption in July 2025, a tooling ecosystem is emerging. This chapter surveys the landscape to identify integration opportunities and position our project.

Category	Examples	Our Relationship
Package Management	Sysand	Integration target
Commercial MBSE Platforms	SysGit, IBM ELM, Cameo	Reference implementations
Open Source Parsers	MontiCore, sysml.rs	Peer projects
Model Collections	GfSE models	Test corpus
AI Integration	(gap)	Our contribution

5.2 Commercial Platforms

5.2.1 SysGit (Prewitt Ridge)

SysGit represents the commercial state-of-the-art for Git-native SysML v2 tooling. Founded by ex-SpaceX engineers (Steve Massey, Zeke Brechtel), they've achieved DoD Tradewinds "Awardable" status and partnerships with NASA/JPL, Anduril, and BAE Systems.

Key Technical Decisions:

- **Zero data storage**: Pure interface layer over Git providers
- **Post-cloud architecture**: Inherits security from existing Git infrastructure
- **Offline AI**: Fully air-gapped operation supported
- **Graph object model**: Proprietary representation enabling "agentic AI"

5.2.1.1 Feature Comparison

Capability	SysGit	open-mcp-sysml
Requirements capture	Full GUI	Out of scope
System modeling	Full GUI	Parsing only
Git integration	Native	Via MCP tools
SysML v2 parsing	Proprietary	tree-sitter
AI integration	Proprietary agents	MCP protocol (open)
Licensing	Commercial	MIT
Target user	Systems engineers	AI assistants

SysGit and `open-mcp-sysml` are **complementary, not competing**—SysGit is a full MBSE platform while we provide an AI integration layer. SysGit co-founder Steve Massey authored the Agile Hardware paper [9] validating the Git-native MBSE approach.

5.2.2 Other Commercial Tools

Tool	Vendor	SysML v2 Status
Cameo/MagicDraw	Dassault	Beta support
IBM ELM	IBM	Rhapsody integration
Capella	Eclipse	No SysML v2

5.3 Open Source Ecosystem

5.3.1 Sensmetry / Sysand

Sensmetry provides the most comprehensive open source SysML v2 tooling outside the official OMG implementation.

Product	License	Description
Sysand	MIT/Apache-2.0	Package manager for SysML v2/KerML
Syside Editor	Free	VS Code extension (basic language support)
Syside Modeler	Commercial	Full modeling environment
sysml-2ls	Archived	Former open source LSP

> ⚠ Commercial Pivot
>
> Sensmetry's open source LSP (`sysml-2ls`) was **archived October 2025** when they pivoted to commercial Syside products. This creates a gap that `tree-sitter-sysml` partially fills.

Sysand provides package management (dependency resolution, lock files, environment isolation) that our MCP server can consume via `sysand sources` CLI or Rust crate integration.

5.3.2 Other Open Source Parsers

Parser	Language	Status	Notes
MontiCore/sysmlv2	Java	Active	RWTH Aachen; useful for cross-validation
artob/sysml.rs	Rust	WIP	Potential collaboration
PySysML2 (DoD DAF)	Python/ANTLR	Low activity	Government-originated
Samonjourus/tree-sitter-sysmlv2	JavaScript	Incomplete	Only other tree-sitter attempt

5.3.3 Model Collections

Collection	Purpose
GfSE/SysML-v2-Models	Teaching, parser testing, LLM training
sensmetry/advent-of-sysml-v2	Educational challenges
OMG training/examples	Reference corpus

5.4 MCP Landscape Analysis

As of February 2026, over 7,364 public MCP server repositories exist:

Category	Count
Database access	500+
Code repositories	200+
MBSE/SysML	**0**

No MCP server exists for MBSE or SysML in the public ecosystem. This confirms a **first-mover opportunity** for `open-mcp-sysml`.

5.5 Ecosystem Position

Dimension	Position
Uniqueness	Only MCP server for SysML v2 (first-mover)
vs. Sysand	Complementary—we consume models they manage
vs. SysGit	Non-competing—different scope and audience
Open Source Gap	`tree-sitter-sysml` partially replaces archived `sysml-2ls` (parsing, not full LSP)

vs. Commercial Tools	vs. Other Open Source
Open protocol (MCP)	Production-ready grammar (100% coverage)
Self-hosted	Rust-based (vs. Java-heavy ecosystem)
MIT licensed	Multi-platform bindings

Chapter 6

Systems Engineering Plan

6.1 Project Overview

6.1.1 Objectives

Per [1, Sec. 2.3.4.1], the project planning process establishes plans for accomplishing project objectives within project constraints. This section defines the project's technical and programmatic objectives.

Technical Objectives:

1. Develop an open source MCP server that bridges AI assistants with SysML v2 models
2. Integrate with Git providers for model storage and version control (GitLab as reference)
3. Integrate with SysML v2 API for model validation and querying
4. Support both stdio and HTTP transport for flexible deployment

Programmatic Objectives:

1. Demonstrate INCOSE systems engineering principles for academic capstone
2. Produce three academic publications:
   - **GVSETS 2026**: AI-augmented MBSE with MCP (Mar-Apr 2026)
   - **Grammar Transposition**: KEBNF to tree-sitter methodology (2026)
   - **INCOSE 2027**: SE benchmark for AI evaluation (2027)
3. Establish open source project with community contribution potential
4. Contribute tree-sitter-sysml grammar to tree-sitter organization
5. Position within SysML v2 ecosystem: complement Sysand (package management) and SysGit (commercial platform)

6.1.2 Scope

In Scope:

- MCP protocol implementation (tools, resources)
- Git provider API integration (read, list, commit, MR) with GitLab as reference
- SysML v2 API client (projects, elements, queries, validation)
- Basic SysML v2 textual parsing
- Container deployment support
- SE documentation (SEP, SyRS, ADD, VVP, RTM)
- Demonstration workflow: GitLab + Duo + OpenCode

Out of Scope:

- Full SysML v2 grammar compliance (current scope is documented subset per Section 12.2; full compliance is future work)
- Multi-agent architectures
- Additional Git providers (GitHub, Gitea) - architecture supports, not implemented

- AI benchmarking framework (future INCOSE paper topic)
- Direct GitLab upstream integration (remains independent open source)

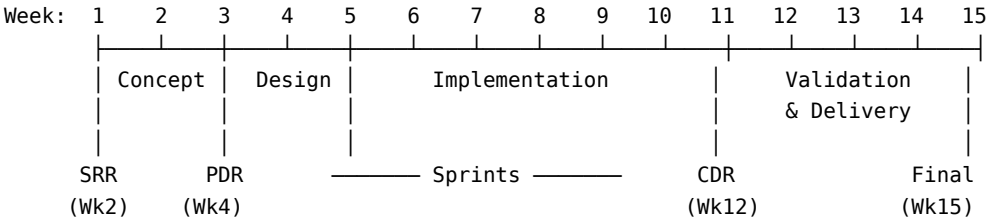
6.1.3 Constraints

Constraint	Impact	Mitigation
15-week timeline	Limits feature scope	Prioritized phased delivery
No local container builds (macOS)	CI-only container testing	Document in VVP, test in CI
SysML v2 API server complexity	Optional dependency	Basic parsing works offline
Academic deliverables parallel	Shared effort required	Clear RACI, integrated schedule

6.2 Lifecycle Model

We adopt a hybrid approach: Agile sprints for implementation velocity with formal SE gates (SRR, PDR, CDR) for academic rigor.

Pre-work: Early January 2026 - Initial research into SysML v2 specifications and prior art.



6.3 Technical Reviews

Review	Week	Purpose	Participants
SRR (System Requirements Review)	2	Baseline requirements, approve SEP	Andrew Dunn, Greg Pappas, Dr. Rapp
PDR (Preliminary Design Review)	4	Approve architecture, confirm build plan	Andrew Dunn, Greg Pappas, Dr. Rapp
CDR (Critical Design Review)	12	Verify implementation, approve for delivery	Andrew Dunn, Greg Pappas, Dr. Rapp

6.4 Review Entry/Exit Criteria

6.4.1 System Requirements Review (SRR)

- **Entry**: Problem statement defined, stakeholders identified, draft SEP
- **Exit**: SyRS baselined, SEP approved, risks identified, PDR scheduled

6.4.2 Preliminary Design Review (PDR)

- **Entry**: Requirements stable, architecture concepts documented
- **Exit**: ADD approved, interfaces defined, implementation plan confirmed

6.4.3 Critical Design Review (CDR)

- **Entry**: Implementation complete, V&V executed
- **Exit**: All acceptance criteria met, ready for delivery

6.5 Schedule

Week	Dates	Phase	Key Activities	Deliverables
0	Jan 1-11	Pre-work	Research SysML v2 specs, prior art analysis	Research notes
1	Jan 12-18	Concept	Finalize plan, set up repos, Quarto scaffold	This plan document
2	Jan 19-25	Concept	Requirements elicitation, stakeholder analysis	SRR: SEP v1, SyRS v1
3	Jan 26-Feb 1	Design	Architecture development, interface definition	ADD draft
4	Feb 2-8	Design	Design review, V&V planning	PDR: ADD v1, VVP v1
5	Feb 9-15	Impl	Phase 1: tree-sitter-sysml grammar foundation	Grammar scaffold, 10% coverage
6	Feb 16-22	Impl	Phase 2: Extended grammar	Requirements, actions
7	Feb 23-Mar 1	Impl	Phase 2: Extended grammar	50% coverage target
8	Mar 2-8	Impl	Phase 3 start, GVSETS draft prep	MCP server integration
9	Mar 9-15	Impl	Phase 3: MCP server integration	sysml_parse, repo tools
10	Mar 16-22	Impl	Phase 3 complete	Full tool suite
11	Mar 23-29	Impl	Integration testing, bug fixes	Stable release
12	Mar 30-Apr 5	V&V	V&V execution, CDR prep	CDR: V&V results
13	Apr 6-12	Delivery	Paper revision, demo prep	GVSETS final paper
14	Apr 13-19	Delivery	Documentation finalization	Final docs
15	Apr 20-25	Delivery	Capstone submission	Final documentation package

6.6 Key Milestones

Date	Milestone
Jan 12	Concept phase begins (Week 1)
Jan 18	Plan review with Greg Pappas and Dr. Rapp
Feb 14	SRR and PDR conducted
Mar 23	GVSETS draft paper submitted
May 1	GVSETS notification of acceptance
Jun 5	GVSETS final paper submitted
Jul 23	GVSETS final presentations due
Aug 11	GVSETS paper presentation (Novi, MI)
Apr 25	Capstone deliverables complete

6.7 Configuration Management

6.7.1 Version Control

- **Branching Model**: GitLab Flow (main + feature branches, MRs required)
- **Commit Convention**: Conventional Commits (feat:, fix:, docs:, chore:)
- **Protected Branches**: main requires MR approval

6.7.2 Artifact Versioning

Artifact	Versioning Scheme
Software	SemVer (v0.1.0, v0.2.0, …)
SE Documents	Date-based (SEP-2026-01-22) or revision (SyRS v1.0, v1.1)

Artifact	Versioning Scheme
Container Images	Git SHA + SemVer tags

6.7.3 Baseline Management

Baseline	Contents	Established At
Requirements Baseline	SyRS v1.0	SRR (Week 2)
Design Baseline	ADD v1.0, VVP v1.0	PDR (Week 4)
Product Baseline	Software v1.0, final docs	CDR (Week 12)

6.8 Risk Management

Per [1, Sec. 2.3.4.4], the risk management process identifies, analyzes, treats, and monitors risks throughout the project lifecycle.

6.8.1 Risk Categories

Category	Description
Technical	Risks related to technology choices, implementation complexity
Schedule	Risks related to timeline, resource availability
External	Risks from external dependencies, stakeholder changes
Quality	Risks related to defects, compliance, acceptance

6.8.2 Risk Scoring

Likelihood: Low (1) / Medium (2) / High (3)

Impact: Low (1) / Medium (2) / High (3)

Risk Score: Likelihood × Impact (1-9)

6.8.3 Risk Register

ID	Risk Description	Category	L	I	Score	Treatment Strategy	Owner	Status
R1	SysML v2 API server difficult to deploy locally	Technical	1	2	2	**Avoided**: Repository tools implemented without API dependency; SysML v2 API deferred to post-capstone.	Andrew	**Mitigated**
R2	GVSETS paper deadline aggressive given parallel implementation	Schedule	2	2	4	**Accept**: Draft deadline Mar 23 with 5-week buffer to final (Jun 5). Benchmark execution remains critical path.	Andrew	Open
R3	Stakeholder availability for reviews limited	External	1	2	2	SRR and PDR completed Feb 14 with all stakeholders present.	Greg	**Closed**

ID	Risk Description	Category	L	I	Score	Treatment Strategy	Owner	Status
R4	Rust MCP SDK or tree-sitter bindings have limitations	Technical	1	3	3	rmcp SDK and tree-sitter Rust bindings proven by working implementation (5 MCP tools, 22 tests passing).	Andrew	**Closed**
R5	Container testing blocked on local macOS development	Technical	3	1	3	**Accept**: CI-only container validation; document limitation in VVP. Local testing uses native Rust binaries.	Andrew	Open
R6	Scope expansion from ecosystem exploration	Schedule	2	1	2	**Accepted**: Scope expanded to 7 projects (grammar benchmark, kebnf converter, PhD planning) but did not impact GVSETS publication, tree-sitter-sysml release, or MCP server delivery. Expanded scope provides deeper ecosystem understanding that strengthens future work articulation in publications. Managed via multi-agent orchestration with specialized skills and personas.	Greg	**Mitigated**
R7	SysML v2 specification changes during project	External	1	2	2	OMG spec stable; grammar achieves 99.6% external file coverage. No spec changes encountered.	Andrew	**Closed**
R8	Grammar complexity requires external scanner	Technical	2	2	4	100% training file coverage and 99.6% external coverage achieved without external scanner. All constructs (expressions, constraints, states, actions) handled by pure grammar.js.	Andrew	**Closed**

ID	Risk Description	Category	L	I	Score	Treatment Strategy	Owner	Status
R9	Tree-sitter org may not accept grammar contribution	External	2	1	2	**Accept**: Standalone value regardless. GitLab vendor/grammars/ is alternative contribution path. Grammar benefits MBSE community either way.	Andrew	Open

6.8.4 Risk Monitoring

Risks will be reviewed at each technical review (SRR, PDR, CDR) and during weekly sync meetings. New risks should be added to this register with initial assessment.

Escalation Criteria: Risks with Score 6 require immediate mitigation plan and advisor notification.

6.9 Review Status

This section tracks actual review completion status. Entry/exit criteria are defined in Section 6.3.

Attendees for all reviews:

- Andrew Dunn (Technical Lead, GitLab Public Sector)
- Greg Pappas (SE Lead, DoD Army AFC-DEVCOM)
- Dr. Stephen Rapp (Advisor, Wayne State University ISE)

6.9.1 Status Summary

Review	Target Date	Actual Date	Entry Criteria	Exit Criteria	Status
SRR	Jan 25, 2026	Feb 14, 2026	Met	Met (with caveats)	**Complete**
PDR	Feb 8, 2026	Feb 14, 2026	Met	Met (with caveats)	**Complete**
CDR	-	-	Pending	Pending	Not Started

6.9.2 SRR Readiness Checklist

SRR Complete (Feb 14, 2026)

Entry Criteria (per Section 6.3):
- ☒ Problem statement defined — Section 6.1
- ☒ Stakeholders identified — Section 8.1 (8 stakeholders)
- ☒ Draft SEP complete — This chapter

Artifacts Reviewed:

Artifact	Section	Status
SEP v1	Section 6.1 through Section 6.8	Complete
Stakeholder Analysis	Section 8.1	Complete
Stakeholder Needs	SN-001 through SN-015	Complete
Stakeholder Requirements	SR-001 through SR-015	Complete
SyRS v1	Chapter 7	Complete
Risk Register	Section 6.8	Complete

Exit Criteria:
- ☒ SyRS baselined
- ☒ SEP approved
- ☒ Risks identified and acknowledged
- ☒ PDR scheduled

Caveats (accepted per tailoring rationale):
1. Interface requirements (IR-xxx) deferred — formal IDs not required at this stage; interfaces documented informally in Section 10.10
2. Stakeholder validation conducted informally through iterative development rather than formal interviews
3. Requirements baselined with understanding that implementation discoveries (particularly benchmark execution) may drive updates

Tailoring Rationale: SRR was conducted to bootstrap the project and inform the GVSETS publication as the primary capstone deliverable. Requirements have evolved as scope expanded through implementation. Post-capstone work will proceed informally in sprint-based iterations. Per INCOSE Handbook 4.3.4, this tailoring is appropriate for a software-intensive academic project with a small team.

6.9.3 PDR Readiness Assessment

PDR Complete (Feb 14, 2026)

Entry Criteria (per Section 6.3):
- ☒ Requirements stable — SyRS v1 baselined at SRR
- ☒ Architecture concepts documented — ADD complete (617 lines)

Artifacts Reviewed:

Artifact	Section	Status
ADD v1	Section 10.1 through Section 10.14	Complete (context diagram, component architecture, trade study, deployment modes)
VVP v1	Section 11.1 through Section 11.8	Drafted (verification strategy, test cases, CI pipeline)
RTM	Section 16.1 through Section 16.4	Complete (4 trace layers)

Exit Criteria:
- ☒ ADD approved — Architecture validated by working implementation (5 MCP tools, 22 tests)
- ☒ Interfaces defined — Section 10.10 (MCP, Git Provider, SysML v2 API)
- ☒ Implementation plan confirmed — WBS phases validated by actual delivery

Caveats (accepted per tailoring rationale):
1. VVP requirement IDs need reconciliation (FR-Umcp→FR-MCP naming mismatch) and substantive update to reflect actual implementation test structure
2. ADD tool definitions need update to reflect Phase 1 actual tool names vs. planned names
3. Same tailoring philosophy as SRR: implementation-informed, GVSETS-focused, post-capstone work proceeds informally

Tailoring Rationale: Architecture was validated by successful implementation (tree-sitter integration, 5 MCP tools operational, 100% training coverage) prior to formal PDR closure. This "build then verify" approach is appropriate for a rapid-iteration academic project per INCOSE Handbook 4.3.4.

6.9.4 Action Items

Review	Item	Owner	Due	Status
SRR	Baseline SyRS after advisor approval	Greg	Feb 14	Complete

Review	Item	Owner	Due	Status
SRR	Update SEP with review outcomes	Andrew	Feb 14	Complete
PDR	Complete ADD draft	Andrew	Feb 14	Complete
PDR	Draft VVP strategy	Greg	Feb 14	Complete
PDR	Reconcile VVP requirement IDs (FR-Umcp→FR-MCP)	Andrew	Feb 14	Complete
PDR	Update ADD tool definitions to match implementation	Andrew	Feb 14	Complete
PDR	Update VVP test cases to reflect actual test structure	Andrew	-	In Progress

Action items updated after Feb 14, 2026 SRR/PDR closure session with Greg Pappas.

Chapter 7

Work Breakdown Structure

7.1 Overview

This chapter defines the project Work Breakdown Structure and serves as the central task tracker. Tasks link directly to the sections where work is needed.

For current implementation status across all repositories, see Section 1.1.3 on the index page.

7.2 Work Breakdown Structure Tree

- **1.0 AI-Augmented MBSE via MCP Project**
  - **1.1 Project Management**
    - 1.1.1 Planning & Coordination
    - 1.1.2 Technical Reviews (SRR, PDR, CDR)
    - 1.1.3 Risk Management
  - **1.2 Systems Engineering**
    - 1.2.1 Systems Engineering Plan (SEP)
    - 1.2.2 Stakeholder Analysis
    - 1.2.3 System Requirements Specification (SyRS)
    - 1.2.4 Architecture Design Description (ADD)
    - 1.2.5 Verification & Validation Plan (VVP)
    - 1.2.6 Requirements Traceability Matrix (RTM)
  - **1.3 Software Development**
    - 1.3.0 Phase 0: Go Prototype (Superseded)
      - ~~1.3.0.1 Server scaffold (Go, MCP SDK)~~
      - ~~1.3.0.2 Basic sysml_parse tool~~
      - ~~1.3.0.3 Example resources~~
    - 1.3.1 Phase 1: tree-sitter-sysml Grammar Foundation
      - 1.3.1.1 Grammar scaffold (tree-sitter-cli, bindings)
      - 1.3.1.2 Dual CI: GitHub Actions + GitLab CI
      - 1.3.1.3 Core grammar (package, part def, part)
      - 1.3.1.4 Corpus test suite (target: 100% pass)
      - 1.3.1.5 Training file coverage tracking (milestone: 10%)
    - 1.3.2 Phase 2: Extended Grammar + Playground
      - 1.3.2.1 Extended grammar (requirement def, requirement)
      - 1.3.2.2 Extended grammar (action, item, port)
      - 1.3.2.3 Extended grammar (connection, allocation)
      - ~~1.3.2.4 GitLab Pages playground (WASM)~~ — Removed from scope
      - 1.3.2.5 Training file coverage (milestone: 50%)
    - 1.3.3 Phase 3: open-mcp-sysml Integration
      - 1.3.3.1 Rust workspace setup

        - 1.3.3.2 tree-sitter-sysml Rust bindings integration
        - 1.3.3.3 RepoClient trait + GitLab implementation
        - 1.3.3.4 MCP server scaffold (rmcp)
        - 1.3.3.5 sysml_parse tool (tree-sitter backend)
        - 1.3.3.6 repo_list_files, repo_get_file tools
        - 1.3.3.7 Example resources
    * 1.3.4 Phase 4: HTTP Transport (Optional)
        - 1.3.4.1 Streamable HTTP server
        - 1.3.4.2 CORS configuration
    * 1.3.5 kebnf-to-tree-sitter (Spec-Driven Grammar)
        - 1.3.5.1 KEBNF parser (Chumsky 1.0)
        - 1.3.5.2 Tree-sitter emitter
        - 1.3.5.3 Semantic mapping generator
        - 1.3.5.4 Iterative conflict resolution
        - 1.3.5.5 SysML v2 case study
    * 1.3.6 sysml-grammar-benchmark
        - 1.3.6.1 Repository scaffold
        - 1.3.6.2 Quarto + D3 dashboard
        - 1.3.6.3 Parser adapters (tree-sitter, kebnf-generated)
        - 1.3.6.4 Corpus submodules (OMG, GfSE, Advent)
        - 1.3.6.5 GitLab CI pipeline
        - 1.3.6.6 External parser comparison (sysml.rs, MontiCore)
- **1.4 Infrastructure**
    * 1.4.1 Repository Setup
    * 1.4.2 CI/CD Pipeline (software)
    * 1.4.3 CI/CD Pipeline (documentation)
    * 1.4.4 Container Build & Registry
- **1.5 Documentation**
    * 1.5.1 Quarto Book Setup
    * 1.5.2 Chapter Authoring
    * 1.5.3 GitLab Pages Deployment
    * 1.5.4 Software README/CONTRIBUTING
    * 1.5.5 Literature Review Chapter
        - 1.5.5.1 Paper inventory and citations
        - 1.5.5.2 Individual paper summaries
        - 1.5.5.3 Synthesis and gap analysis
    * 1.5.6 Ecosystem Chapter
        - 1.5.6.1 Sensmetry/Sysand analysis
        - 1.5.6.2 SysGit analysis
        - 1.5.6.3 Open source parser inventory
        - 1.5.6.4 Collaboration strategy
    * 1.5.7 Ecosystem Outreach
        - 1.5.7.1 Sensmetry forum/email outreach
        - 1.5.7.2 tree-sitter org submission prep
- **1.6 External Deliverables**
    * 1.6.1 GVSETS 2026 Paper
        - 1.6.1.1 GVSETS Abstract
        - 1.6.1.2 GVSETS Draft (Mar 23)
        - 1.6.1.3 GVSETS Final (Jun 5)
    * 1.6.2 Grammar Transposition Paper (2026)
        - 1.6.2.1 kebnf-to-tree-sitter tool complete
        - 1.6.2.2 Paper outline
        - 1.6.2.3 Draft submission (Q3-Q4 2026)
    * 1.6.3 INCOSE 2027 Benchmark Paper (Future)
        - 1.6.3.1 Task design (Q3 2026)
        - 1.6.3.2 Pilot study (Q4 2026)
        - 1.6.3.3 Paper submission (Q3 2027)

- 1.6.4 Capstone Submission (Apr 25)

7.3 1.1 Project Management

7.3.1 1.1.1 Project Planning

Per [1, Sec. 2.3.4.1].

- ☒ Define project objectives, scope, constraints - Section 6.1
- ☒ Develop breakdown structures (WBS) - This chapter
- ☒ Establish schedule with milestones - Section 6.5
- ☒ Generate SEMP/SEP - Section 6.2

7.3.2 1.1.2 Technical Reviews

Per [1, Sec. 2.1.4].

- ☒ SRR - Section 6.9 (Complete Feb 14)
- ☒ PDR - Section 6.9 (Complete Feb 14)
- ☐ CDR - Section 6.9

7.3.3 1.1.3 Risk Management

Per [1, Sec. 2.3.4.4].

- ☒ Identify risks and opportunities - Section 6.8
- ☒ Establish risk thresholds and categories - Section 6.8
- ☒ Define treatment strategies - Section 6.8

7.4 1.2 Systems Engineering

7.4.1 1.2.1 Systems Engineering Plan (SEP)

Per [1, Sec. 2.3.4.1].

- ☒ Life cycle model definition - Section 6.2
- ☒ Technical review entry/exit criteria - Section 6.3
- ☒ Configuration management approach - Section 6.7

7.4.2 1.2.2 Stakeholder Analysis

Per [1, Sec. 2.3.5.2].

- ☒ Identify stakeholders with interests - Section 8.1
- ☒ Establish stakeholder management approach - Section 8.2
- ☒ Develop operational concept - Section 10.2.2
- ☒ Define stakeholder needs (SN-001 through SN-015) - Section 8.3
- ☒ Transform needs to stakeholder requirements (SR-001 through SR-015) - Section 8.4
- ☒ Consolidate stakeholders (rename Technical Users → MBSE Practitioners, remove Defense/Aerospace)
- ☒ Elevate SysML v2 Implementers importance, add spec compliance requirements
- ☒ Add traceability anchors and cross-references throughout
- ☒ Define validation criteria for stakeholder needs/requirements

7.4.3 1.2.3 System Requirements Specification (SyRS)

Per [1, Sec. 2.3.5.3].

- ☒ Define functional boundary of system - Section 9.2
- ☒ Define system functions with performance - Section 9.3.1
- ☒ Define constraints (operational, regulatory) - Section 9.5

- ☒ Define verification criteria per requirement - Section 9.6
- ☒ Analyze requirements (complete, consistent, feasible) - Section 9.7
- ☒ Research SysML v2 parser approaches (subset vs full compliance) - Section 12.2
- ☒ Add SysML v2 compliance requirements (FR-SYS-006, 007, 008)
- ☒ Document supported grammar subset with coverage rationale - Section 12.2
- ☒ Add traceability anchors and cross-references to FR/NFR tables

7.4.4 1.2.4 Architecture Design Description (ADD)

Per [1, Sec. 2.3.5.4].

- ☒ Identify architecture viewpoints - Section 10.1
- ☒ Define system context and boundary - Section 10.2
- ☒ Synthesize candidate architectures - Section 10.4
- ☒ Select architecture via trade study - Section 2.7
- ☒ Define interfaces (internal/external) - Section 10.10
- ☒ Allocate requirements to elements - Section 10.12

7.4.5 1.2.5 Verification & Validation Plan (VVP)

Per [1, Secs. 2.3.5.9, 2.3.5.11].

- ☒ Define verification scope and strategy - Section 11.1
- ☒ Select verification methods per requirement - Section 11.2
- ☒ Define verification success criteria - Section 11.3
- ☒ Plan enabling systems (test tools, CI) - Section 11.4
- ☒ Define validation approach (stakeholder acceptance) - Section 11.8

7.4.6 1.2.6 Requirements Traceability Matrix (RTM)

Per [1, Sec. 3.2.3].

- ☒ Stakeholder needs → Stakeholder requirements - Section 16.1
- ☒ Stakeholder requirements → System requirements - Section 16.2
- ☒ System requirements → Architecture elements - Section 16.3
- ☒ System requirements → Test cases - Section 16.4

7.5 1.3 Software Development

7.5.1 Phase 0: Go Prototype (Superseded)

- ☒ ~~**1.3.0.1** Server scaffold (Go, MCP SDK)~~ - Superseded
- ☒ ~~**1.3.0.2** Basic `sysml_parse` tool~~ - Superseded
- ☒ ~~**1.3.0.3** Example resources~~ - Superseded

Note (Feb 9, 2026): Go prototype superseded by Rust + tree-sitter implementation for GitLab Knowledge Graph alignment and community grammar reuse.

7.5.2 Phase 1: tree-sitter-sysml Grammar Foundation

- ☒ **1.3.1.1** Grammar scaffold (tree-sitter-cli, all bindings) - Week 5
- ☒ **1.3.1.2** Dual CI: GitHub Actions + GitLab CI - Week 5
- ☒ **1.3.1.3** Core grammar: package, part def, part - Week 5
- ☒ **1.3.1.4** Corpus test suite (87/87 passing) - Week 5
- ☒ **1.3.1.5** Training file coverage tracking (100%) - Week 5

Status (Feb 11, 2026): Phase 1 complete. Grammar scaffold with all bindings, 87/87 corpus tests passing, 100% training file parse rate.

7.5.3 Phase 2: Extended Grammar + Playground

- ☒ **1.3.2.1** Extended grammar: requirement def, requirement - Week 5
- ☒ **1.3.2.2** Extended grammar: action, item, port - Week 5
- ☒ **1.3.2.3** Extended grammar: connection, allocation - Week 5
- ☒ ~~**1.3.2.4** GitLab Pages playground (WASM)~~ - Removed from scope
- ☒ **1.3.2.5** Training file coverage milestone: 100% - Week 5
- ☒ **1.3.2.6** Context-sensitive definition bodies - Week 5
- ☒ **1.3.2.7** Negative test framework (7/8 passing) - Week 5
- ☒ **1.3.2.8** Technical debt documentation - Week 5

Status (Feb 10, 2026): Phase 2 complete. Full grammar coverage achieved ahead of schedule. All constructs implemented: states, constraints, verification, analysis, views, metadata, occurrences, events, messages, and more.

7.5.4 Phase 3: open-mcp-sysml Integration

- ☒ **1.3.3.1** Rust workspace setup - Week 5
- ☒ **1.3.3.2** tree-sitter-sysml Rust bindings integration - Week 5
- ☒ **1.3.3.3** RepoClient trait + GitLab implementation - Complete
- ☒ **1.3.3.4** MCP server scaffold (rmcp) - Week 5
- ☒ **1.3.3.5** `sysml_parse` tool (tree-sitter backend) - Week 5
- ☒ **1.3.3.6** `repo_list_files`, `repo_get_file` tools - Complete
- ☐ **1.3.3.7** Example resources - Deferred
- ☐ **1.3.3.8** Cache ID + Summary optimization pattern - Phase 2 PRD ready
- ☒ **1.3.3.9** L0/L1/L2 tiered responses - Week 5

Status (Feb 14, 2026): Phase 1 Complete. Tree-sitter integration working with 5 MCP tools: `sysml_parse` (L0/L1/L2), `sysml_validate`, `sysml_list_definitions`, `repo_list_files`, `repo_get_file`. 22 tests passing. 100% training file coverage. Phase 2 PRD ready for 7 token reduction strategies.

7.5.5 Phase 4: HTTP Transport (Optional)

- ☐ **1.3.4.1** Streamable HTTP server - Week 11+
- ☐ **1.3.4.2** CORS configuration - Week 11+

7.5.6 Phase 5: kebnf-to-tree-sitter (Spec-Driven Grammar)

- ☒ **1.3.5.1** KEBNF format documentation - Complete
- ☒ **1.3.5.2** Tool architecture design - Complete
- ☒ **1.3.5.3** KEBNF parser (Chumsky 1.0) - Complete (640/640 rules)
- ☒ **1.3.5.4** Tree-sitter grammar.js emitter - Complete (produces output with conflicts)
- ☐ **1.3.5.5** Iterative conflict resolution - In Progress (335+ conflicts)
- ☐ **1.3.5.6** Semantic mapping generator - Pending
- ☐ **1.3.5.7** SysML v2 case study - Q2 2026

Status (Feb 12, 2026): Parser and emitter complete. Generated grammar has 335+ conflicts requiring iterative resolution. Strategy selected: resolve conflicts one at a time, document progress for INCOSE paper. See Section 12.3 for implementation details.

7.5.7 Phase 6: sysml-grammar-benchmark

- ☒ **1.3.6.1** Repository scaffold - Complete
- ☒ **1.3.6.2** PRD and AGENTS.md - Complete
- ☐ **1.3.6.3** Quarto + D3 dashboard - In Progress
- ☐ **1.3.6.4** Parser adapters (tree-sitter) - Pending
- ☐ **1.3.6.5** Corpus submodules (OMG, GfSE, Advent) - Pending
- ☐ **1.3.6.6** GitLab CI pipeline - Pending
- ☐ **1.3.6.7** External parser comparison - Pending

Purpose: Automated grammar benchmark dashboard comparing SysML v2 parser implementations. Provides objective evidence for "production-ready grammar" claims. Community deliverable.

7.6 1.4 Infrastructure

- ☒ **1.4.1** Repository Setup - Complete
- ☒ **1.4.2** CI/CD Pipeline (tree-sitter-sysml) - Complete (GitHub Actions + GitLab CI)
- ☒ **1.4.3** CI/CD Pipeline (documentation) - Complete (HTML + PDF)
- ☐ **1.4.4** CI/CD Pipeline (open-mcp-sysml) - Pending
- ☐ **1.4.5** Container Build & Registry - Pending

7.7 1.5 Documentation

- ☒ **1.5.1** Quarto Book Setup - Complete
- ☒ **1.5.2** Chapter Authoring - Ongoing
  - ☒ Ch 12a: Spec-driven approach (kebnf-to-tree-sitter)
  - ☒ Ch 12b: Grammar retrospective
  - ☒ Ch 12c: Brute force approach
- ☒ **1.5.3** GitLab Pages Deployment - Complete
- ☒ **1.5.4** tree-sitter-sysml README/CONTRIBUTING/AGENTS.md - Complete
- ☐ **1.5.5** open-mcp-sysml documentation - Pending

7.7.1 1.5.6 Literature Review Chapter

- ☒ **1.5.6.1** Paper inventory and citations (15 papers) - Complete
- ☐ **1.5.6.2** Individual paper summaries - Pending
  - ☐ Li et al. (2025) - LLM-Assisted Semantic Alignment for SysML v2
  - ☐ Hendricks & Cicirello (2025) - Text to Model via SysML
  - ☐ Darm et al. (2025) - Inference-Time Intervention
  - ☐ Jerry et al. (2026) - LLM-Driven Accessible Interface
  - ☐ Otten et al. (2026) - LLM Risk Assessment Framework
  - ☐ Giannouris & Ananiadou (2025) - NOMAD Multi-Agent UML
  - ☐ Erikstad (2024) - Multi-Agent LLMs and MBSE
  - ☐ Ferrari et al. (2024) - Requirements to UML Sequence Diagrams
  - ☐ Rouabhia & Hadjadj (2025) - UML Class Diagram Augmentation
  - ☐ Mao et al. (2025) - UML to Code Generation
  - ☐ Trendowicz et al. (2026) - DeepQuali User Story Quality
  - ☐ Bader et al. (2024) - User-Centric MBSE with GenAI
  - ☐ Crabb & Jones (2024) - Accelerating MBSE with GenAI
  - ☐ Neema et al. (2025) - Evaluating Engineering AGI
- ☐ **1.5.6.3** Synthesis and gap analysis - Pending

See Section 3.1 for chapter content.

7.7.2 1.5.7 Ecosystem Chapter

- ☒ **1.5.7.1** Sensmetry/Sysand detailed analysis - Complete
- ☒ **1.5.7.2** SysGit feature comparison - Complete
- ☒ **1.5.7.3** Open source parser inventory - Complete
- ☒ **1.5.7.4** Model collections identified - Complete
- ☒ **1.5.7.5** MCP landscape gap analysis - Complete
- ☒ **1.5.7.6** Integration architecture diagram - Complete
- ☐ **1.5.7.7** Collaboration strategy execution - Pending

See Section 5.1 for chapter content.

7.7.3 1.5.8 Ecosystem Outreach

- ☐ **1.5.8.1** Draft Sensmetry forum post - Pending
- ☐ **1.5.8.2** Sensmetry email outreach - Pending
- ☐ **1.5.8.3** tree-sitter org submission prep - Pending
- ☐ **1.5.8.4** GfSE model collection outreach - Pending

7.8 1.6 External Deliverables

7.8.1 1.6.1 GVSETS 2026 Paper

Repository: `gvsets/` (separate LaTeX project)

- ☒ **1.6.1.1** LaTeX template setup - Complete
- ☒ **1.6.1.2** Abstract written - Complete
- ☐ **1.6.1.3** Draft paper submitted - Due Mar 23
- ☐ **1.6.1.4** Final paper submitted - Due Jun 5
- ☐ **1.6.1.5** Final presentations due - Due Jul 23
- ☐ **1.6.1.6** Paper presentation (Novi, MI) - Aug 11

See Section B.2 for paper outline.

7.8.2 1.6.2 Grammar Transposition Paper (2026)

- ☒ **1.6.2.1** kebnf-to-tree-sitter tool design - Complete
- ☒ **1.6.2.2** Paper outline - Complete (Section B.3)
- ☐ **1.6.2.3** Tool implementation - In Progress
- ☐ **1.6.2.4** Case study execution - Q2 2026
- ☐ **1.6.2.5** Paper draft submission - Q3-Q4 2026

7.8.3 1.6.3 INCOSE 2027 Benchmark Paper (Future)

- ☒ **1.6.3.1** Paper outline - Complete (Section B.4)
- ☐ **1.6.3.2** Task design - Q3 2026
- ☐ **1.6.3.3** Pilot study - Q4 2026
- ☐ **1.6.3.4** Paper submission - Q3 2027

7.8.4 1.6.4 Capstone Submission

- ☐ **1.6.4.1** Final documentation package - Due Apr 25
- ☐ **1.6.4.2** Defense presentation - Apr 2026

7.9 Milestones

Milestone	Planned	Actual	Status
Plan finalized, repos set up	Jan 18	Jan 18	Complete
SRR - SEP, SyRS baselined	Jan 25	Feb 14	**Complete**
PDR - ADD, VVP approved	Feb 8	Feb 14	**Complete**
tree-sitter-sysml Phase 1+2 (100% coverage)	Feb 15	Feb 10	Complete
tree-sitter-sysml 99.6% external coverage	-	Feb 12	Complete
MCP server Phase 1 (tree-sitter integration)	Feb 22	Feb 13	Complete
Benchmark vignettes executed	-	-	Pending
sysml-grammar-benchmark dashboard	-	-	Pending

Milestone	Planned	Actual	Status
GVSETS quantitative section complete	-	-	Pending
GVSETS draft submitted	Mar 23	-	Pending
GVSETS notification of acceptance	May 1	-	Pending
GVSETS final submitted	Jun 5	-	Pending
GVSETS presentations due	Jul 23	-	Pending
GVSETS paper presentation (Novi, MI)	Aug 11	-	Pending
CDR - V&V complete	-	-	Pending
Capstone final delivery	Apr 25	-	Pending

Schedule Note (Feb 14, 2026): Removed fixed week targets for non-GVSETS work items. GVSETS publication timeline updated per NDIA. Remaining work prioritized as backlog rather than date-driven.

7.10 Risk Summary

See Section 6.8 for full risk register.

ID	Risk	L	I	Score	Status
R1	SysML v2 API server difficult to deploy	1	2	2	**Mitigated** — repo tools work without API
R2	GVSETS deadline aggressive	2	2	4	Open — benchmark execution is critical path
R3	Stakeholder availability	1	2	2	**Closed** — SRR/PDR completed Feb 14
R4	Rust MCP SDK/tree-sitter limitations	1	3	3	**Closed** — proven by working implementation
R5	Container testing blocked locally	3	1	3	Open — CI-only, documented in VVP
R6	Scope expansion	2	1	2	**Mitigated** — managed, enriched ecosystem understanding
R7	SysML v2 spec changes	1	2	2	**Closed** — spec stable, 99.6% coverage
R8	Grammar needs external scanner	2	2	4	**Closed** — 100% coverage without scanner
R9	Tree-sitter org may not accept	2	1	2	Open — standalone value regardless

Chapter 8

Stakeholder Analysis

8.1 Stakeholder Identification

Per [1, Sec. 2.3.5.2], the stakeholder needs and requirements definition process identifies stakeholders and their needs throughout the system lifecycle.

Stakeholder	Power	Interest	Strategy	Success Criteria
Academic Advisor	High	High	Manage Closely	Complete SE artifacts, proper methodology
Capstone Collaborator	High	High	Manage Closely	Shared workload, defensible deliverables
Open Source Community	Medium	High	Keep Satisfied	Working software, good documentation
GitLab (Author Affiliation)	Medium	Medium	Keep Informed	Working demonstration, GVSETS presentation
MBSE Practitioners	Medium	High	Keep Satisfied	Solves real workflow problems
SysML v2 Implementers	Medium	Medium	Keep Informed	Correct API usage, grammar conformance
INCOSE/SE Community	Low	Medium	Keep Informed	Novel contribution, reproducible results
Sensmetry (Sysand)	Low	Medium	Keep Informed	Interoperability, grammar contribution

8.2 Team & Responsibilities

Role	Person	Affiliation
Technical Lead	Andrew Dunn	GitLab Public Sector
SE Lead	Greg Pappas	DoD, Army, AFC-DEVCOM
Advisor	Dr. Stephen Rapp	Wayne State University, ISE

WBS Element	Andrew	Greg	Dr. Rapp
Planning & Reviews	R	C	A
SEP, SyRS, VVP, RTM	C	R	A
ADD	R	C	A
Software Development	R	I	I
Quarto Book	C	R	I

WBS Element	Andrew	Greg	Dr. Rapp
Papers (GVSETS, INCOSE)	R	R	C

Legend: R=Responsible, A=Accountable, C=Consulted, I=Informed

8.3 Stakeholder Needs

ID	Stakeholder	Need
SN-001	OSS Community	Git provider API integration for existing repositories
SN-002	OSS Community	Truly open source (MIT license)
SN-003	MBSE Practitioners	CI/CD integration examples
SN-004	GitLab	Demonstrate AI-augmented MBSE for GVSETS
SN-005	Advisor	Follow INCOSE SE processes
SN-006	Advisor	Formal technical reviews (SRR, PDR, CDR)
SN-007	MBSE Practitioners	Easy installation (single binary)
SN-008	MBSE Practitioners	Clear documentation with examples
SN-009	DevOps Engineers	Container deployment support
SN-010	MBSE Practitioners	Natural language model queries
SN-011	MBSE Practitioners	Self-hosted Git provider support
SN-012	MBSE Practitioners	Model validation against SysML v2 spec
SN-013	Tool Vendors	Open standards, complement commercial tools
SN-014	SysML v2 Implementers	OMG spec conformance
SN-015	SysML v2 Implementers	Parse errors with location info

8.4 Stakeholder Requirements

ID	Traces To	Requirement
SR-001	SN-001	Integrate with Git provider REST APIs (GitLab as reference)
SR-002	SN-002	MIT open source license
SR-003	SN-003	CI/CD integration examples (GitLab CI as reference)
SR-004	SN-005	Produce SE artifacts per INCOSE Handbook
SR-005	SN-006	Conduct SRR, PDR, CDR with documented criteria
SR-006	SN-007	Single static binary, no external dependencies

ID	Traces To	Requirement
SR-007	SN-008	README with installation/configuration instructions
SR-008	SN-008	Example SysML v2 models demonstrating capabilities
SR-009	SN-009	OCI-compliant container image
SR-010	SN-011	Self-hosted Git provider support via configurable URL
SR-011	SN-012	SysML v2 syntax validation via API integration
SR-012	SN-010	Parse SysML v2 textual notation, extract elements
SR-013	SN-014	Parse SysML v2 conforming to OMG grammar spec
SR-014	SN-015	Parse errors include file, line, column
SR-015	SN-014	Documented subset coverage

8.5 Validation Criteria

Requirement	Validation	Acceptance Criteria
SR-001	Demo	Read file from GitLab repository
SR-002	Inspect	MIT LICENSE present
SR-003	Demo	CI pipeline example executes
SR-004	Inspect	SE artifacts present, INCOSE-aligned
SR-005	Inspect	Review records document criteria
SR-006	Demo	Binary runs without dependencies
SR-007–008	Inspect	README complete, examples work
SR-009	Demo	Container builds and runs
SR-010	Demo	Connect to self-hosted instance
SR-011–015	Test	Parser/API tests pass

Detailed verification procedures in Section 11.8. Operational concept and use cases in Section 10.2.

Chapter 9

System Requirements Specification

9.1 Overview

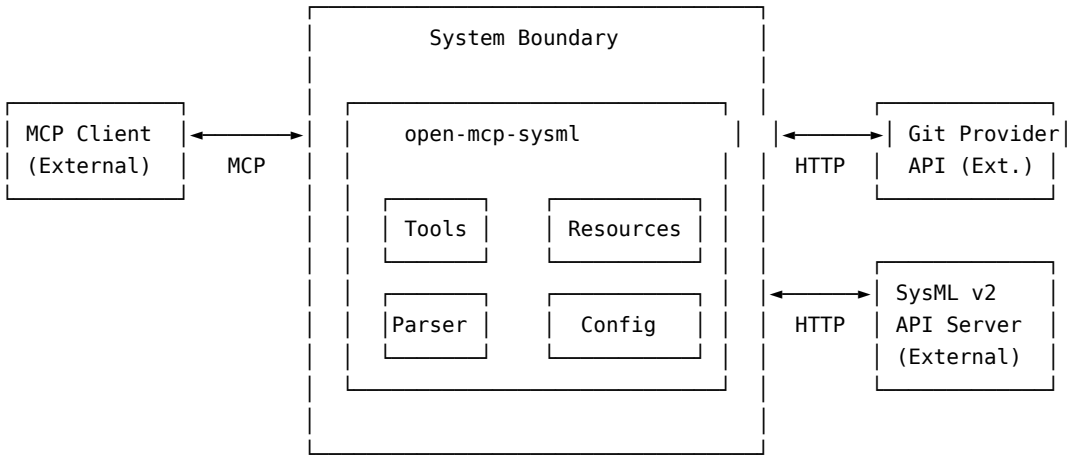
This chapter defines the system requirements for the open-mcp-sysml server per [1, Sec. 2.3.5.3]. Requirements are organized by functional area and traced to stakeholder requirements.

9.2 System Scope and Boundary

9.2.1 System Definition

The open-mcp-sysml server is a software system that implements the Model Context Protocol (MCP) to enable AI-augmented Model-Based Systems Engineering (MBSE) workflows. It provides AI assistants with programmatic access to SysML v2 models stored in Git repositories, with intelligent context management to optimize token usage. The design is provider-agnostic, with GitLab as the reference implementation.

9.2.2 System Boundary



9.2.3 External Interfaces

Interface	Type	Protocol	Description
MCP Client	Input	MCP over stdio/HTTP	AI assistant sending requests
Git Provider API	Output	REST/HTTP	Repository file operations (GitLab reference)
SysML v2 API	Output	REST/HTTP	Model validation and queries

Interface	Type	Protocol	Description
Configuration	Input	Environment variables	Server configuration

> **Interface Requirements (Tailored)**
>
> Per INCOSE Handbook 4.3.4, formal IR-xxx interface requirements are tailored for this project. External interfaces (MCP protocol, Git provider API, SysML v2 API, configuration) are documented in the Architecture Design Description (Section 10.10) rather than duplicated here. This avoids information duplication while maintaining traceability through the RTM (Section 16.3).

9.3 Functional Requirements

9.3.1 Model Context Protocol (MCP)

	Requirement	Priority	Verification	Trace
MCP-	The server SHALL implement MCP protocol version 2024-11-05	High	Test	SR-001
MCP-	The server SHALL support stdio transport	High	Test	SR-006
MCP-	The server SHALL support HTTP transport	Medium	Test	SR-003
MCP-	The server SHALL respond to initialize requests with server capabilities	High	Test	SR-001
MCP-	The server SHALL list available tools via tools/list	High	Test	SR-001
MCP-	The server SHALL list available resources via resources/list	High	Test	SR-001

9.3.2 Repository Integration

9.3.3 SysML v2 Operations

9.4 Non-Functional Requirements

9.4.1 Performance

ID	Requirement	Priority	Verification	Trace
NFR-PERF-001	The server SHALL respond to tool calls within 5 seconds under normal network conditions	Medium	Test	-
NFR-PERF-002	The server SHALL handle SysML v2 files up to 1MB in size	Medium	Test	-

9.4.2 Security

ID	Requirement	Priority	Verification	Trace
FR-REPO-001	The server SHALL read files from Git repositories	High	Test	SR-001
FR-REPO-002	The server SHALL list .sysml files in a repository directory	High	Test	SR-001
FR-REPO-003	The server SHALL support gitlab.com as a reference target	High	Test	SR-001
FR-REPO-004	The server SHALL support self-hosted Git providers via configurable base URL	Medium	Test	SR-010
FR-REPO-005	The server SHALL authenticate using Personal Access Token	High	Test	SR-001
FR-REPO-006	The server SHALL commit file changes to Git repositories	Medium	Test	SR-001
FR-REPO-007	The server SHALL create merge/pull requests	Low	Test	SR-001

ID	Requirement	Priority	Verification	Trace
FR-SYS-001	The server SHALL parse SysML v2 textual notation per the grammar subset defined in FR-SYS-008	High	Test	SR-012 SR-0
FR-SYS-002	The server SHALL extract element names and types from parsed models	High	Test	SR-012
FR-SYS-003	The server SHALL validate SysML v2 syntax via API server when available	Medium	Test	SR-011
FR-SYS-004	The server SHALL query model elements by type via API server	Medium	Test	SR-012
FR-SYS-005	The server SHALL provide bundled example SysML v2 models	Low	Inspection	SR-008
FR-SYS-006	The server SHALL parse SysML v2 package, part, action, requirement, and item definitions per OMG SysML v2 grammar	High	Test	SR-013
FR-SYS-007	The server SHALL report parse errors with file path, line number, and column number	High	Test	SR-014
FR-SYS-008	The server SHALL document supported SysML v2 grammar subset with coverage matrix	Medium	Inspection	SR-015
FR-SYS-009	The grammar SHALL be validated against the GfSE SysML v2 model collection as an external test corpus	Medium	Test	SR-013

ID	Requirement	Priority	Verification	Trace
NFR-SEC-001	The server SHALL NOT log authentication tokens to any output	High	Inspection	-
NFR-SEC-002	The server SHALL support configuration via environment variables for secrets	High	Test	-
NFR-SEC-003	The server SHALL validate all input parameters to prevent injection attacks	High	Test	-

9.4.3 Deployment

ID	Requirement	Priority	Verification	Trace
NFR-DEP-001	The server SHALL be distributable as a single static binary with no external runtime dependencies	High	Demonstration	SR-006
NFR-DEP-002	The server SHALL be distributable as an OCI-compliant container image	High	Demonstration	SR-009
NFR-DEP-003	The server SHALL support Linux operating systems (amd64, arm64 architectures)	High	Test	SR-006
NFR-DEP-004	The server SHALL support macOS operating systems (amd64, arm64 architectures)	High	Test	SR-006

9.4.4 Documentation

ID	Requirement	Priority	Verification	Trace
NFR-DOC-001	The software repository SHALL include README with installation instructions	High	Inspection	SR-007
NFR-DOC-002	The software repository SHALL include usage examples	High	Inspection	SR-008
NFR-DOC-003	The software repository SHALL include CONTRIBUTING guide	Medium	Inspection	SR-002

9.5 Constraints and Assumptions

9.5.1 Design Constraints

ID	Constraint	Rationale
DC-001	The server SHALL be implemented in Rust	GKG alignment, memory safety, single static binary
DC-002	The server SHALL use the official Rust MCP SDK (rmcp)	Ensures protocol compliance, official SDK
DC-003	The server SHALL use a provider-agnostic repository interface	Future GKG integration, extensibility
DC-004	Container builds SHALL use Buildah/Podman	OCI-compliant, rootless, CI-friendly
DC-005	The parser SHALL maintain a documented grammar coverage matrix	Transparency about spec compliance

9.5.2 Operational Constraints

ID	Constraint	Impact
OC-001	SysML v2 API server is an optional dependency	Basic parsing works offline; validation requires API
OC-002	Container testing limited to CI environment	macOS development cannot test containers locally
OC-003	Git provider PAT required for private repositories	Public repos accessible without authentication

9.5.3 Assumptions

ID	Assumption	Risk if Invalid
A-001	MCP protocol spec stable through project duration	May require protocol updates
A-002	SysML v2 API spec stable (July 2025 OMG adoption)	May require API client changes
A-003	Rust MCP SDK (rmcp) supports required features	May need SDK contributions or workarounds
A-004	Git provider APIs stable for file operations	Low risk - mature APIs

9.6 Verification Methods

Per [1, Sec. 2.3.5.9], each requirement has an assigned verification method:

Method	Code	Description
Inspection	I	Visual examination of documentation, code
Analysis	A	Mathematical or logical evaluation
Demonstration	D	Functional operation without quantitative measurement
Test	T	Execution with quantitative measurement and pass/fail criteria

9.6.1 Verification Summary

Category	Test	Demonstration	Inspection	Analysis	Total
FR-MCP	6	0	0	0	6

Category	Test	Demonstration	Inspection	Analysis	Total
FR-REPO	7	0	0	0	7
FR-SYS	7	0	2	0	9
NFR-PERF	2	0	0	0	2
NFR-SEC	2	0	1	0	3
NFR-DEP	2	2	0	0	4
NFR-DOC	0	0	3	0	3
Total	**26**	**2**	**6**	**0**	**34**

9.7 Requirements Analysis

Per [1, Sec. 2.3.5.3], requirements must be analyzed for completeness, consistency, and feasibility.

9.7.1 Completeness Check

Criterion	Status	Notes
All stakeholder requirements traced		See traceability matrix
All functional areas covered		MCP, GitLab, SysML operations
NFRs address FURPS+		Performance, Security, Deployment, Documentation
Verification method assigned		All requirements have verification
Priority assigned		High/Medium/Low for all

9.7.2 Consistency Check

Criterion	Status	Notes
No contradictory requirements		Reviewed for conflicts
Terminology consistent		Glossary in Appendix A
Units/formats consistent		SI units, ISO date formats

9.7.3 Feasibility Assessment

Requirement Area	Feasibility	Risk
MCP Protocol	High	Official rmcp SDK provides implementation
Repository Integration	High	Provider-agnostic trait with GitLab implementation
SysML v2 Parsing (Subset)	High	Documented grammar subset achievable; see Section 12.2
SysML v2 Parsing (Full)	Low	Full compliance requires Xtext port or JVM interop; future work

Requirement Area	Feasibility	Risk
SysML v2 API	Medium	Depends on API server availability
Container Deployment	High	Standard Rust cross-compilation

9.7.4 TBD Items

Item	Original Target	Disposition
OAuth authentication scope	PDR (Week 4)	Deferred — Phase 1 uses PAT authentication only (FR-REPO-005)
SysML v2 API error handling patterns	Week 7	Deferred — SysML v2 API integration deferred to post-capstone (R1 mitigation)
HTTP transport security (TLS) requirements	PDR (Week 4)	Deferred — HTTP transport is Phase 2+ scope; Phase 1 uses stdio only

9.8 Tool Definitions

9.8.1 Implemented (Phase 1 Complete)

Tool	Description	Status
`sysml_parse`	Parse SysML v2 text and extract elements with L0/L1/L2 detail levels	**Complete**
`sysml_validate`	Validate SysML v2 syntax via tree-sitter parse diagnostics	**Complete**
`sysml_list_definitions`	List definition names and types in SysML v2 text	**Complete**
`repo_list_files`	List files in Git repository (with optional `.sysml` filter)	**Complete**
`repo_get_file`	Read file content from Git repository	**Complete**

9.8.2 Planned (Phase 2+)

Tool	Description	Status
`sysml_validate` (API)	Full semantic validation via SysML v2 API server	Planned (extends current syntax validation)
`sysml_query`	Query model elements by type/properties	Planned
`repo_commit`	Commit changes to repository	Planned
`repo_create_mr`	Create merge/pull request	Planned

9.9 Resource Definitions

Resource URI	Phase	Description
`sysml://examples/{name}`	0	Bundled example models
`repo://{project}/file/{path}`	1	Git repository file access
`sysml://projects`	2	SysML v2 API project list

Chapter 10

Architecture Design Description

The architecture of this system is driven by a single constraint: context windows are scarce. Modern LLMs process 100K+ tokens, but effective reasoning requires focused context. Per [4], "Context Engineering" is the discipline of optimizing LLM inputs for quality outputs.

10.1 Context Management as Architectural Driver

Per [1, Sec. 2.3.5.4], architecture decisions should trace to stakeholder concerns. The primary concern driving this architecture is **context efficiency**: enabling AI agents to reason effectively about SysML v2 models within constrained token budgets.

10.1.1 The Problem

A typical systems engineering project context consumes 40K+ tokens for project awareness (requirements, architecture decisions, constraints, current task state). This leaves limited tokens for model reasoning. When an AI agent must also understand a SysML v2 model, naive approaches (loading entire files) quickly exhaust available context.

10.1.2 Four Context Management Strategies

Literature analysis (Section 3.1) identified four proven strategies:

Strategy	Description	Example Pattern
Avoidance	Minimize context needs through tiny, focused prompts	Sentence-level LLM calls [11]
Staged Decomposition	Pipeline with intermediate representations	JSON between stages [10]
Progressive Narrowing	Similarity + reachability-based pruning	Subgraph extraction [12]
Multi-Agent Partitioning	Divide context across specialized agents	Pipeline agents

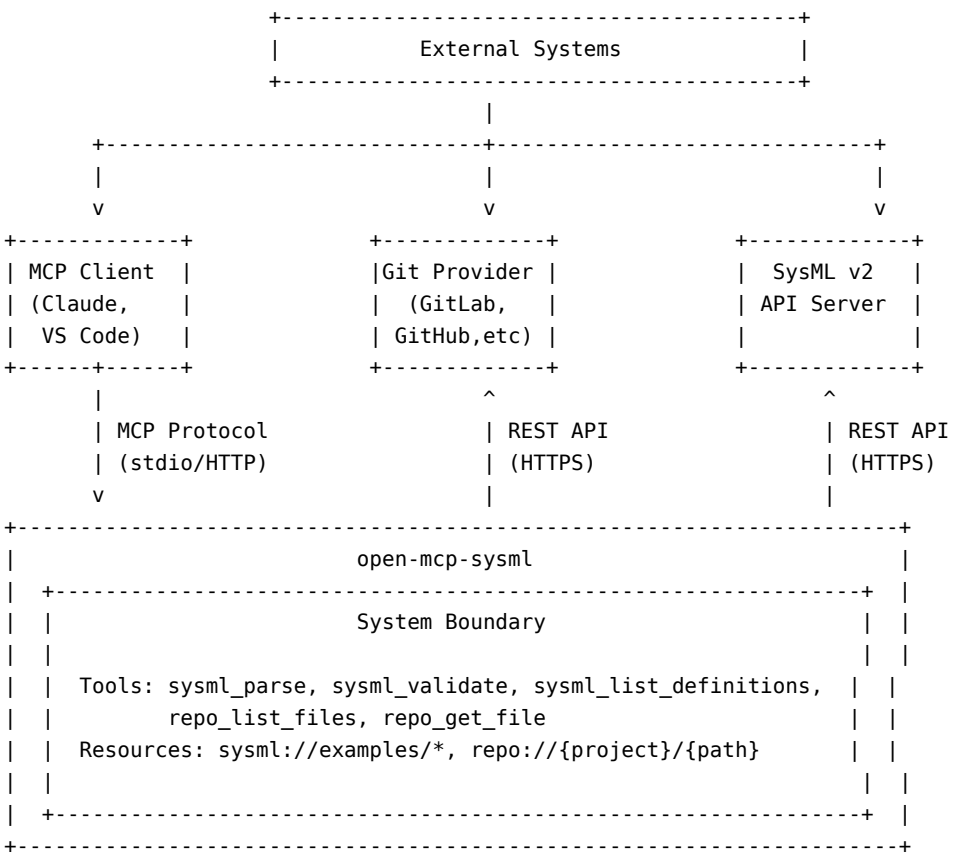
10.1.3 Architectural Response

This system enables all four strategies:

1. **Fine-grained query tools** support avoidance (get exactly what you need)
2. **JSON intermediate representation** enables staged decomposition
3. **Subgraph extraction** implements progressive narrowing
4. **Tool design** supports multi-agent workflows via stateless calls

10.2 System Context Diagram

Per [1, Sec. 2.3.5.4], the context diagram defines the system boundary and external interfaces.



External Interfaces:

Interface	Protocol	Direction	Description
MCP Client	MCP 2024-11-05 (stdio/HTTP)	Bidirectional	AI tool integration
Git Provider API	REST (HTTPS)	Outbound	Repository file access (provider-agnostic)
SysML v2 API	REST (HTTPS)	Outbound	Model query and validation (planned)

10.2.1 Operational Modes

Mode	Transport	Use Case	Authentication
Local Development	stdio	Individual engineer with Claude/VS Code	Git provider PAT
Team Server	HTTP (planned)	Shared server for team access	PAT per request
CI/CD Pipeline	HTTP (planned)	Automated validation	CI job token

10.2.2 Use Cases

UC-1: AI-Assisted Model Review

1. Engineer opens Claude Desktop with MCP server configured
2. Asks: "List all requirement definitions in the vehicle model"
3. MCP server fetches model via `repo_get_file`, parses via `sysml_parse`
4. Claude presents findings with `sysml_list_definitions` and suggests improvements
5. Engineer requests changes via iterative tool calls

UC-2: Model Validation in CI/CD

1. Developer commits SysML v2 model changes
2. CI pipeline calls `sysml_validate` via stdio
3. Validation results reported in merge request

UC-3: Exploratory Model Query

1. New team member asks natural language questions about model structure
2. MCP server uses `sysml_query` to search elements
3. AI explains model architecture and relationships

10.3 Novel Contributions

This project addresses gaps in the open-source MBSE ecosystem. While proprietary solutions exist, the community lacks accessible building blocks for AI-augmented systems engineering.

Contribution	Gap Addressed	Community Benefit
tree-sitter-sysml	No open-source SysML v2 grammar in tree-sitter ecosystem	Syntax highlighting, IDE support, MCP integration
kebnf-to-tree-sitter	Manual grammar maintenance as OMG spec evolves	Automated updates, formal traceability
open-mcp-sysml	No MBSE MCP server in 7,364+ public repos	AI-augmented MBSE workflows

10.3.1 tree-sitter-sysml

The first open-source SysML v2 grammar for tree-sitter, enabling:

- Syntax highlighting in any tree-sitter-compatible editor
- Incremental parsing for responsive IDE experiences
- WASM compilation for browser-based tooling
- Contribution path to tree-sitter GitHub organization
- GitLab contribution path via vendor/grammars/

Status: Production ready. 125 corpus tests passing, 99.6% coverage across 275 external files (OMG Training, OMG Examples, GfSE, Advent of SysML v2). Context-sensitive definition bodies implemented. Technical debt documented.

10.3.2 kebnf-to-tree-sitter

Automated grammar generation from OMG KEBNF specifications:

- One-shot conversion preserves spec traceability
- Semantic mapping document captures conversion decisions
- Enables automated updates when OMG publishes new specs
- Research contribution on grammar transposition for MBSE languages

Status: Parser and emitter complete (640/640 KEBNF rules). Generated grammar has 335+ conflicts requiring resolution. Iterative conflict resolution strategy selected—fixing conflicts one at a time with documented rationale for INCOSE paper.

10.3.3 open-mcp-sysml

MCP server enabling AI agents to work with SysML v2 models:

- Fine-grained query tools for context-efficient access
- Provider-agnostic repository access (GitLab as reference implementation)
- Tree-sitter integration for reliable parsing
- Stateless tool design supporting multi-agent workflows

Status: Phase 1 Complete (Feb 13, 2026). Tree-sitter integration working with 5 MCP tools: `sysml_parse` (L0/L1/L2 detail levels), `sysml_validate`, `sysml_list_definitions`, `repo_list_files`, `repo_get_file`. 22 tests passing. Phase 2 PRD ready for 7 token reduction strategies. Ready for benchmark vignette testing.

10.3.4 sysml-grammar-benchmark (NEW)

Automated grammar benchmark dashboard comparing SysML v2 parser implementations:

- Objective comparison across standardized test corpora
- GitLab CI automation with scheduled runs
- Quarto + D3 static dashboard on GitLab Pages
- Community contribution path for new parsers and corpora

Status: Repository scaffolded with PRD. Dashboard and CI pipeline pending implementation.

10.4 Architecture Alternatives

Per [1, Sec. 2.3.5.4], candidate architectures were evaluated before selection.

Alternative	Evaluation
Python + FastMCP	Rejected: additional runtime dependency
TypeScript + official SDK	Rejected: heavier deployment footprint
Go + go-sdk	Initially selected, then superseded
Rust + rmcp + tree-sitter	**Selected**: GKG alignment, community grammar

Parser Architecture Pivot: Replaced planned hand-rolled parser with tree-sitter grammar. Key factors:

Factor	Hand-rolled	Tree-sitter
Error recovery	Must implement	Built-in
Incremental parsing	Must implement	Built-in
Multi-platform	Rust only	C, WASM, Rust, Go, Python, Swift
Community reuse	Project-specific	tree-sitter org contribution
GitLab integration	N/A	vendor/grammars/ path

10.5 Technology Stack

Component	Technology	Rationale
Language	Rust 1.85+	GKG alignment, memory safety, single static binary
MCP SDK	rmcp (official)	Official Rust SDK, tokio async runtime
Git Client	Provider-agnostic trait	Future GKG integration; GitLab as first implementation
Parser	tree-sitter-sysml	Community grammar, multi-platform

Component	Technology	Rationale
Transport	stdio (HTTP planned)	stdio for local dev; HTTP transport planned for Phase 2
Container	Buildah/Podman	OCI-compliant, rootless, CI-friendly
Documentation	Quarto	Markdown-native, GitLab Pages compatible

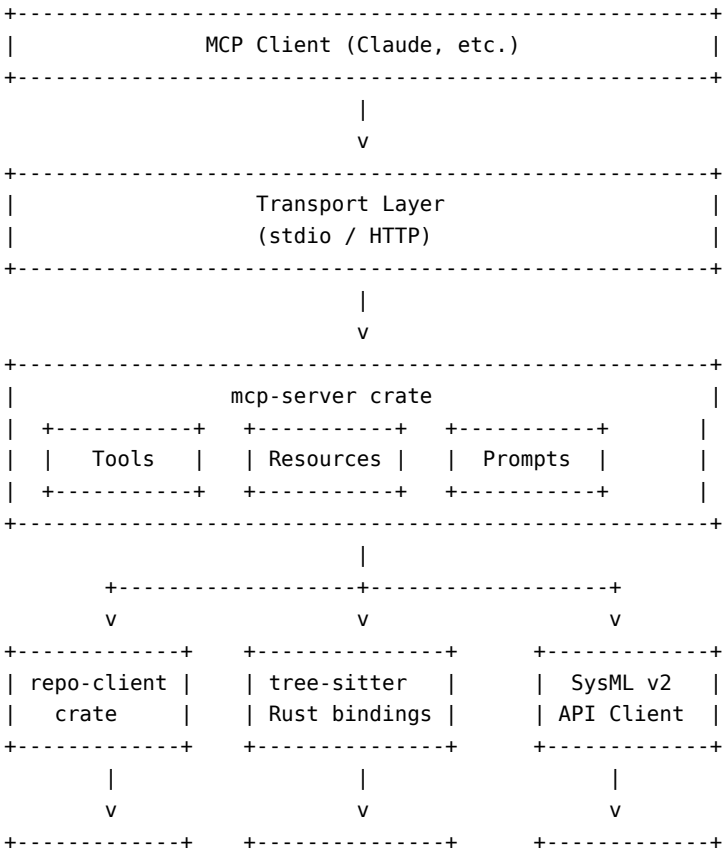
10.6 Repository Structure

```
gitlab.com/dunn.dev/open-mcp-sysml/    # GitLab Group (monorepo root)
+-- capstone/                           # SE Documentation (Quarto Book)
+-- open-mcp-sysml/                     # Rust MCP Server
|   +-- crates/
|       +-- sysml-parser/              # tree-sitter wrapper (parse, validate, list)
|       +-- repo-client/               # Provider-agnostic Git interface
|       +-- mcp-server/                # MCP server binary
+-- tree-sitter-sysml/                 # Brute-Force Grammar (99.6% coverage)
+-- kebnf-to-tree-sitter/              # Spec-Driven Grammar Converter
+-- sysml-grammar-benchmark/           # Grammar Comparison Dashboard (NEW)
```

Key Design Decisions:

- **Dual-path grammar strategy**: Brute-force for immediate value, spec-driven for research
- `open-mcp-sysml` consumes grammar via Rust bindings
- Dual CI: GitHub Actions for tree-sitter ecosystem, GitLab CI for coverage tracking

10.7 Component Architecture



```
|Git Provider |    |tree-sitter-   |      | SysML v2    |
| API         |    |sysml grammar  |      | API Server  |
+-------------+    +---------------+      +-------------+
```

Component Responsibilities:

Component	Responsibility
mcp-server	MCP protocol handling, tool dispatch
sysml-parser	Rust API wrapping tree-sitter for parse, validate, list_definitions
repo-client	Provider-agnostic Git operations (GitLab reference implementation)
tree-sitter-sysml	Grammar definition, generated parser

10.8 Context-Aware Tool Design

Per literature recommendations, tools are organized by context cost. The implemented tools (Phase 1) prioritize low-context operations; future phases will add higher-context analysis and generation tools.

10.8.1 Implemented Tools (Phase 1)

```yaml
sysml_parse:
  description: Parse SysML v2 text and extract elements
  input:
    source: string
    detail_level: "L0" | "L1" | "L2"  # L0: names only, L1: +types, L2: full AST
  output: Element[]
  context_cost: L0 ~100 tokens, L1 ~300 tokens, L2 ~2000 tokens

sysml_validate:
  description: Validate SysML v2 syntax via tree-sitter parse diagnostics
  input:
    source: string
  output: ValidationResult (errors, warnings)
  context_cost: ~300 tokens

sysml_list_definitions:
  description: List all definitions in SysML v2 text
  input:
    source: string
  output: Definition[] (name, type)
  context_cost: ~200 tokens

repo_list_files:
  description: List files in Git repository
  input:
    project: string
    path: string?
    ref: string?
    sysml_only: boolean?
  output: FileEntry[]
  context_cost: ~300 tokens

repo_get_file:
  description: Read file content from Git repository
  input:
    project: string
```

```
    path: string
    ref: string?
  output: FileContent
  context_cost: varies by file size
```

10.8.2 Planned Tools (Phase 2+)

```
sysml_query:
  description: Query elements by type/properties
  status: Planned

repo_commit:
  description: Commit file changes to repository
  status: Planned
```

Design Principles:

- **Stateless calls**: No session pollution between tool invocations
- **Server-side pruning**: Return minimal sufficient context
- **JSON intermediate representation**: Enable staged decomposition
- **Syntax validation**: Catch errors early on all generated content

10.9 Implementation Patterns for Progressive Context Budgeting

The tool design principles in Section 10.8 establish *what* to build; this section addresses *how* to implement these patterns using proven techniques from the rapidly evolving ecosystem of MCP servers and AI agents. These patterns emerge primarily from practitioner implementations rather than academic literature—many creators are building rapidly without publishing formal papers.

OpenAI's "Harness Engineering" report [5] provides production-scale validation of these patterns: a team built a million-line product with zero manually-written code by treating repository-local knowledge as the system of record, enforcing architectural invariants mechanically, and using progressive disclosure (a short AGENTS.md as map, not encyclopedia). Their experience confirms that context management patterns are not academic abstractions but production necessities.

This survey reflects the ecosystem as of February 2026. Given the pace of change, we recommend revisiting these patterns monthly.

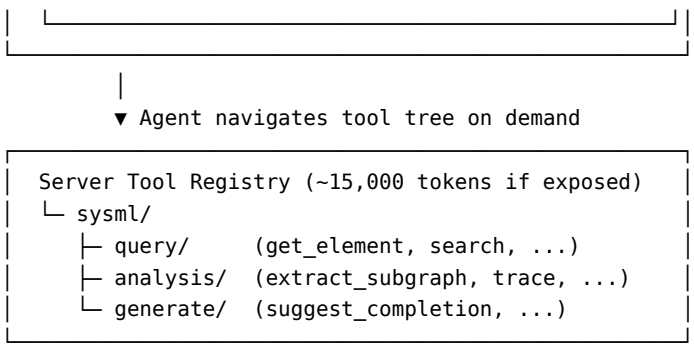
10.9.1 MCP Server Patterns

Three implementation patterns have emerged for managing context budgets within MCP tool servers, each offering 90%+ token reduction from naive implementations.

10.9.1.1 Two Meta-Tool Architecture

Instead of exposing all tools directly to the agent (consuming 15,000+ tokens), expose only two meta-tools [26]:

```
┌─────────────────────────────────────────────┐
│  Client Context Window                      │
├─────────────────────────────────────────────┤
│  Tools available: 2 (~800 tokens)           │
│  ┌─────────────────────────────────────────┐│
│  │ get_tools_in_category(category_path)    ││
│  │ → Returns tool names + descriptions in category ││
│  └─────────────────────────────────────────┘│
│  ┌─────────────────────────────────────────┐│
│  │ execute_tool(tool_path, parameters)     ││
│  │ → Executes any tool by hierarchical path ││
```



Token savings: ~95% (15,000 → 800 tokens) **Trade-off**: Adds one round-trip for tool discovery per category

10.9.1.2 Cache ID + Summary Pattern

Return concise summaries with cache identifiers; provide details on demand [27]:

```
Phase 1: Query returns summary + cache ID
─────────────────────────────────────────

sysml_list_elements(package="Vehicle")
→ {
    cacheId: "elem-abc123",
    summary: { totalElements: 47, parts: 12, ports: 8, ... },
    quickAccess: [ "Vehicle::PowerSubsystem", "Vehicle::Chassis", ... ]
  }
  (~500 tokens)

Phase 2: Details fetched only when needed
─────────────────────────────────────────

sysml_get_details(cacheId: "elem-abc123", filter: "parts-only")
→ { elements: [ ... full element definitions ... ] }
  (~2000 tokens, only when required)
```

Token savings: ~97% for list operations (18,700 → 540 tokens baseline)

Complementary patterns:

- **RTFM documentation**: Tool descriptions minimal (<10 words); agents call `rtfm(toolName)` for full documentation on demand
- **`defer_loading` flag**: Platform-native lazy discovery (Claude-specific)

10.9.1.3 L0/L1/L2 Tiered Loading

Filesystem-inspired progressive context loading [28]:

Tier	Content	Tokens	Use Case
L0 (Abstract)	Names + one-line descriptions	~100	Quick relevance check
L1 (Overview)	Structure, relationships, key properties	~2,000	Understand organization
L2 (Details)	Full element definitions, constraints	Full	Deep analysis/editing

Application to SysML: L0 returns qualified names; L1 returns element skeletons with relationship counts; L2 returns full textual notation with documentation.

10.9.2 Complementary Techniques

Beyond MCP-specific patterns, several techniques optimize context at other layers.

10.9.2.1 Prompt Compression

LLMLingua [29] uses small language models (GPT-2, LLaMA-7B) to identify and remove non-essential tokens before the main LLM call.

Performance: Up to 20x compression with minimal quality loss **RAG improvement**: +21.4% accuracy using only 1/4 tokens **Application**: Client-side preprocessing of large retrieved documents

10.9.2.2 KV-Cache-First Design

Manus's production experience [30] identifies KV-cache hit rate as "the single most important metric for a production AI agent":

- **Append-only context**: Never modify previous actions/observations
- **Stable prefixes**: Avoid timestamps or dynamic content at prompt start
- **Tool masking**: Constrain action space via logit masking, not tool removal

Cost impact: 10x difference between cached ($0.30/MTok) and uncached ($3.00/MTok)

10.9.2.3 Frequent Intentional Compaction

HumanLayer's workflow [31] structures development around context management:

1. **Research**: Subagent explores codebase, returns compressed findings
2. **Plan**: Outline steps with verification criteria (human reviews here)
3. **Implement**: Execute plan phase-by-phase, compacting state between phases

Key insight: "A bad line of research could lead to thousands of bad lines of code"—focus human review on research and plans, not just generated code.

Context utilization: Maintain 40-60% window utilization for optimal reasoning.

10.9.3 Performance Metrics Framework

The following metrics enable comparison across implementations:

Category	Metric	Baseline Examples	Source
Token efficiency	Compression ratio	95-97% (MCP patterns)	[26], [27]
Cost impact	Cached vs uncached tokens	10x difference	[30]
Latency	Tool response time	120ms (semantic) vs 2000ms (visual)	[27]
Quality	Task completion rate	24%→51% with optimization	DSPy MIPROv2
Context utilization	% window used effectively	40-60% sweet spot	[31]
Multi-agent gain	Single vs multi-agent success	90.2% improvement	[32]

10.9.4 Implementation Options for open-mcp-sysml

Based on the patterns surveyed, the following implementation options are available, each addressing different aspects of context efficiency:

Cache ID + Summary

All list/query tools return summaries with cache IDs. Add `sysml_get_details(cacheId, detailType)` for on-demand expansion. Expected savings: ~90% on common workflows.

L0/L1/L2 Tiered Responses

Add `detail_level` parameter to existing tools. Default to L1; agents request L2 only when editing.

RTFM Documentation Pattern

Minimize tool descriptions to <10 words. Add `sysml_rtfm(toolName)` for full documentation.

Two Meta-Tool Wrapper

Useful if tool count exceeds ~20. Consider as enhancement once tool surface stabilizes.

KV-Cache Optimization

Design response formats for append-only workflows. Ensure deterministic JSON serialization.

10.9.5 Reference Implementations

Repository	Stars	Pattern	Language	URL
mcp-proxy	3	Two meta-tool	Go	github.com/IAMSamuell proxy
xc-mcp	59	Cache ID + RTFM + defer_loading	TypeScript	github.com/conorluddy/ mcp
OpenViking	1,100	L0/L1/L2 tiered loading	Python	github.com/volcengine/C
LLMLingua	5,800	Prompt compression	Python	github.com/microsoft/LL
12-factor-agents	18,200	Workflow patterns	Docs	github.com/humanlayer/ factor- agents
Beads	16,000	Git-backed agent memory	Go	github.com/steveyegge/b
OpenAI Harness	—	Progressive disclosure + golden principles	Internal	[5]

10.10 Interface Definitions

10.10.1 MCP Protocol Interface

The server implements MCP 2024-11-05:

- `initialize` - Protocol handshake
- `tools/list` - Enumerate available tools
- `tools/call` - Execute a tool
- `resources/list` - Enumerate resources
- `resources/read` - Read a resource

10.10.2 Repository Interface Schema

```
{
  "RepoClient": {
    "methods": {
      "read_file": {
        "params": ["project", "path", "ref"],
        "returns": "bytes"
      },
      "list_files": {
        "params": ["project", "path", "ref"],
        "returns": "FileEntry[]"
      }
```

```
    }
  }
}
```

10.10.3 Parser Interface Schema

```
{
  "sysml_parse": {
    "input": {
      "content": "string"
    },
    "output": {
      "success": "boolean",
      "tree": "ParseTree?",
      "errors": "ParseError[]"
    }
  }
}
```

10.11 Dual-Path Grammar Strategy

Both grammar development paths are actively maintained:

Path	Repository	Purpose	Status
Brute Force	tree-sitter-sysml	Practical parsing, MCP server	Tier 1 complete
Spec-Driven	kebnf-to-tree-sitter	Formal compliance, INCOSE paper	Tool complete, grammar in progress

Cross-validation: Comparing outputs identifies spec interpretation errors in brute-force and practical parsing issues in spec-driven output.

Why both matter:

- Brute-force provides immediate practical value
- Spec-driven enables automated updates and formal traceability
- Comparison validates both approaches

10.12 Requirements Allocation

Per [1, Sec. 2.3.5.4], requirements are allocated to architecture elements.

Requirement	Architecture Element	Component
FR-MCP-001, FR-MCP-002	MCP Server	mcp-server crate
FR-MCP-003	HTTP Transport	mcp-server crate
FR-REPO-001, FR-REPO-002	Repo Client	repo-client crate
FR-SYS-001, FR-SYS-002	SysML Parser	tree-sitter-sysml
NFR-DEP-001	Build Configuration	Cargo.toml
NFR-DEP-002	Container Image	Containerfile

10.13 Deployment Architecture

10.13.1 Deployment Modes

Mode	Transport	Use Case	Configuration
Local Development	stdio	Claude Desktop, VS Code	`--transport stdio`
CI/CD Integration	HTTP	GitLab CI services	`--transport http --port 8080`
Container	HTTP	Production deployment	Docker/Podman with port mapping

10.13.2 Development Constraints

Constraint: No local container builds on macOS (no podman machine).

Mitigation:

- Local development uses `cargo build` and `cargo test` directly
- MCP protocol testing via stdio (no containers required)
- Container builds run exclusively in GitLab CI

10.14 CI/CD Pipeline

GitLab Ultimate features leveraged:

Feature	Purpose
SAST	Static Application Security Testing for Rust
Dependency Scanning	Scan Cargo.lock for vulnerabilities
Secret Detection	Prevent accidental credential commits
License Compliance	Track crate licenses
Code Quality	Clippy integration for Rust linting

10.14.1 tree-sitter-sysml CI/CD

Dual CI for ecosystem compatibility:

Platform	Purpose
GitHub Actions	Tree-sitter org compatibility
GitLab CI	Coverage tracking, security scanning

Chapter 11

Verification & Validation Plan

11.1 Verification & Validation (V&V) Strategy

Per [1, Secs. 2.3.5.9, 2.3.5.11], this plan defines how we confirm the system meets requirements (verification) and stakeholder needs (validation).

The V&V strategy reflects the layered architecture of the system itself. The SysML v2 MCP server is built atop a tree-sitter grammar, wrapped in Rust crates, and exposed via the MCP protocol. Each layer has distinct failure modes and appropriate verification techniques: grammar correctness is best verified by corpus tests against known-good parse trees, Rust crate behavior by unit tests with the standard cargo test framework, and MCP protocol compliance by integration tests that exercise the full JSON-RPC message flow. This layered approach ensures defects are caught at the earliest possible stage — a grammar error surfaces in corpus testing before it can propagate to MCP tool responses.

The dual-CI strategy (GitHub Actions for tree-sitter-sysml, GitLab CI for the capstone ecosystem) reflects the grammar's dual contribution path: tree-sitter organization conventions require GitHub-hosted CI, while project-level coverage tracking and security scanning use GitLab Ultimate features.

Method	Scope	Environment
Corpus Testing	tree-sitter grammar constructs	Local (tree-sitter test)
Coverage Testing	Training file parse rate	GitLab CI
Unit Testing	Rust crates	Local (cargo test)
Integration Testing	MCP protocol compliance	Local (stdio)
Container Testing	Image builds, runtime	GitLab CI only
HTTP Transport Testing	Remote MCP connections	GitLab CI (service containers)
Acceptance Testing	End-to-end with Claude/VS Code	Local (stdio) + manual

System verification (this chapter) establishes that tools are functional and meet their requirements. The benchmark vignettes (Section C.1) then use these verified tools to evaluate AI-MBSE workflow effectiveness for the GVSETS publication — measuring whether MCP-enabled AI outperforms baseline approaches on real SE tasks. The two activities are complementary: verification is a prerequisite for meaningful benchmarking.

11.2 Verification Methods

Per [1, Sec. 2.3.5.9], verification uses IADT methods:

Method	Abbreviation	Description	When Used
Inspection	I	Visual examination of artifacts	Documentation, code review
Analysis	A	Mathematical/logical evaluation	Performance, security assessment
Demonstration	D	Functional operation shown	MCP protocol interaction
Test	T	Execution with defined inputs	Unit tests, integration tests

11.2.1 Verification Method Assignment

Test (T) dominates the verification method assignments below because this is a software-intensive system where most requirements are directly executable. The MCP protocol, repository integration, and SysML parsing requirements all produce observable, deterministic outputs given controlled inputs — making automated testing the most efficient and repeatable verification approach. Inspection (I) is reserved for documentation requirements where pass/fail is assessed by human review, and Demonstration (D) supplements testing for protocol compliance where showing a working client interaction provides additional confidence beyond unit-level assertions.

Requirement	Method	Rationale
FR-MCP-001	T, D	Test server initialization, demonstrate with client
FR-MCP-002, FR-MCP-005	T	Test tool enumeration and execution
FR-REPO-001, FR-REPO-002	T	Test file read from Git repositories
FR-SYS-001	T	Test parsing via tree-sitter corpus tests
FR-SYS-006	T	Test grammar subset via training file parse rate
FR-SYS-007	T	Test error recovery (tree-sitter ERROR nodes)
FR-SYS-008	I	Inspect tree-sitter-sysml README coverage docs
NFR-DEP-001	T, A	Test binary builds, analyze size
NFR-DEP-002	T	Test container builds in CI
NFR-DOC-001	I	Inspect Quarto output for completeness

Tailoring Note

The VMA table above covers the 11 highest-risk requirements that are directly verifiable through the current test infrastructure. The remaining 23 system requirements (covering HTTP transport, SysML v2 API integration, container deployment, and security) are deferred to post-Phase 1 verification as their corresponding features are implemented. Per INCOSE Handbook 4.3.4, this tailoring is appropriate for a software-intensive academic project where verification activities are prioritized by implementation phase.

11.3 Acceptance Criteria

Requirement Category	Verification Method	Acceptance Criteria
MCP Protocol Compliance	Integration test	Server initializes, lists tools/resources, executes tools
Repository Integration	Integration test	Read files from GitLab (reference) and self-hosted

Requirement Category	Verification Method	Acceptance Criteria
SysML v2 Validation	System test	Validates correct/incorrect SysML syntax
Container Deployment	CI pipeline	Image builds, runs, responds to MCP requests
Documentation	Inspection	Quarto renders, deploys to GitLab Pages

11.4 Enabling Systems

Per [1, Sec. 2.3.5.9], enabling systems support verification activities.

Enabling System	Purpose	Responsibility
tree-sitter CLI	Grammar testing (`tree-sitter test`)	Local + CI
Cargo Test Framework	Rust unit and integration testing	Built into Rust toolchain
GitLab CI/CD	Automated pipeline execution	GitLab SaaS runners
GitHub Actions	tree-sitter grammar CI	GitHub runners
Buildah/Podman	Container image builds	CI environment only
Claude Desktop	Manual acceptance testing	Local development
MCP Inspector	Protocol debugging	Local development
Quarto	Documentation builds	Local + CI

11.4.1 Test Environment Configuration

Environment	Transport	External Services	Use Case
Local Dev	stdio	Mocked/optional	Unit tests, rapid iteration
CI Test	stdio	Mocked	Automated test suite
CI Integration	HTTP	GitLab API (PAT)	Integration tests
CI Container	HTTP	Service containers	End-to-end container tests

11.5 Test Cases

11.5.1 MCP Protocol Tests

ID	Test Case	Expected Result	Method
TC-MCP-001	Send initialize request	Server responds with capabilities	T
TC-MCP-002	Request tools/list	Returns list including sysml_parse	T
TC-MCP-003	Call sysml_parse with valid SysML	Returns parsed elements	T
TC-MCP-004	Request resources/list	Returns example resources	T

ID	Test Case	Expected Result	Method
TC-MCP-005	Read sysml://examples/hello	Returns vehicle model content	T

11.5.2 Repository Integration Tests

ID	Test Case	Expected Result	Method
TC-REPO-001	Read file from public repo	Returns file content	T
TC-REPO-002	Read file with PAT auth	Returns file content	T
TC-REPO-003	List .sysml files in directory	Returns file list	T
TC-REPO-004	Read from self-hosted Git provider	Returns file content	T
TC-REPO-005	Handle non-existent file	Returns appropriate error	T

11.5.3 SysML Parsing Tests

11.5.3.1 tree-sitter Corpus Tests

ID	Test Case	Expected Result	Method
TC-SYS-001	Parse package declaration	Correct CST structure	T (corpus)
TC-SYS-002	Parse part definition	Correct CST structure	T (corpus)
TC-SYS-003	Parse requirement definition	Correct CST structure	T (corpus)
TC-SYS-004	Parse nested elements	Correct CST hierarchy	T (corpus)
TC-SYS-005	Parse with syntax errors	ERROR node in CST, partial parse	T (corpus)

11.5.3.2 Training File Coverage Tests

ID	Test Case	Expected Result	Method
TC-COV-001	Parse Module 01 files	Clean parse (no ERROR nodes)	T (CI)
TC-COV-002	Parse Module 02 files	Clean parse (no ERROR nodes)	T (CI)
TC-COV-003	Calculate overall parse rate	100% achieved (target was 10% Phase 1, 50% Phase 2)	A (CI)

11.6 Known Limitations

1. **Container testing**: Cannot be performed locally on macOS; relies on CI (risk R5, accepted)

2. **HTTP transport**: Requires CI service containers or Linux machine
3. **SysML v2 API**: Requires running API server; deferred to post-capstone. Basic parsing and repository operations work without API dependency
4. **Grammar coverage**: The tree-sitter-sysml grammar achieves 99.6% coverage across 275 external files (274/275) and 100% coverage of OMG training files (100/100). The single unparseable file uses non-standard UML syntax outside the SysML v2 specification. Full *semantic* compliance (type checking, import resolution) is out of scope for the tree-sitter grammar and deferred to future work (sysml.rs)

11.7 Continuous Integration/Continuous Delivery (CI/CD) Verification Pipeline

Per [1, Sec. 2.3.5.9], automated verification integrates into CI/CD.

11.7.1 Pipeline Stages

```yaml
stages:
  - lint
  - test
  - build
  - integration
  - publish
```

11.7.2 tree-sitter-sysml Test Stage

```yaml
test:
  stage: test
  image: node:20-alpine
  script:
    - npm ci
    - npx tree-sitter test
    - npx tree-sitter parse --quiet training/**/*.sysml 2>&1 | tee parse-results.txt
  artifacts:
    reports:
      metrics: coverage-metrics.txt
```

11.7.3 open-mcp-sysml Test Stage

```yaml
test:
  stage: test
  image: rust:1.85
  script:
    - cargo test --workspace
    - cargo clippy --workspace -- -D warnings
  rules:
    - if: $CI_PIPELINE_SOURCE == "merge_request_event"
    - if: $CI_COMMIT_BRANCH == "main"
```

11.7.4 Integration Test Stage

```yaml
integration:
  stage: integration
  image: rust:1.85
  variables:
```

```
    GITLAB_TOKEN: $CI_JOB_TOKEN
  script:
    - cargo test --workspace --features integration
  rules:
    - if: $CI_COMMIT_BRANCH == "main"
```

11.7.5 Container Test Stage

```
container-test:
  stage: test
  image: quay.io/buildah/stable
  services:
    - name: $CI_REGISTRY_IMAGE:$CI_COMMIT_SHORT_SHA
      alias: mcp-server
  script:
    - echo '{"jsonrpc":"2.0","id":1,"method":"initialize"...}' | nc mcp-server 8080
  rules:
    - if: $CI_COMMIT_BRANCH == "main"
```

11.8 Validation Approach

Per [1, Sec. 2.3.5.11], validation confirms the system meets stakeholder needs.

11.8.1 Validation Activities

Activity	Stakeholder Need	Method	Acceptance
End-to-end demo	AI tool integration	Demonstration	Claude reads SysML from GitLab
User acceptance	Developer experience	Interview	Positive feedback from pilot users
Paper submission	Academic validation	Peer review	GVSETS acceptance
Capstone review	Educational objectives	Review	Advisor approval

11.8.2 Validation Schedule

Milestone	Date	Validation Activity	Status
SRR	Feb 14, 2026	Requirements validated with stakeholders	**Complete**
PDR	Feb 14, 2026	Architecture validated against requirements (and by implementation)	**Complete**
CDR	Mar 29, 2026	Implementation validated, acceptance tests pass	Pending
Final	Apr 25, 2026	Stakeholder acceptance, capstone submission	Pending

11.9 Review Verification

Review	Verification Activities
SRR	Requirements complete, traceable to stakeholders
PDR	Architecture addresses all requirements
CDR	All tests pass, acceptance criteria met

Chapter 12

Implementation

12.1 Overview

Implementation pivoted in February 2026 to a tree-sitter grammar plus Rust MCP server architecture. This chapter documents progress organized by deliverable.

Deliverable	Status	Description
tree-sitter-sysml	Complete	Brute-force grammar, 99.6% external file coverage, 100% query coverage
kebnf-to-tree-sitter	In Progress	Spec-driven converter, 640 rules parsed, 335+ conflicts
open-mcp-sysml	Phase 1 Complete	5 MCP tools, 22 tests, tree-sitter integration working

12.2 tree-sitter-sysml

Production-ready SysML v2 grammar for tree-sitter ecosystem.

12.2.1 Status

Metric	Value
Grammar size	~2,236 lines
Corpus tests	125/125 passing
Training file coverage	100% (100/100 OMG examples)
External file coverage	99.6% (274/275 files)
Query coverage	100% (190/190 highlights, tags, folds)
Development time	~25 hours

12.2.2 Architecture

```
tree-sitter-sysml/
├── grammar.js              # Grammar definition
├── src/parser.c            # Generated parser
├── bindings/{c,go,node,python,rust,swift}/
├── corpus/                 # 125 test cases
└── .gitlab-ci.yml          # CI with coverage tracking
```

12.2.3 Methodology: Brute Force

The grammar was built through empirical iteration against training files:

1. Run coverage harness against 100 OMG training files
2. Identify failing files, inspect ERROR nodes
3. Implement missing construct in grammar.js
4. Generate parser, verify no regressions
5. Repeat until 100% coverage

This approach achieved rapid results (~10% → 100% in ~20 hours of focused grammar development, ~25 hours total including corpus and test infrastructure) but has limitations: - **Unknown specification alignment**: Grammar was reverse-engineered - **Likely over-acceptance**: May accept invalid syntax - **No semantic preservation**: AST structure not metamodel-aligned

12.2.4 Key Constructs Implemented

Category	Constructs
Definitions	part, item, port, action, state, requirement, constraint, use case, view, metadata, allocation
Usages	All corresponding usage forms plus event, exhibit, message, satisfy, verify
Behavioral	Actions, states, transitions, fork/join/decision, if/while/loop
Expressions	Operators, invocations, feature chains, select (`.?{}`), collect (`.{}`), units

12.2.5 Technical Debt Remediation

Session 5 addressed accumulated technical debt: - Split generic `relationship_part` into explicit specialization, subsetting, redefinition - Created context-sensitive body types for definitions (part_body vs action_body) - Added negative test framework (7/8 invalid patterns correctly rejected)

12.3 kebnf-to-tree-sitter

Automated converter from OMG KEBNF specifications to tree-sitter grammars.

12.3.1 Purpose

Goal	Description
Reproducibility	Re-generate grammar when SysML v2 spec updates
Traceability	Map tree-sitter rules to KEBNF source lines
Research	Quantify automation rates for INCOSE paper

12.3.2 Status

Metric	Value
KEBNF rules parsed	640/640 (100%)
Direct conversion	38% (247 rules)
Strip-and-convert	55% (353 rules)
Best-effort	6% (37 rules)
Manual review	<1% (3 rules)
LR conflicts remaining	335+

12.3.3 Methodology: Spec-Driven

The tool parses official KEBNF files and generates tree-sitter grammar:

1. **Parse**: Lex and parse KEBNF into AST (Chumsky 1.0)
2. **Classify**: Categorize each construct (direct, strip, approximate, manual)
3. **Transform**: Apply appropriate conversion strategy
4. **Emit**: Generate grammar.js with mapping document

KEBNF extends standard EBNF with metamodel annotations (type annotations, property assignments, cross-references). These are stripped for tree-sitter and recorded in `mapping.json` for downstream tools.

12.3.4 Conversion Categories

Category	%	Handling
Direct	38%	Basic syntax maps directly to tree-sitter
Strip & convert	55%	Remove annotations, keep syntax structure
Best-effort	6%	Approximate semantic actions
Manual review	<1%	Complex disambiguation needed

Overall: ~93% automated, ~7% requiring manual intervention

12.3.5 Architecture

```
kebnf-to-tree-sitter/
├── src/
│   ├── ast.rs            # KEBNF AST representation
│   ├── parser.rs         # KEBNF parser
│   ├── emitter.rs        # Tree-sitter grammar emitter
│   └── mapping.rs        # Semantic mapping generator
└── docs/
    ├── KEBNF-SPEC.md     # KEBNF format documentation
    └── MAPPING.md        # Semantic gap documentation
```

See Section B.3 for the INCOSE paper outlining the methodology.

12.4 open-mcp-sysml

Rust MCP server providing SysML v2 tools to AI assistants.

12.4.1 Status

Phase 1 complete (Feb 13, 2026). Tree-sitter integration working with 5 MCP tools and 22 tests passing. Phase 2 PRD ready for token reduction strategies.

Metric	Value
MCP tools	5 (sysml_parse, sysml_validate, sysml_list_definitions, repo_list_files, repo_get_file)
Tests	22 (4 unit + 4 integration + 10 protocol + 1 repo + 3 doc)
Training coverage	100% (100/100 files parse without errors)
Detail levels	L0 (names), L1 (structure), L2 (full)

12.4.2 Architecture

```
open-mcp-sysml/
├── Cargo.toml              # Workspace root
├── crates/
│   ├── sysml-parser/       # Tree-sitter wrapper, L0/L1/L2 detail levels
│   ├── repo-client/        # Provider-agnostic Git interface
│   └── mcp-server/         # MCP server binary (rmcp SDK)
└── tests/fixtures/sysml-v2/ # OMG training files (git submodule)
```

12.4.3 Implemented Tools

Tool	Description	Backend	Status
`sysml_parse`	Parse SysML v2 text with L0/L1/L2 detail	tree-sitter-sysml	Complete
`sysml_validate`	Return parse diagnostics	tree-sitter-sysml	Complete
`sysml_list_definitions`	List all definitions in model	tree-sitter-sysml	Complete
`repo_list_files`	List .sysml files in repository	repo-client (GitLab)	Complete
`repo_get_file`	Read file from repository	repo-client (GitLab)	Complete

12.4.4 Repository Client Interface

Provider-agnostic trait with GitLab as reference implementation:

```rust
pub trait RepoClient: Send + Sync {
    async fn read_file(&self, project: &str, path: &str, ref_: &str)
        -> Result<Vec<u8>, RepoError>;
    async fn list_files(&self, project: &str, path: &str, ref_: &str)
        -> Result<Vec<FileEntry>, RepoError>;
}
```

12.5 Dual-Path Grammar Strategy

Both grammar approaches provide value:

Aspect	tree-sitter-sysml	kebnf-to-tree-sitter
Approach	Empirical iteration	Spec conversion
Coverage	99.6% external files	100% KEBNF rules
Conflicts	Manually tuned	Auto-detected
Traceability	None	Full source mapping
Use case	MCP server, editors	Research, spec updates

Recommendation: Use brute-force grammar for immediate needs (syntax highlighting, MCP server). Use spec-driven for long-term maintenance and INCOSE paper contribution.

12.6 Lessons Learned

12.6.1 On Grammar Development

1. **Test harnesses matter**: Simple coverage metrics enabled rapid iteration
2. **Specification access crucial**: KEBNF files enabled systematic approach
3. **Semantic vs syntactic**: Tree-sitter handles parsing; semantics require additional layer

12.6.2 On Automation Assessment

Initial Assumption	Actual Result
40% fully automatable (estimated from KEBNF complexity analysis)	93% automated (38% direct + 55% strip-and-convert)
30% manual work required	7% manual intervention
60-100 hours effort	~12 hours

Key insight: Semantic annotations can be stripped and documented externally rather than requiring complex transformation.

12.6.3 On Practical vs Formal

- **Both have value**: Neither approach sufficient alone
- **Practical catches edge cases**: Specifications have gaps
- **Formal enables maintenance**: Reproducible > ad-hoc
- **Hybrid recommended**: Spec foundation + practical fixes

12.7 Next Steps

1. **Phase 2 token reduction**: Implement vanilla baseline, Cache ID + Summary, and remaining optimization strategies per PRD
2. **Benchmark execution**: Run V1, V4, V5 vignettes for GVSETS quantitative data
3. **kebnf conflict resolution**: Iteratively resolve remaining LR conflicts for INCOSE paper
4. **Grammar benchmark dashboard**: Wire tree-sitter adapter and corpus into sysml-grammar-benchmark

Chapter 13

Conclusions

13.1 SE Process Outcomes

This capstone applied INCOSE systems engineering processes [1] to a rapid software-intensive project: building the open source infrastructure required to demonstrate AI-augmented MBSE workflows for a GVSETS 2026 publication. The SE process was not an end in itself but a disciplined means of ensuring the software systems, grammar tools, and evaluation framework are built on defensible foundations.

13.1.1 Review Gate Completion

Both formal review gates were completed on February 14, 2026, with the full team (Technical Lead, SE Lead, and Advisor) participating:

Review	Target	Actual	Outcome
SRR	Jan 25	Feb 14	Complete with caveats
PDR	Feb 8	Feb 14	Complete with caveats

The two-week SRR delay reflected a deliberate choice: rather than reviewing requirements in isolation, we allowed implementation to inform and validate the requirements before baselining them. This "implementation-informed review" approach proved more effective than a traditional waterfall gate — by the time SRR was conducted, every stakeholder requirement had been tested against working software.

13.1.2 Tailoring for Academic Scope

Per INCOSE Handbook 4.3.4, the SE processes were tailored for a software-intensive academic project with a three-person team and 15-week timeline. Key tailoring decisions, each documented with rationale:

Tailored Element	Full INCOSE Process	Adopted Approach	Risk Accepted
Interface requirements	Formal IR-xxx identifiers in SyRS	Documented in ADD only	Potential traceability gap
Stakeholder validation	Formal interview protocol	Iterative development feedback	Less rigorous needs capture
Requirements evolution	Change control board after SRR	Baseline with understood mutability	Requirements may drift
Review formality	Separate SRR and PDR events	Combined assessment session	Less ceremony

13.1.3 Requirements Traceability

The RTM (Section 16.1) maintains four trace layers covering 15 stakeholder needs, 15 stakeholder requirements, 34 system requirements, and 22 implemented test cases. All functional requirements trace upward to stakeholder needs and downward to architecture elements and verification activities.

13.1.4 Risk Management

Of nine identified risks, four were closed by implementation evidence (R3, R4, R7, R8), one was mitigated by architectural decisions (R1), and one was mitigated after scope expansion proved manageable (R6). Three risks remain open: R2 (GVSETS deadline pressure), R5 (container testing blocked on macOS), and R9 (tree-sitter org acceptance uncertain). The most significant risk closure was R8 (external scanner): the grammar achieved 100% training file coverage without requiring an external scanner, eliminating a major technical uncertainty. See Section 6.8 for the full register.

13.2 Technical Outcomes

The SE process produced three operational software systems and validated a dual-path grammar strategy:

13.2.1 tree-sitter-sysml: Production-Ready Grammar

The brute-force grammar achieved results that exceeded initial expectations:

Metric	Target	Achieved
Corpus tests	100% pass	125/125 (100%)
Training file coverage	50%	100/100 (100%)
External file coverage	Not targeted	274/275 (99.6%)
Query coverage	Not targeted	190/190 (100%)
Development time	8 weeks planned	~25 hours actual

The grammar supports all major SysML v2 constructs: packages, parts, actions, states, requirements, constraints, verification cases, views, metadata, allocations, and expressions. Context-sensitive definition bodies (part_body vs action_body) reduce over-acceptance. Six language bindings (C, Rust, Go, Python, Node.js, Swift) enable broad ecosystem integration.

13.2.2 open-mcp-sysml: MCP Server Phase 1

The Rust MCP server delivers five operational tools with L0/L1/L2 detail levels designed for an estimated 80-97% token reduction from naive approaches (to be validated by benchmark vignettes V1/V4/V5):

Tool	Purpose	Token Impact
`sysml_parse`	Parse with configurable detail	L0: ~100 tokens, L2: full
`sysml_validate`	Return parse diagnostics	~300 tokens
`sysml_list_definitions`	List all definitions	~500 tokens
`repo_list_files`	List .sysml files in repository	~200 tokens
`repo_get_file`	Read file from repository	Variable

Twenty-two tests verify correctness across unit, integration, and MCP protocol compliance levels.

13.2.3 kebnf-to-tree-sitter: Spec-Driven Converter

The automated converter parses 640/640 KEBNF rules (100%) and achieves 93% automated conversion (38% direct mapping + 55% strip-and-convert), with 7% requiring manual intervention. The generated grammar has 335+ conflicts requiring iterative resolution — a process that is itself a research contribution documenting the gap between formal specifications and practical parsers.

13.2.4 Dual-Path Grammar Strategy Validated

The decision to maintain both empirical (tree-sitter-sysml) and formal (kebnf-to-tree-sitter) grammar paths proved valuable: the brute-force grammar provides immediate practical value while the spec-driven approach enables formal traceability and automated updates when OMG publishes spec revisions. Cross-validation between the two identifies spec interpretation errors in the hand-written grammar and practical parsing issues in the generated grammar.

13.3 Lessons Learned

13.3.1 Grammar Development Velocity

The tree-sitter-sysml grammar went from 10% to 100% training file coverage in approximately 20 hours of focused development. The key enablers were: (1) a simple test harness that provided immediate coverage feedback, (2) tree-sitter's built-in error recovery allowing partial progress, and (3) the empirical "fix the next failing file" methodology that avoided premature optimization. This suggests that for textual notations with good training corpora, grammar development can be faster than conventionally assumed.

13.3.2 Scope Expansion as Ecosystem Understanding

The project expanded from 3 planned repositories to 7, adding a grammar benchmark dashboard, PhD research planning, and detailed publication strategies. Rather than impeding the core GVSETS deliverable, this expansion provided a richer understanding of the SysML v2 ecosystem that strengthens the paper's future work articulation and positions the team for sustained contribution beyond the capstone.

13.3.3 Implementation-Informed Reviews

Conducting SRR and PDR after implementation began (rather than before) produced more meaningful reviews. Requirements were validated against working software rather than hypothetical designs. Architecture was confirmed by running code rather than paper analyses. This approach is consistent with agile SE practices for software-intensive systems, though it requires discipline to maintain formal traceability retroactively.

13.3.4 The SE/LSE Push-Pull Dynamic

The systems-engineer and lead-systems-engineer skill personas established a productive tension: the SE persona drafts comprehensively per INCOSE processes, while the LSE persona prunes pragmatically for project constraints. This dynamic produced artifacts that are both INCOSE-aligned and appropriately scoped — better than either perspective alone would achieve.

13.4 GVSETS Publication

The NDIA GVSETS 2026 paper ("Enabling AI-Augmented Model-Based Systems Engineering with the Model Context Protocol") is drafted with all sections present in `gvsets/paper/main.tex`. The evaluation section contains placeholder data pending execution of benchmark vignettes V1, V4, and V5 (Section C.1) against the Eve Mining Frigate model.

The paper argues that MCP provides a practical integration path for AI-augmented MBSE, with projected token reduction ratios and accuracy comparisons across three conditions: baseline (naive file concatenation), vanilla MCP (simple tool calls), and optimized MCP (Cache ID + Summary pattern with L0/L1/L2 tiered responses).

Critical path to submission: Execute benchmark vignettes to replace placeholder data, then polish prose from outline-quality to submission-quality. Draft submission deadline is March 23, 2026.

13.5 Future Work

13.5.1 Deferred In-Scope Items

Several items from the original scope (Section 1.6) were deferred based on risk mitigation and prioritization decisions during the capstone:

- **HTTP transport**: Deferred to Phase 2; stdio transport is sufficient for local development and GVSETS evaluation
- **SysML v2 API integration**: Deferred post-capstone per R1 mitigation; local tree-sitter parsing provides sufficient validation
- **Container deployment**: Blocked by R5 (macOS testing limitations); deferred to Phase 2

13.5.2 Post-Capstone (Informal Sprint Model)

After the capstone deadline (April 25, 2026), development continues informally without formal INCOSE review gates:

- **Token reduction Phase 2**: Implement remaining 6 strategies from the PRD (Cache ID + Summary, RTFM Documentation, Two Meta-Tool Architecture, KV-Cache Optimization, Overflow Detection, Vanilla Baseline)
- **tree-sitter-sysml 1.0**: Complete pre-release cleanup (Phase 3 CI verification, Phase 4 tree-sitter org submission)
- **Grammar benchmark dashboard**: Wire tree-sitter adapter and corpus into sysml-grammar-benchmark for community comparison
- **kebnf conflict resolution**: Iterative resolution of 335+ conflicts, documenting patterns for the grammar transposition paper

13.5.3 Research Instrument: sysml.rs

A planned Rust implementation of full SysML v2 semantic analysis (import resolution, type checking, constraint evaluation) will serve as a research instrument for PhD work. This complements tree-sitter-sysml's syntax-only capabilities with deep language understanding, enabling semantic-level comparison and context-aware MCP tool responses. See the PhD research portfolio for design documentation and timeline.

13.5.4 Publications

Three publications build on this capstone foundation (see Section B.1):

1. **GVSETS 2026** (Mar-Jun 2026): MCP architecture and proof of value
2. **Grammar Transposition** (Q3-Q4 2026): KEBNF-to-tree-sitter methodology
3. **INCOSE 2027** (Q3 2027): SE benchmark for AI evaluation

13.5.5 Ecosystem Positioning

This project fills a specific gap in the SysML v2 tooling ecosystem: no MCP server exists for MBSE among 7,364+ public MCP repositories. The open source, provider-agnostic design (MIT license, GitLab as reference implementation) complements commercial platforms like SysGit rather than competing with them. The tree-sitter-sysml grammar fills the gap left by Sensmetry's archived sysml-2ls language server, providing community-maintained SysML v2 parsing for any tree-sitter-compatible editor.

13.6 Acknowledgments

To be added prior to final submission.

13.7 References

See Section 15.1 for complete bibliography.

Chapter 14

Glossary

Term	Definition
ADD	Architecture Design Description
AST	Abstract Syntax Tree - semantic representation of parsed code
CDR	Critical Design Review
CI/CD	Continuous Integration / Continuous Deployment
Corpus Test	Tree-sitter test case in `corpus/` directory verifying grammar constructs
CST	Concrete Syntax Tree - parse tree preserving all source tokens (tree-sitter output)
External Scanner	C code in tree-sitter grammar for context-sensitive parsing
GKG	GitLab Knowledge Graph - code intelligence platform
INCOSE	International Council on Systems Engineering
KerML	Kernel Modeling Language (SysML v2 foundation)
MBSE	Model-Based Systems Engineering
MCP	Model Context Protocol
MR	Merge Request (GitLab term for Pull Request)
NDIA	National Defense Industrial Association
OMG	Object Management Group
PAT	Personal Access Token
PDR	Preliminary Design Review
RACI	Responsible, Accountable, Consulted, Informed
RTM	Requirements Traceability Matrix
SE	Systems Engineering
SEP	Systems Engineering Plan
SRR	System Requirements Review
SysML	Systems Modeling Language
SyRS	System Requirements Specification
Tree-sitter	Incremental parsing framework generating parsers from grammar definitions
V&V	Verification and Validation
VVP	Verification and Validation Plan
WASM	WebAssembly - portable binary format for web execution
WBS	Work Breakdown Structure

Chapter 15

References

15.1 Bibliography

[1] INCOSE, *INCOSE systems engineering handbook*, 5th ed. Wiley, 2023.

[2] Anthropic, "Model context protocol specification." 2024. Available: https://spec.modelcontextprotocol.io/

[3] Object Management Group, "OMG systems modeling language (SysML) v2.0 specification," 2025. Available: https://www.omg.org/spec/SysML/2.0/

[4] D. Horthy, "12-factor agents: Principles for building reliable LLM applications." https://github.com/humanlayer/12-factor-agents, 2025.

[5] R. Lopopolo, "Harness engineering: Leveraging Codex in an agent-first world." Accessed: Feb. 17, 2026. [Online]. Available: https://openai.com/index/harness-engineering/

[6] Anthropic, "Building effective agents." https://www.anthropic.com/research/building-effective-agents, 2024.

[7] Object Management Group, "OMG kernel modeling language (KerML) 1.0 specification," 2025. Available: https://www.omg.org/spec/KerML/1.0/

[8] Object Management Group, "OMG systems modeling API and services 1.0," 2025. Available: https://www.omg.org/spec/SystemsModelingAPI/1.0/

[9] S. Massey, "A discussion on accelerating hardware engineering through agile practices." 2025. Available: https://resources.sysgit.io/a-discussion-on-accelerating-hardware-engineering-through-agile-practices/

[10] Z. Li, S. Husung, and H. Wang, "LLM-Assisted Semantic Alignment and Integration in Collaborative Model-Based Systems Engineering Using SysML v2," in *2025 IEEE international symposium on systems engineering (ISSE)*, 2025, pp. 1–8. doi: 10.1109/ISSE65546.2025.11369983.

[11] M. A. Hendricks and A. Cicirello, "Text to Model via SysML: Automated Generation of Dynamical System Computational Models from Unstructured Natural Language Text via Enhanced System Modeling Language Diagrams," *arXiv preprint*, 2025, Available: https://arxiv.org/abs/2507.06803

[12] P. Darm, J. Xie, and A. Riccardi, "Inference-Time Intervention in Large Language Models for Reliable Requirement Verification," *arXiv preprint*, 2025, Available: https://arxiv.org/abs/2503.14130

[13] S. Otten *et al.*, "Generative AI in Systems Engineering: A Framework for Risk Assessment of LLMs," *arXiv preprint*, 2026, Available: https://arxiv.org/abs/2602.04358

[14] Y. Bouamra, B. Yun, A. Poisson, and F. Armetta, "SysTemp: A Multi-Agent System for Template-Based Generation of SysML v2." 2025. Available: https://arxiv.org/abs/2506.21608

[15] D. Jin, Z. Jin, L. Li, Z. Fang, J. Li, and X. Chen, "A System Model Generation Benchmark from Natural Language Requirements." 2025. Available: https://arxiv.org/abs/2508.03215

[16] P. Giannouris and S. Ananiadou, "NOMAD: Multi-Agent LLM System for UML Class Diagram Generation from Natural Language Requirements," *arXiv preprint*, 2025, Available: https://arxiv.org/abs/2511.22409

[17] A. Ferrari, S. Abualhaija, and C. Arora, "Model Generation with LLMs: From Requirements to UML Sequence Diagrams," *arXiv preprint*, 2024, Available: https://arxiv.org/abs/2404.06371

[18] E. Bader, D. Vereno, and C. Neureiter, "Facilitating User-Centric Model-Based Systems Engineering Using Generative AI," in *Proceedings of the 12th international conference on model-based software and systems engineering (MODELSWARD 2024)*, SCITEPRESS, 2024.

[19] S. Neema *et al.*, "On the Evaluation of Engineering Artificial General Intelligence," *arXiv preprint*, 2025, Available: https://arxiv.org/abs/2505.10653

[20] B. Jerry, L. Moreno, V. Francisco, and R. Hervas, "LLM-Driven Accessible Interface: A Model-Based Approach," *arXiv preprint*, 2026, Available: https://arxiv.org/abs/2601.06616

[21] S. O. Erikstad, "Multi-Agent LLMs and MBSE for Developing Design Optimization Models," 2024, Available: https://www.researchgate.net/publication/380882908

[22] D. Rouabhia and I. Hadjadj, "Behavioral Augmentation of UML Class Diagrams: LLMs for Method Generation," *arXiv preprint*, 2025, Available: https://arxiv.org/abs/2506.00788

[23] W. Mao *et al.*, "Data Dependency-Aware Code Generation from Enhanced UML Sequence Diagrams," *arXiv preprint*, 2025, Available: https://arxiv.org/abs/2508.03379

[24] A. Trendowicz *et al.*, "DeepQuali: LLMs for Assessing Quality of User Stories," *arXiv preprint*, 2026, Available: https://arxiv.org/abs/2602.08887

[25] E. S. Crabb and M. T. Jones, "Accelerating Model-Based Systems Engineering by Harnessing Generative AI," in *2024 19th annual system of systems engineering conference (SoSE)*, IEEE, 2024.

[26] S. Rodda, *Mcp-proxy: Aggregating MCP proxy with progressive tool disclosure.* (2025). Available: https://github.com/IAMSamuelRodda/mcp-proxy

[27] C. Luddy, *Xc-mcp: XCode CLI MCP server with progressive disclosure.* (2025). Available: https://github.com/conorluddy/xc-mcp

[28] Volcengine, *OpenViking: Context database for AI agents.* (2025). Available: https://github.com/volcengine/OpenViking

[29] Microsoft Research, *LLMLingua: Prompt compression for LLMs.* (2024). Available: https://github.com/microsoft/LLMLingua

[30] Y. Ji, "Context engineering for AI agents: Lessons from building manus." Accessed: Feb. 12, 2026. [Online]. Available: https://manus.im/blog/Context-Engineering-for-AI-Agents-Lessons-from-Building-Manus

[31] D. Horthy, "Advanced context engineering for coding agents." Accessed: Feb. 12, 2026. [Online]. Available: https://humanlayer.dev/blog/advanced-context-engineering

[32] Anthropic, "How we built our multi-agent research system." Accessed: Feb. 12, 2026. [Online]. Available: https://www.anthropic.com/engineering/multi-agent-research-system

Chapter 16

Requirements Traceability Matrix

> **Note**
>
> This RTM is the single source of truth for traceability. Individual chapters reference this appendix rather than duplicating trace information.

16.1 Stakeholder Needs to Stakeholder Requirements

Per [1, Sec. 3.2.3], traceability links stakeholder needs to derived requirements.

eholder Need	Stakeholder Requirement	Rationale
01 (Git API integration)	SR-001	Direct derivation
02 (Open source)	SR-002	Direct derivation
03 (CI/CD integration)	SR-003	Direct derivation
04 (GitLab demo)	SR-001, SR-003	Demonstration workflow
05 (INCOSE process)	SR-004	Direct derivation
06 (Technical reviews)	SR-005	Direct derivation
07 (Single binary)	SR-006	Direct derivation
08 (Documentation)	SR-007, SR-008	Direct derivation
09 (Container deployment)	SR-009	Direct derivation
10 (Natural language query)	SR-012	Direct derivation
11 (Self-hosted Git)	SR-010	Direct derivation
12 (Model validation)	SR-011	Direct derivation
13 (Open standards)	SR-002	Open source enables interop
14 (OMG spec conformance)	SR-013, SR-015	Direct derivation
15 (Error location)	SR-014	Direct derivation

16.2 Stakeholder Requirements to System Requirements

16.3 System Requirements to Architecture Elements

Stakeholder Requirement	System Requirement	Allocation
SR-001 (Git Provider API)	FR-REPO-001, FR-REPO-002, FR-REPO-003, FR-REPO-004, FR-REPO-005	Repo Client
SR-002 (Open source license)	NFR-DOC-003	Documentation
SR-003 (CI/CD examples)	FR-MCP-003	MCP Server
SR-006 (Single binary)	NFR-DEP-001, NFR-DEP-003, NFR-DEP-004	Build/Deploy
SR-007 (README)	NFR-DOC-001	Documentation
SR-008 (Examples)	NFR-DOC-002, FR-SYS-005	Documentation
SR-009 (Container)	NFR-DEP-002	Build/Deploy
SR-010 (Self-hosted Git provider)	FR-REPO-004	Repo Client
SR-011 (Model validation)	FR-SYS-003	SysML API Client
SR-012 (SysML parsing)	FR-SYS-001, FR-SYS-002, FR-SYS-004	SysML Parser
SR-013 (OMG grammar conformance)	FR-SYS-001, FR-SYS-006	SysML Parser
SR-014 (Error location)	FR-SYS-007	SysML Parser
SR-015 (Documented subset)	FR-SYS-008	Documentation

System Requirement	Architecture Element	Component
FR-MCP-001 through FR-MCP-006	MCP Server	open-mcp-sysml/crates/mcp-server
FR-REPO-001 through FR-REPO-007	Repo Client	open-mcp-sysml/crates/repo-client
FR-SYS-001, FR-SYS-002, FR-SYS-006, FR-SYS-007	SysML Parser	tree-sitter-sysml (grammar + Rust bindings)
FR-SYS-003, FR-SYS-004	SysML API Client	open-mcp-sysml/crates/mcp-server
FR-SYS-008	Documentation	tree-sitter-sysml/README.md
NFR-DEP-001	Build Configuration	Cargo.toml
NFR-DEP-002	Container Image	Containerfile

16.4 System Requirements to Test Cases

Requirement	Test Case	Verification Method
FR-MCP-001	TC-MCP-001	Test
FR-MCP-002	TC-MCP-002, TC-MCP-003	Test
FR-MCP-004	TC-MCP-004	Test
FR-MCP-005	TC-MCP-005	Test
FR-REPO-001	TC-REPO-001, TC-REPO-002	Test
FR-REPO-002	TC-REPO-003	Test
FR-REPO-004	TC-REPO-004	Test
FR-REPO-005	TC-REPO-005	Test
FR-SYS-001	TC-SYS-001 through TC-SYS-005, corpus tests	Test
FR-SYS-006	Corpus tests (100%), training file coverage (%)	Test, Analysis
FR-SYS-007	TC-SYS-005 (error recovery)	Test
FR-SYS-008	tree-sitter-sysml README review	Inspection
NFR-DEP-001	CI build job	Test, Analysis

WBS	Phase	Requirements Addressed

Appendix A

Appendix: GitLab Knowledge Graph Plugin Architecture Proposal

A.1 Executive Summary

This appendix documents exploratory research into extending GitLab Knowledge Graph (GKG) for domain-specific language support, using SysML v2 as a case study. The goal is to identify architectural patterns that could enable GKG to understand repository content beyond traditional programming languages.

> **i** Status: Aspirational Future Work
>
> This proposal has not been coordinated with GKG maintainers and represents aspirational future work. The immediate project goal is to build a standalone SysML v2 MCP server; lessons learned from that implementation would inform any future GKG contribution.
> Contributing to GKG would require:
> 1. Demonstrating value through the standalone implementation
> 2. Coordinating with GKG maintainers on priorities
> 3. Aligning with GKG's roadmap and architectural decisions

A.2 Background

A.2.1 What is GitLab Knowledge Graph?

GitLab Knowledge Graph (GKG) is an open-source project that creates structured, queryable representations of code repositories to power AI features and enhance developer productivity. Key characteristics:

Aspect	Description
Language	Rust
Parser	`gitlab-code-parser` using tree-sitter + ast-grep
Storage	KuzuDB (embedded graph database) + Parquet files
Interface	CLI, HTTP server, MCP protocol
Scope	Code structure: definitions, references, imports, call graphs

Repository: gitlab.com/gitlab-org/rust/knowledge-graph

A.2.2 Current Language Support

GKG supports programming languages via `gitlab-code-parser`:

Language	Definitions	Intra-file Refs	Cross-file Refs
Ruby			
Python			Partial
TypeScript/JavaScript			Partial
Kotlin			
Java			
Rust			Partial

A.2.3 How Languages Are Added

Languages are added to GKG via compile-time extension of `gitlab-code-parser`:

1. Add language to `SupportedLanguage` enum in `parser.rs`
2. Create YAML rule files using ast-grep patterns
3. Implement post-processing logic for structured extraction
4. Update `RuleManager` to load rules for the new language

This model assumes:

- A tree-sitter grammar exists for the language
- The language follows programming language patterns (definitions, references, imports)
- Language support is baked into the binary at compile time

A.3 The Domain-Specific Language Challenge

A.3.1 Programming Languages vs. Modeling Languages

GKG's current architecture is optimized for programming languages. Domain-specific languages (DSLs) like SysML v2 present different characteristics:

Aspect	Programming Languages	SysML v2
Primary artifacts	Functions, classes, modules	Parts, requirements, actions
Relationships	Calls, imports, inheritance	Specialization, allocation, composition
Semantics	Execution semantics	Model semantics (KerML FOL)
Files	`.py`, `.rs`, `.ts`	`.sysml`
Tree-sitter grammar	Widely available	Does not exist

A.3.2 Why SysML v2 Doesn't Fit the Current Model

1. **No tree-sitter grammar**: GKG relies on tree-sitter for AST generation. No tree-sitter grammar exists for SysML v2.
2. **Different relationship types**: GKG's graph schema centers on code relationships (calls, imports). SysML v2 has different relationship types (allocation, satisfaction, derivation).
3. **Compile-time language list**: Languages must be added at compile time. Organizations can't add domain-specific support without forking GKG.

A.3.3 Other Affected Domains

SysML v2 is not unique. Other domain-specific content faces similar challenges:

Domain	File Types	Value to Repository Understanding
SysML v2	`.sysml`	Systems architecture, requirements traceability
Terraform	`.tf`	Infrastructure dependencies, resource relationships
OpenAPI	`.yaml`, `.json`	API structure, endpoint relationships
Protobuf	`.proto`	Service definitions, message relationships
GraphQL	`.graphql`	Schema structure, type relationships
Kubernetes	`.yaml`	Deployment topology, resource dependencies

A.4 GKG's Existing Extensibility Work

GKG maintainers have already considered extensibility. Key initiatives:

A.4.1 Graph Extractor Language (GEL)

Issue #227 introduced GEL, a DSL for custom extraction rules:

> "Custom Extraction with Graph Extractor Language (GEL): We will introduce and document GEL, a custom DSL that allows developers to define their own rules for extracting framework-specific nodes and relationships from the AST."

Example use case: Extracting Next.js API routes from TypeScript files.

Limitation: GEL operates on top of already-parsed ASTs. It cannot handle languages without tree-sitter grammars.

A.4.2 Contributions Pipeline

Issue #139 outlines a vision for extensibility:

> "Engineer an Extensible Framework: The core of the strategy is to provide clear, powerful extension points that allow developers to add significant value without needing to modify the indexer's core logic."

Key pillars identified:

1. Adding new languages via `gitlab-code-parser` patterns
2. Custom extraction via GEL
3. Comprehensive documentation

A.4.3 SCIP Integration

Issue #270 explores SCIP (Source Code Intelligence Protocol) for broader language support:

> "We are currently developing custom code parsers in-house, which requires significant maintenance effort and limits our language coverage. We should investigate integrating with SCIP…"

This indicates interest in reducing parser maintenance burden.

A.5 Proposed Plugin Architecture

To support domain-specific languages like SysML v2, GKG could introduce a plugin architecture for **context providers**.

A.5.1 Design Goals

1. **Runtime extensibility**: Add language support without recompiling GKG
2. **Isolation**: Plugins cannot crash the core indexer
3. **Schema flexibility**: Plugins can introduce custom node/relationship types
4. **Discoverability**: Users can find and install plugins easily

A.5.2 Proposed Interface

```rust
/// A plugin that provides domain-specific context for a file type
pub trait ContextProvider: Send + Sync {
    /// Plugin metadata
    fn metadata(&self) -> PluginMetadata;

    /// File extensions this plugin handles
    fn supported_extensions(&self) -> &[&str];

    /// Parse a file and extract definitions
    fn extract_definitions(
        &self,
        content: &str,
        path: &str
    ) -> Result<Vec<Definition>, PluginError>;

    /// Extract relationships between definitions
    fn extract_relationships(
        &self,
        content: &str,
        path: &str,
        definitions: &[Definition],
    ) -> Result<Vec<Relationship>, PluginError>;
}

pub struct PluginMetadata {
    pub name: String,
    pub version: String,
    pub description: String,
    pub author: String,
}

pub struct Definition {
    pub id: String,
    pub name: String,
    pub definition_type: String,  // e.g., "PartDefinition", "Requirement"
    pub fqn: String,
    pub location: Location,
    pub properties: HashMap<String, Value>,
}

pub struct Relationship {
    pub source_id: String,
    pub target_id: String,
    pub relationship_type: String,  // e.g., "specializes", "allocates"
    pub properties: HashMap<String, Value>,
}
```

A.5.3 Plugin Discovery Mechanisms

Several approaches could enable runtime plugin loading:

Mechanism	Pros	Cons
WASM	Sandboxed, portable	Performance overhead, limited I/O

Mechanism	Pros	Cons
Dynamic libraries	Native performance	Platform-specific, security concerns
Subprocess	Language-agnostic, isolated	IPC overhead, process management
gRPC service	Network-capable, language-agnostic	Deployment complexity

Recommendation: Start with subprocess-based plugins (simple JSON protocol over stdin/stdout), evolve to WASM for sandboxing.

A.5.4 Schema Evolution

Plugins introducing custom node/relationship types need schema management:

```yaml
# sysml-plugin/schema.yaml
nodes:
  - name: PartDefinition
    extends: Definition
    properties:
      - name: isAbstract
        type: boolean
  - name: RequirementDefinition
    extends: Definition
    properties:
      - name: text
        type: string

relationships:
  - name: specializes
    from: [PartDefinition, RequirementDefinition]
    to: [PartDefinition, RequirementDefinition]
  - name: satisfies
    from: [PartDefinition]
    to: [RequirementDefinition]
  - name: allocates
    from: [PartDefinition]
    to: [PartDefinition]
```

GKG's schema manager would:

1. Load plugin schemas at startup
2. Create corresponding KuzuDB tables
3. Validate plugin output against declared schema

A.6 SysML v2 as Reference Plugin

A.6.1 What SysML Context Would Provide

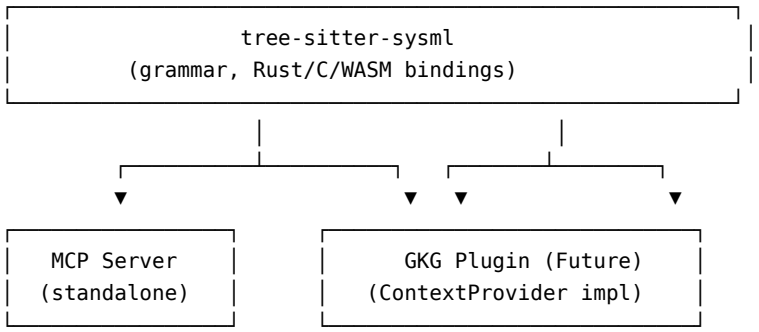
A SysML v2 plugin would enable GKG to understand:

Graph Node	Description	Query Example
`PartDefinition`	System/component definitions	"What parts does Vehicle contain?"
`RequirementDefinition`	Requirements	"What requirements trace to Engine?"
`ActionDefinition`	Behaviors/functions	"What actions does StartEngine perform?"

Graph Node	Description	Query Example
`AllocationRelationship`	Function-to-structure	"What functions are allocated to ECU?"
`SatisfactionRelationship`	Requirement satisfaction	"What requirements are satisfied by tests?"

A.6.2 Integration with Standalone MCP Server

The standalone SysML v2 MCP server (this project) would share parsing logic with a potential GKG plugin via the tree-sitter grammar:



This allows:

1. Proving grammar correctness via MCP server
2. Reusing grammar in GKG plugin without duplication
3. Contributing grammar to GitLab vendor/grammars/ for syntax highlighting
4. Independent evolution of MCP tools vs. GKG integration

A.6.3 Example Queries Enabled

With SysML v2 support, GKG could answer:

```
// Find all parts that satisfy safety requirements
MATCH (p:PartDefinition)-[:satisfies]->(r:RequirementDefinition)
WHERE r.name CONTAINS 'Safety'
RETURN p.name, r.name

// Trace function allocation to physical structure
MATCH (a:ActionDefinition)-[:allocatedTo]->(p:PartDefinition)
RETURN a.name AS function, p.name AS component

// Find unallocated requirements
MATCH (r:RequirementDefinition)
WHERE NOT (r)<-[:satisfies]-()
RETURN r.name AS untraced_requirement
```

A.7 Implementation Considerations

A.7.1 WASM vs Native vs Subprocess

Approach	Performance	Security	Portability	Complexity
WASM	Medium	High (sandboxed)	High	Medium
Native (.so/.dylib)	High	Low	Low	High
Subprocess	Low-Medium	Medium	High	Low

Recommendation for MVP: Subprocess with JSON-over-stdio. Simple to implement, language-agnostic, and isolates plugin crashes.

A.7.2 Performance Implications

Plugin-based parsing adds overhead:

Operation	In-process	Subprocess
Parse 1KB file	~1ms	~10-50ms
Parse 1MB file	~100ms	~200-500ms
Batch 1000 files	~1s	~10-30s

Mitigations:

1. Batch file processing to amortize IPC cost
2. Plugin-side caching of parsed ASTs
3. Incremental re-indexing (only changed files)

A.7.3 Security Considerations

Plugins execute arbitrary code. Security measures:

1. **Sandboxing**: Prefer WASM for production plugins
2. **Capabilities**: Plugins declare required permissions (file read, network, etc.)
3. **Signed plugins**: Require cryptographic signatures for trusted plugins
4. **Review process**: Plugin registry with review before listing

A.8 Contribution Path

A.8.1 Alignment with GKG Roadmap

This proposal aligns with GKG maintainers' expressed interests:

GKG Initiative	Alignment
Issue #139 (Contributions Pipeline)	Plugin architecture enables external contributions
Issue #227 (GEL)	Plugins extend beyond GEL's AST-based scope
Issue #270 (SCIP)	Plugin approach is complementary to SCIP

A.8.2 Phased Approach

1. **Phase A: Standalone MCP Server** (this project)
   - Prove SysML v2 parser correctness
   - Establish parsing patterns and AST design
   - Demonstrate value to MBSE practitioners
2. **Phase B: Tree-sitter Grammar** (future)
   - Contribute tree-sitter grammar for SysML v2
   - Enables native GKG support without plugins
   - Requires significant specification analysis
3. **Phase C: Plugin Architecture Proposal** (future)
   - Present findings to GKG maintainers
   - Collaborate on plugin interface design
   - Implement SysML v2 as reference plugin

A.8.3 Prerequisites

Before proposing to GKG team:

- ☐ Working standalone SysML v2 MCP server
- ☐ Documented parser with coverage matrix
- ☐ Usage data demonstrating value
- ☐ Draft plugin interface specification
- ☐ Performance benchmarks

A.9 Conclusion

Extending GitLab Knowledge Graph for domain-specific languages like SysML v2 is technically feasible through a plugin architecture. The standalone SysML v2 MCP server serves as a proving ground for:

1. Parser design and correctness
2. Graph schema for MBSE concepts
3. Value proposition for AI-assisted systems engineering

Lessons learned will inform future contributions to GKG, contingent on maintainer interest and project priorities.

A.10 References

- GitLab Knowledge Graph: https://gitlab.com/gitlab-org/rust/knowledge-graph
- gitlab-code-parser: https://gitlab.com/gitlab-org/rust/gitlab-code-parser
- GKG Issue #139 (Contributions Pipeline): https://gitlab.com/gitlab-org/rust/knowledge-graph/-/issues/139
- GKG Issue #227 (GEL POC): https://gitlab.com/gitlab-org/rust/knowledge-graph/-/issues/227
- GKG Issue #270 (SCIP Integration): https://gitlab.com/gitlab-org/rust/knowledge-graph/-/issues/270
- ASIMOV Platform (reference): https://asimov.blog/introducing-asimov/
- sysml.rs (reference): https://github.com/artob/sysml.rs

Appendix B

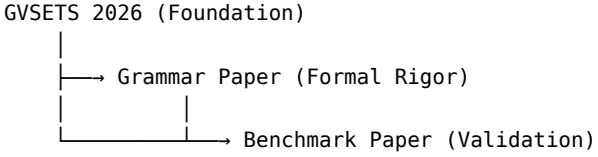
Publication Strategy

B.1 Overview

The expanded project scope supports three distinct publications, each building on the prior:

Paper	Repository	Target Venue	Timing	Status
GVSETS 2026	gvsets/	NDIA GVSETS	Draft Mar 23, Final Jun 5	Drafted, evaluation pending
Grammar Transposition	kebnf-to-tree-sitter	MODELS/SLE 2026 or SE Journal	Q3-Q4 2026	Outline complete
SE Benchmark for AI	sysml-grammar-benchmark	INCOSE IS 2027	Q3 2027	Notional

B.1.1 Publication Dependencies



B.1.2 Authors (All Papers)

- Andrew Dunn (GitLab Public Sector)
- Greg Pappas (DoD, Army, AFC-DEVCOM)
- Dr. Stephen Rapp (Wayne State University, ISE)

B.2 GVSETS 2026: AI-Augmented MBSE

Working Title: "Enabling AI-Augmented Model-Based Systems Engineering with the Model Context Protocol"

Attribute	Value
Track	Digital Engineering / AI
Format	8-page technical paper + presentation
Draft Due	March 23, 2026
Notification	May 1, 2026

Attribute	Value
Final Due	June 5, 2026
Presentations Due	July 23, 2026
Presentation	August 11, 2026 (Novi, MI)

B.2.1 Key Thesis

MCP provides a standardized interface enabling AI assistants to interact with SysML v2 models stored in Git repositories, offering token efficiency via selective retrieval (estimated 80-97% reduction with L0/L1/L2 detail levels, pending benchmark validation), structured responses eliminating parsing ambiguity, and authoritative answers from tools connected to real repositories.

B.2.2 Current Status

The paper is drafted in `gvsets/paper/main.tex` with all sections present. The evaluation section (Section 5) contains placeholder data pending benchmark vignette execution (V1, V4, V5 from Section C.1). Eight TODO markers flag unvalidated quantitative claims that must be resolved before submission.

B.2.3 3-Condition Experiment Design

Condition	Description
Baseline	All files concatenated into prompt (naive approach)
Vanilla MCP	Simple tool calls without optimization
Optimized MCP	Cache ID + Summary pattern, L0/L1/L2 tiered responses

Critical path: Execute benchmark vignettes V1/V4/V5 against the Eve Mining Frigate model to replace placeholder data with measured results.

B.2.4 Relationship to Capstone

This paper establishes the foundation — demonstrating practical AI-MBSE integration and proof of value. The systems engineering artifacts in this capstone (SEP, SyRS, ADD, VVP, RTM) provide the methodological rigor backing the paper's claims. The expanded ecosystem understanding from scope exploration (7 projects) enables articulation of meaningful future research directions.

B.3 Grammar Transposition Paper

Working Title: "Automated Grammar Transposition: Converting OMG KEBNF Specifications to Tree-sitter Parsers"

Target Venue: MODELS/SLE 2026 or Systems Engineering Journal

Attribute	Value
Format	10-12 page technical paper
Target	Q3-Q4 2026 submission
Repository	kebnf-to-tree-sitter

B.3.1 Key Thesis

Formal specification grammars (KEBNF) can be systematically converted to practical parser generators (tree-sitter) with ~93% automation and documented semantic mappings. This enables reproducible grammar generation when specifications update, formal traceability from parser rules to specification sources, and a reusable methodology for any OMG KEBNF-based specification (OCL, Alf, future textual notations).

B.3.2 Unique Contribution

This is the first documented methodology for KEBNF → tree-sitter conversion. No existing literature addresses KEBNF specifically, despite OMG's use of KEBNF across multiple standards and tree-sitter's rapid adoption across major editors and platforms.

B.3.3 Paper Structure

1. **Introduction** (~1 page): MBSE adoption driving need for SysML v2 tooling; gap between OMG specifications and practical parsers; contribution statement
2. **Background** (~2 pages): OMG grammar specifications, KEBNF syntax elements (type annotations, property assignments, cross-references, semantic actions), tree-sitter architecture
3. **Methodology** (~3 pages): KEBNF pattern taxonomy, conversion algorithm (parse → classify → transform → record mapping → emit), semantic mapping document design
4. **Implementation** (~2 pages): Tool architecture (Chumsky parser → Mapper → tree-sitter emitter), technology choices, conflict detection approach
5. **Case Study: SysML v2** (~2 pages): Input corpus (640 rules across KerML + SysML KEBNF files), automation results by category, comparison with hand-written tree-sitter-sysml grammar
6. **Discussion** (~1 page): Applicability beyond SysML (OCL, Alf), limitations, tree-sitter enhancement opportunities
7. **Conclusions** (~0.5 page)

B.3.4 Current Status

The kebnf-to-tree-sitter tool is functional: parser complete (640/640 rules), emitter produces tree-sitter grammar.js output. Generated grammar has 335+ conflicts requiring iterative resolution. The iterative resolution process (fixing conflicts one at a time with documented rationale) is itself a contribution for the paper.

B.3.5 Automation Results

Category	% Rules	Handling
Direct conversion	38%	Basic syntax maps directly
Strip & convert	55%	Remove annotations, keep structure
Best-effort	6%	Approximate semantic actions
Manual review	<1%	Complex disambiguation

B.3.6 Dual-Path Cross-Validation

Comparing the generated grammar against the hand-written tree-sitter-sysml identifies spec interpretation errors in the hand-written grammar and practical parsing issues in the generated grammar. This cross-validation is a novel contribution.

B.4 INCOSE 2027: SE Benchmark for AI

Working Title: "Toward a Systems Engineering Benchmark for Large Language Models"

Target Venue: INCOSE International Symposium 2027 or Systems Engineering Journal

Attribute	Value
Format	10-12 page technical paper
Target	Q2-Q3 2027 submission
Builds On	GVSETS 2026 (MCP server), Grammar Paper (formal methodology)

B.4.1 Key Thesis

The systems engineering community needs standardized benchmarks to evaluate AI/LLM capabilities on SE tasks, analogous to SWE-bench for software engineering. No standardized benchmark exists for requirements engineering, architecture definition, or V&V — core SE activities. Without benchmarks, progress in AI4SE cannot be measured objectively.

B.4.2 Gap Analysis

Existing Benchmark	Domain	SE Coverage
SWE-bench	Software bug fixing	None
HumanEval	Code generation	None
Humanity's Last Exam	Expert knowledge	Minimal
No existing benchmark	Systems engineering	-

B.4.3 Proposed Framework

SE task taxonomy aligned with INCOSE processes: requirements elicitation, requirements quality assessment, model completion, test generation from requirements, and requirements-to-design traceability. Each task has deterministic ground truth, measurable evaluation metrics, and supports comparison between baseline AI and MCP-enabled AI conditions.

B.4.4 Relationship to Benchmark Vignettes

The benchmark vignettes defined in Section C.1 (V1-V8) serve as pilot tasks for this paper. The GVSETS paper uses V1, V4, V5 for proof of value; this paper expands to the full set and adds formal evaluation methodology.

B.4.5 Timeline

Phase	Target	Activities
Foundation	Current (capstone)	Literature review, MCP implementation, vignette definitions
Task Design	Q3 2026	Define 50-100 tasks, evaluation protocols
Pilot Study	Q4 2026	Run benchmark, collect data
SME Validation	Q1 2027	Expert review of tasks and results
Paper Draft	Q2 2027	Write and internal review
Submission	Q3 2027	Target INCOSE IS 2027

Appendix C

Benchmark Vignettes: MCP Evaluation Tasks

C.1 Overview

This appendix defines concrete, reproducible benchmark tasks for evaluating MCP-based SysML v2 tooling. Each vignette measures either **token efficiency** or **retrieval accuracy**, using publicly available models.

C.1.1 Design Principles

1. **Reproducibility**: All models are publicly available on GitHub
2. **Measurability**: Each task has deterministic ground truth
3. **Discriminative**: "Dump everything" baseline fails; targeted retrieval succeeds
4. **Executable**: Implementable within 3-week timeline (by March 2026)

C.1.2 Source Models

Repository	Model	Files	Characters	Est. Tokens
GfSE/SysML-v2-Models	Eve Online Mining Frigate	18	~98K	~25K
GfSE/SysML-v2-Models	VehicleModel.sysml	1	~27K	~7K
Systems-Modeling/SysML-v2-Pilot	Training Examples	~40+	~200K	~50K

Token estimates use 4 chars/token heuristic for code.

C.2 Vignette 1: Requirement Constraint Extraction

Task: Extract all numeric constraint values from requirements in a multi-file model.

C.2.1 Specification

Attribute	Value
Question	"List all requirements with numeric thresholds and their constraint values"
Input Model	Eve Online Mining Frigate (18 files, ~98K chars)
Files Required	`MiningFrigateRequirementsDef.sysml` (4.8K chars)
Measurement	Token efficiency + Accuracy

C.2.2 Ground Truth

```
MFRQ01: miningRateLS >= 50.0 (m³/min)
MFRQ02: cargoCapacity >= 5000.0 (m³)
MFRQ03: shieldStrengthHS >= 200.0 (DPS), shieldStrengthLS >= 400.0 (DPS)
MFRQ04: droneCapacity >= 5
STRQ05: threatDetectionRange >= 20.0 (AU)
MFRQ06: warpSpeed >= 5.0 (AU/s), alignTime <= 3.0 (s)
MFRQ07: lockedTargets <= 3
MFRQ08: dockingTime <= 60.0 (s)
MFRQ09: fleetSize <= 10
MFRQ10: compressionFactor == 10.0
```

C.2.3 Evaluation

Metric	Calculation
Accuracy	F1 score: correct constraints / (correct + missed + hallucinated)
Token Efficiency	Baseline tokens / MCP tokens

C.2.4 Baseline vs MCP

Condition	Input Tokens	Expected Accuracy
Baseline (all files)	~25,000	60-80% (distractor content)
MCP (targeted query)	~1,500	95%+ (requirement file only)

C.3 Vignette 2: Requirements-to-Verification Traceability

Task: For a specific requirement, identify which verification cases test it.

C.3.1 Specification

Attribute	Value
Question	"Which verification case tests requirement MFRQ03 (SurvivabilityRequirement)?"
Input Model	Eve Online Mining Frigate
Files Required	`MiningFrigateVerificationCases.sysml`, `MiningFrigateRequirements.sysml`
Measurement	Accuracy (binary: correct/incorrect)

C.3.2 Ground Truth

```
Requirement: MFRQ03 (SurvivabilityRequirement)
Verification Case: SurvivabilityTest
Verification Instance: survivabilityTest
Method: test
Actions: simulateHighSecAttack, simulateLowSecAttack, evaluateData
Verdict Binding: survivabilityRequirementLowSec
```

C.3.3 Evaluation

Metric	Calculation
Accuracy	1 if correct case identified with method, 0 otherwise
Partial Credit	0.5 if case name correct but method/actions wrong

C.3.4 Baseline vs MCP

Condition	Input Tokens	Challenge
Baseline	~25,000	Must find `verify survivabilityRequirementLowSec` across files
MCP	~3,500	Query verification cases, follow `verify` reference

C.4 Vignette 3: Cross-File Interface Compatibility

Task: Determine if two ports are type-compatible for connection.

C.4.1 Specification

Attribute	Value
Question	"Can MiningFrigate.controlPort connect to Domain.PodPort?"
Input Model	Eve Online Mining Frigate
Files Required	`MiningFrigate.sysml`, `Domain.sysml`
Measurement	Accuracy + Reasoning correctness

C.4.2 Ground Truth

```
MiningFrigate.controlPort : ~Domain::PodPort (conjugate)
Domain.PodPort : (defined in Domain.sysml)

Compatibility: YES
Reason: controlPort is conjugate (~) of PodPort, enabling bidirectional flow
```

C.4.3 Evaluation

Metric	Calculation
Binary Accuracy	1 if yes/no correct

Metric	Calculation
Reasoning Score	1 if conjugate relationship explained, 0 otherwise

C.4.4 Baseline vs MCP

Condition	Input Tokens	Challenge
Baseline	~25,000	Must understand ~ conjugate syntax across files
MCP	~4,000	Query port definitions, resolve type references

C.5 Vignette 4: Element Inventory Count

Task: Count specific element types across a model.

C.5.1 Specification

Attribute	Value
Question	"How many `part def` definitions exist in the VehicleModel?"
Input Model	VehicleModel.sysml (598 lines, 27K chars)
Measurement	Exact match accuracy

C.5.2 Ground Truth

```
part def count: 32

Including: Vehicle, Engine, Cylinder, Transmission, Driveshaft,
AxleAssembly, Axle, FrontAxle, HalfAxle, Differential, Wheel,
Software, VehicleSoftware, VehicleController, FuelTank, Road,
VehicleRoadContext, SpatialTemporalReference, Engine4Cyl,
Engine6Cyl, TransmissionChoices, TransmissionAutomatic,
TransmissionManual, Sunroof, ...
```

C.5.3 Evaluation

Metric	Calculation
Exact Match	1 if count == 32, 0 otherwise
Tolerance	±2 for partial credit (0.5)

C.5.4 Baseline vs MCP

Condition	Input Tokens	Challenge
Baseline	~7,000	Must parse and count all `part def`
MCP	~500	`count_elements(type="part def")` tool call

C.6 Vignette 5: Constraint Satisfaction Check

Task: Given attribute values, determine if a constraint is satisfied.

C.6.1 Specification

Attribute	Value
Question	"If miningRate = 45.0, does the Mining Frigate satisfy MFRQ01?"
Input Model	Eve Online Mining Frigate
Files Required	`MiningFrigateRequirementsDef.sysml`
Measurement	Accuracy + Reasoning

C.6.2 Ground Truth

```
Requirement MFRQ01 (OreExtractionEfficiencyRequirement):
  require constraint { miningRateLS >= 50.0 }

Given: miningRate = 45.0
Evaluation: 45.0 >= 50.0 → FALSE

Answer: NO, requirement is NOT satisfied (45.0 < 50.0 required)
```

C.6.3 Evaluation

Metric	Calculation
Binary Accuracy	1 if correct yes/no
Reasoning	1 if constraint value cited correctly

C.7 Vignette 6: State Machine Transition Query

Task: Identify what triggers a specific state transition.

C.7.1 Specification

Attribute	Value
Question	"What command triggers the Mining Frigate to transition from InGrid to OnWarp state?"
Input Model	Eve Online Mining Frigate
Files Required	`MiningFrigate.sysml`
Measurement	Accuracy

C.7.2 Ground Truth

```
State Machine: miningFrigatesStates
Transition: inGrid_to_onWarp
  first InGrid
  accept warpCommand : Domain::ShipCommand via miningFrigates.controlPort
  do action executeWarpDrive : ExecuteWarpDrive
```

```
  then OnWarp

Trigger: warpCommand (Domain::ShipCommand) via controlPort
Action: executeWarpDrive
```

C.7.3 Evaluation

Metric	Calculation
Trigger Accuracy	1 if warpCommand identified
Port Accuracy	1 if controlPort identified
Action Accuracy	1 if executeWarpDrive identified
Composite	Average of three

C.8 Vignette 7: Stakeholder Concern Traceability

Task: Find which requirements address a specific stakeholder concern.

C.8.1 Specification

Attribute	Value
Question	"Which requirements frame the SecurityConcern?"
Input Model	Eve Online Mining Frigate
Files Required	`MiningFrigateRequirementsDef.sysml`, `Concerns.sysml`
Measurement	Precision/Recall

C.8.2 Ground Truth

```
Requirements framing SecurityConcern:
1. MFRQ03 (SurvivabilityRequirement) - frame concern SecurityConcern
2. STRQ05 (ThreatDetectionRequirement) - frame concern SecurityConcern
```

C.8.3 Evaluation

Metric	Calculation
Precision	Correct / Total Returned
Recall	Correct / Ground Truth (2)
F1	$2 \times (P \times R) / (P + R)$

C.9 Vignette 8: Import Dependency Resolution

Task: Trace what a specific element depends on through imports.

C.9.1 Specification

Attribute	Value
Question	"What packages must be imported to use MiningFrigate::MiningFrigate?"
Input Model	Eve Online Mining Frigate
Files Required	Multiple (follow import chain)
Measurement	Completeness

C.9.2 Ground Truth

```
Direct imports in MiningFrigate.sysml:
- ScalarValues::*
- ISQ::*
- SI::*
- ParametersOfInterestMetadata::*
- OperationalUseCaseActions::*
- Domain::*

Transitive: Domain imports additional packages...
```

C.9.3 Evaluation

Metric	Calculation
Direct Import Recall	Correct direct imports / 6
Token Efficiency	Baseline / MCP tokens to get answer

C.10 Summary: Benchmark Matrix

ID	Task Type	Primary Metric	Token Ratio Target	Difficulty
V1	Extraction	F1 + Efficiency	15:1	Medium
V2	Traceability	Accuracy	7:1	Medium
V3	Compatibility	Accuracy + Reason	6:1	Hard
V4	Inventory	Exact Match	14:1	Easy
V5	Constraint	Accuracy + Reason	10:1	Easy
V6	Behavior	Composite	8:1	Medium
V7	Traceability	F1	10:1	Medium
V8	Resolution	Completeness	5:1	Hard

C.11 Execution Protocol

C.11.1 Setup

1. Clone model repositories:

```
git clone https://github.com/GfSE/SysML-v2-Models.git
git clone https://github.com/Systems-Modeling/SysML-v2-Pilot-Implementation.git
```

2. Configure MCP server with model paths
3. Prepare baseline prompts (concatenated file contents)

C.11.2 Per-Vignette Execution

1. **Baseline Run**:
   - Concatenate all relevant files into prompt
   - Record input token count
   - Submit question, record response
   - Evaluate against ground truth
2. **MCP Run**:
   - Submit question with MCP tools available
   - Record tool calls and their token costs
   - Record total token usage
   - Evaluate against ground truth
3. **Metrics Collection**:
   - Accuracy score per evaluation criteria
   - Input tokens (baseline)
   - Input tokens (MCP total)
   - Output tokens (both conditions)
   - Latency (optional)

C.11.3 Statistical Validity

- Run each vignette 3 times per condition
- Report mean and standard deviation
- Use same model temperature (0.0 for reproducibility)
- Document model version and date

C.12 Future Extensions

C.12.1 Additional Vignettes (Post-March)

- **V9**: Multi-model consistency checking
- **V10**: Requirements completeness analysis
- **V11**: Allocation verification (requirements → design)
- **V12**: Change impact analysis

C.12.2 Expanded Model Set

- OMG SysML v2 training examples (~50K tokens)
- Larger industrial models (if available under open license)
- Synthetic models with known properties