SysML v2 MCP Server

Systems Engineering Capstone Project

Andrew Dunn Greg Pappas Dr. Stephen Rapp

2026-01-25

Table of contents

Chapter 1

SysML v2 MCP Server

Download as 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**: Position GitLab as infrastructure for AI-augmented Model-Based Systems Engineering (MBSE) workflows
2. **Academic Capstone**: Demonstrate INCOSE systems engineering principles [1] for a Wayne State University masters engineering capstone project

1.1.1 Key Deliverables

- Working MCP server with GitLab integration and SysML v2 API support
- NDIA GVSETS paper (draft March 5, final April)
- Capstone SE documentation (SEP, SyRS, ADD, VVP)

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

1.2 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 GitLab.

```
WITHOUT MCP SERVER:


     Engineer       copy/paste        AI Assistant
                    copy/paste       (Claude, etc.)





      GitLab        (no connection)      Generic SysML
      .sysml                             knowledge only


  Problems: AI sees snippets, not full project. Cannot validate.
            Cannot commit. Context lost between conversations.


WITH MCP SERVER:

                 MCP
     Engineer       Protocol       AI Assistant
                               (Claude, etc.)


                                               MCP


                                        SysML v2 MCP
                                           Server
```

```
        GitLab        SysML v2        Local
        .sysml        API Server      Parser


  Benefits: AI reads full project. Validates models. Commits changes.
            Structured understanding. Persists across conversations.
```

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 GitLab 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 GitLab for model persistence and CI/CD
3. Connect to SysML v2 API Services for validation
4. Demonstrate AI-augmented MBSE workflows
5. Publish findings at NDIA GVSETS

1.5 Scope

1.5.1 In Scope

- MCP server implementation (Go)
- GitLab file read/write operations
- SysML v2 API client integration
- stdio and HTTP transport mechanisms
- Container deployment
- Documentation and examples

1.5.2 Out of Scope (Future Work)

- AI benchmarking framework
- Multi-agent architectures
- Commercial integrations
- Full SysML v2 parser implementation

1.6 Document Structure

This book contains the complete systems engineering documentation:

- **Chapter 1**: SysML v2 background
- **Chapter 2**: Upstream research and prior art
- **Chapter 3**: Model Context Protocol
- **Chapter 4**: Systems Engineering Plan (SEP)
- **Chapter 5**: Work Breakdown Structure (WBS)
- **Chapter 6**: Stakeholder Analysis
- **Chapter 7**: System Requirements Specification (SyRS)
- **Chapter 8**: Architecture Design Description (ADD)
- **Chapter 9**: Verification & Validation Plan (VVP)
- **Chapter 10**: Implementation
- **Chapter 11**: Conclusions

Appendices include glossary, references, and traceability matrix.

Chapter 2

SysML v2: The Computational Revolution

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 3.7.3). 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) [4] 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. When OMG created SysML in 2006-2007, they built atop UML 2's metamodel—a reasonable choice for leveraging existing tool infrastructure but one that embedded software-centric assumptions and semantic ambiguity into a language intended for systems engineering.

2.3.1 The Semantic Precision Problem

OMG's SysML v2 requirements documentation states: "The semantics of SysML v1 are often defined in English rather than a more precise formal representation." This natural language approach meant different practitioners and tools could interpret the same model elements differently. The specification also "does not include a complete formal mapping between the concrete syntax and the abstract syntax," allowing diagrams that couldn't be unambiguously interpreted computationally.

2.3.2 Critical Language Gaps

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"

2.3.3 Broken Model Interchange

XMI (XML Metadata Interchange) failed in practice because "every tool supports UML differently and exports XMI differently." The OMG Model Interchange Working Group found systematic incompatibilities even for basic model elements. Diagrams—where practitioners estimated 90% of modeling work occurred—weren't included in XMI exchange.

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. KerML adopts "4D semantics" treating every occurrence as having both temporal extent (lifetime) and potentially spatial extent that can change over time.

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 (models diff, merge, branch)
- Programmatic access through standard parsing
- CI/CD pipeline integration
- AI-assisted modeling through LLM processing

The textual and graphical notations are complementary renderings of identical underlying models.

2.4.2 Calculations and Constraints

Calculations become first-class reusable definitions:

```
calc def TotalMass {
    in masses : MassValue[*];
    return result : MassValue = masses->sum();
}
```

Constraints express evaluable assertions:

```
constraint def PowerConstraint {
    in totalPower : PowerValue;
    in maxPower : PowerValue;
    totalPower <= maxPower
}
```

2.4.3 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. This structure enables automated verification—tools can systematically evaluate all requirements against a design model, reporting pass/fail status with full traceability.

2.5 The Systems Modeling API

The Systems Modeling API and Services specification [5] 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)
- Ad-hoc and saved queries

This API-first architecture enables the computational workflows our MCP server exposes:

- **Automated validation**: Traverse models evaluating requirement constraints
- **CI/CD integration**: Run model checks on every commit
- **Cross-tool workflows**: Exchange model data through standard REST calls
- **AI-assisted modeling**: Process textual notation through language models

2.6 Comparative Standards Landscape

SysML v2 hasn't eliminated the need for specialized standards. The ecosystem has evolved into a "hub and spoke" architecture:

Standard	Relationship to SysML v2	Use Case
Modelica	Complementary	Continuous physical simulation (DAEs)
AADL	Complementary	Timing/schedulability analysis for embedded systems
MARTE	Overlapping	Real-time constraint specification
Capella	Alternative	Method-integrated architecture tooling
UAF/DoDAF	Profile	Defense enterprise architecture views

For this project, the key insight is that SysML v2 serves as the architectural backbone while domain-specific tools provide specialized analysis. The MCP server focuses on SysML v2 as the integration point, with future extensions potentially bridging to domain tools.

2.7 Tool Ecosystem Status

As of early 2026, the tool landscape includes:

Commercial tools with announced SysML v2 support:

- CATIA Magic (Dassault Systèmes) - claims 100% conformance
- IBM Rhapsody, PTC Modeler, Ansys SAM, Sparx Enterprise Architect - various maturity

Open source alternatives:

- Eclipse SysON (Obeo/CEA) - web-based graphical environment [6]
- Syside (Sensmetry) - VS Code extension for textual editing [7]
- OMG Pilot Implementation - reference Eclipse and Jupyter environments [8]

Key gap: No lightweight Go-based tooling exists. The MCP server fills this gap with basic parsing and API proxy capabilities, aligning with GitLab's infrastructure-first approach (see Section 9.4 for rationale).

2.8 Implications for This Project

SysML v2's transformation from documentation to computation directly enables our MCP server architecture:

1. **Textual notation** allows storing models in GitLab repositories, enabling the file-based tools our MCP server provides
2. **Standardized API** means we can proxy to the reference implementation for full validation while providing lightweight local operations
3. **Formal semantics** ensure consistent interpretation across our tools and upstream validators
4. **Git-native workflows** align with GitLab's collaboration model and CI/CD integration

The practical path forward: use the MCP server for AI-augmented model creation and exploration, with GitLab for persistence and collaboration, delegating complex validation to the SysML v2 API Services when formal compliance checking is needed.

2.9 Further Reading

For deeper exploration of SysML v2 concepts, see the OMG specifications [3], [4], [5] and the upstream research in Section 3.1 covering:

- **KerML semantics**: 4D semantics, temporal/spatial extent modeling, first-order logic foundation
- **Expression language**: Collection operators, conditional expressions, feature chaining

- **Tool implementation**: Conformance testing, reference implementations (Section 3.3)
- **API integration**: REST endpoints, query language (Section 3.4)

Chapter 3

SysML v2 Upstream Research

3.1 Overview

This chapter documents research into upstream SysML v2 specifications and reference implementations. This research informs architecture decisions (Section 9.4) and identifies integration points for the MCP server. For SysML v2 conceptual background, see Section 2.1.

3.2 Official 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-Java-Client	Generated Java client (OpenAPI)	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

3.3 Pilot Implementation

3.3.1 Repository Structure

```
SysML-v2-Pilot-Implementation/
  kerml/                             # KerML examples
  sysml/                             # SysML examples
  sysml.library/                     # Standard library models
  org.omg.sysml/                     # Core EMF metamodel (Ecore)
  org.omg.kerml.xtext/               # KerML Xtext grammar
  org.omg.kerml.xtext.ide/           # KerML IDE support
  org.omg.kerml.xtext.ui/            # KerML Eclipse UI
  org.omg.sysml.xtext/               # SysML Xtext grammar
  org.omg.sysml.xtext.ide/           # SysML IDE support
  org.omg.sysml.xtext.ui/            # SysML Eclipse UI
  org.omg.kerml.expressions.xtext/   # Expression language grammar
  org.omg.sysml.interactive/         # Standalone interactive JAR
  org.omg.sysml.jupyter.kernel/      # Jupyter kernel
  org.omg.sysml.plantuml/            # PlantUML visualization
  org.omg.sysml.execution/           # Execution engine
  pom.xml                            # Maven build (Tycho)
```

3.3.2 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)

3.3.3 Standalone Usage

The `org.omg.sysml.interactive` module provides a standalone JAR:

```
mvn clean package
# JAR: org.omg.sysml.interactive/target/org.omg.sysml.interactive-*.jar
```

Key Class: `org.omg.sysml.interactive.SysMLInteractive`

- Parse SysML/KerML files
- Access the resolved AST/model
- Execute models

3.4 SysML v2 API Specification

3.4.1 REST API Endpoints

3.4.1.1 Projects

Method	Endpoint	Description
GET	`/projects`	List all projects
POST	`/projects`	Create project
GET	`/projects/{projectId}`	Get project by ID
PUT	`/projects/{projectId}`	Update project
DELETE	`/projects/{projectId}`	Delete project

3.4.1.2 Branches

Method	Endpoint	Description
GET	`/projects/{projectId}/branches`	List branches
POST	`/projects/{projectId}/branches`	Create branch
GET	`/projects/{projectId}/branches/{branchId}`	Get branch
DELETE	`/projects/{projectId}/branches/{branchId}`	Delete branch

3.4.1.3 Commits

Method	Endpoint	Description
GET	`/projects/{projectId}/commits`	List commits
POST	`/projects/{projectId}/commits`	Create commit
GET	`/projects/{projectId}/commits/{commitId}`	Get commit
GET	`/projects/{projectId}/commits/{commitId}/changes`	Get changes

3.4.1.4 Elements

Method	Endpoint	Description
GET	`/projects/{projectId}/commits/{commitId}/elements`	List elements
GET	`/projects/{projectId}/commits/{commitId}/elements/{elementId}`	Get element
GET	`/projects/{projectId}/commits/{commitId}/roots`	Get root elements

3.4.1.5 Queries

Method	Endpoint	Description
GET	`/projects/{projectId}/queries`	List saved queries
POST	`/projects/{projectId}/queries`	Create query
GET/POST	`/projects/{projectId}/query-results`	Execute a query

3.4.2 Query Constraints

Queries support:

- `PrimitiveConstraint` - single property constraints
- `CompositeConstraint` - AND/OR combinations of constraints

3.5 Reference API Server

3.5.1 Technology Stack

Component	Technology
Framework	Play Framework (Scala/Java)
Build Tool	sbt
Database	PostgreSQL
Java Version	JDK 11

3.5.2 Running Locally

```
# 1. Start PostgreSQL
docker run --name sysml2-postgres \
  -p 5432:5432 \
  -e POSTGRES_PASSWORD=mysecretpassword \
  -e POSTGRES_DB=sysml2 \
  -d postgres

# 2. Clone and run
git clone https://github.com/Systems-Modeling/SysML-v2-API-Services.git
cd SysML-v2-API-Services
sbt clean
sbt run

# 3. Access Swagger UI
open http://localhost:9000/docs/
```

3.6 Existing SysML Tools Analysis

Evaluated existing SysML v2 tooling to inform architecture decisions:

Tool	Type	Language	Pros	Cons
Pilot Implementation	Full parser	Java	Complete parsing, official	JVM dependency, complex
API Services	REST server	Java/Scala	Standard API, well-documented	Requires PostgreSQL
Jupyter Kernel	Interactive	Python	Good for exploration	Depends on JVM parser
Python Client	API client	Python	Generated, maintained	Requires running API server

Conclusion: No existing lightweight Go-based tooling. MCP server fills this gap with basic parsing and API proxy capabilities. See Section 9.5 for technology choices.

3.7 Integration Options

3.7.1 Option A: Pure API Proxy

```
LLM Client        MCP Server        SysML v2 API
                     (Go)                 (REST/HTTP)
```

Pros: Simple, no JVM dependency, aligns with upstream API spec
Cons: Requires running API server, no offline parsing

3.7.2 Option B: Embedded Parser (JVM)

```
LLM Client        MCP Server (JVM)
                        org.omg.sysml.interactive
                  SysML v2 API Client
```

Pros: Offline parsing, full AST access
Cons: Larger footprint, JVM dependency

3.7.3 Option C: Hybrid (Selected Approach)

```
 LLM Client       MCP Server       SysML v2 API
                    (Go)                (for validation)



                      Basic Parser
```

Go server with basic parsing, delegating full validation to API server.

Rationale: Balances deployment simplicity (single Go binary) with validation capability (API server for full SysML v2 compliance). Basic parsing handles common operations offline; API server handles complex validation when available.

3.8 Data Models

3.8.1 Element (JSON-LD)

```json
{
  "@id": "uuid",
  "@type": "PartDefinition",
  "name": "Vehicle",
  "qualifiedName": "Package1::Vehicle",
  "ownedElement": [{"@id": "..."}],
  "owner": {"@id": "..."}
}
```

3.8.2 Query

```json
{
  "@type": "Query",
  "select": ["@id", "name", "@type"],
  "where": {
    "@type": "CompositeConstraint",
    "operator": "and",
    "constraint": [
      {"@type": "PrimitiveConstraint", "property": "@type", "value": "PartDefinition"},
      {"@type": "PrimitiveConstraint", "property": "name", "value": "Vehicle"}
    ]
  }
}
```

3.9 MCP Tool Design

Based on upstream API capabilities, the MCP server implements tools in phases (see Section 8.3.1 for full requirements):

Phase	Tool	Description
0	`sysml_parse`	Parse SysML v2 text, extract elements (complete)
1	`gitlab_read_file`	Read .sysml file from GitLab repository
1	`gitlab_list_models`	List .sysml files in repo/directory
2	`sysml_validate`	Full validation via SysML v2 API server
2	`sysml_query`	Query elements by type/properties
2	`gitlab_commit`	Commit changes to GitLab
2	`gitlab_create_mr`	Create merge request

Resources follow MCP's URI-based access pattern:

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

3.10 Licensing Considerations

All repositories use **LGPL-3.0** (with GPL-3.0 for some components):

- Can link to LGPL libraries without making your code LGPL
- Modifications to LGPL code must be released under LGPL
- Compatible with building MIT-licensed MCP implementations

3.11 Industry Context: Agile Hardware Engineering

[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, with AI agents serving as "scribes" keeping models synchronized with engineering artifacts.

Aspect	INCOSE Formal	Agile Hardware
Reviews	Gated (SRR, PDR, CDR)	Continuous via PRs
Artifacts	Comprehensive docs	Lightweight models

Aspect	INCOSE Formal	Agile Hardware
Iteration	Spiral/Vee	Branch/merge cycles
AI Role	Analysis support	Model sync agent

Our position: This capstone follows INCOSE processes for academic rigor (see Section 5.1), while the MCP server aligns with the agile vision—enabling AI tools to interact with SysML models in Git via GitLab. The formal documentation proves we *can* do rigorous SE; the tooling enables teams to move faster when appropriate.

3.12 Key Findings

1. **Parser complexity**: Full SysML v2 parsing requires JVM (Xtext/EMF). A basic regex-based parser suffices for element extraction.
2. **API maturity**: The REST API spec is stable and well-documented. OpenAPI clients available for Java and Python.
3. **Deployment burden**: Running the reference API server requires PostgreSQL and JVM. Consider mock/stub for development.
4. **JSON-LD format**: All API responses use JSON-LD with `@id`, `@type` conventions. Must handle linked data patterns.
5. **Query capability**: The query language supports sophisticated filtering. Useful for AI-driven model exploration.
6. **Git-native workflows**: Industry momentum toward storing SysML v2 models in Git repositories, enabling software-style collaboration patterns [9].

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

```
                 MCP Host (AI Application)
              Claude Desktop / VS Code / etc.

                        MCP Client
      - Maintains connection to server
      - Discovers available tools/resources
      - Routes tool calls from LLM
```

```
                              JSON-RPC 2.0
                              (stdio or HTTP)


                          MCP Server

     Tools           Resources        Prompts

   sysml_parse     sysml://        (templates)
   gitlab_read     gitlab://
   sysml_valid
```

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, GitLab CI pipelines

The SysML v2 MCP server supports both transports, enabling local development with Claude Desktop and remote deployment for CI/CD integration.

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 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
- `gitlab://myorg/project/file/model.sysml` — file from GitLab

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 8.3.1.

4.4.1 Tool Definitions

Tool	Purpose	Inputs	Output
`sysml_parse`	Extract elements from SysML text	`source`	Element list (JSON)
`gitlab_read_file`	Read .sysml from GitLab	`project`, `path`, `ref`	File content
`gitlab_list_models`	List .sysml files in directory	`project`, `path`	File list
`sysml_validate`	Validate via SysML v2 API	`source`	Validation result
`sysml_query`	Query elements by type	`project`, `element_type`	Element list
`gitlab_commit`	Commit file changes	`project`, `branch`, `files`, `message`	Commit URL

4.4.2 Resource URIs

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

4.4.3 Typical Workflow

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

```
User: "What requirements are defined in this project?"



1. AI calls gitlab_list_models(project="myorg/vehicle")
   → Returns: ["requirements.sysml", "architecture.sysml"]



2. AI calls gitlab_read_file(path="requirements.sysml")
   → Returns: SysML v2 source text



3. AI calls sysml_parse(source=<file content>)
   → Returns: [{type: "RequirementDefinition", name: "..."}]



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
SysML v2 API unavailable	Fall back to local parsing (no full validation)
GitLab authentication failure	Return clear error with remediation steps
Invalid SysML syntax	Return parse errors with line numbers
Network timeout	Configurable timeout with retry guidance

4.5.2 Security

- GitLab Personal Access Token passed via environment variable (`GITLAB_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	GitLab CI service	Automated validation

See Section 9.10 for detailed deployment architecture.

Chapter 5

Systems Engineering Plan

5.1 Project Overview

5.1.1 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 GitLab for model storage and version control
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 NDIA GVSETS paper on AI-augmented MBSE
3. Establish open source project with community contribution potential

5.1.2 Scope

In Scope:

- MCP protocol implementation (tools, resources)
- GitLab API integration (read, list, commit, MR)
- SysML v2 API client (projects, elements, queries, validation)
- Basic SysML v2 textual parsing
- Container deployment support

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

Out of Scope:

- Full SysML v2 parser implementation (deferred to JVM-based solution)
- Multi-agent architectures
- GitHub/Gitea integration (future work)
- AI benchmarking framework (future work)

5.1.3 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

5.2 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.

```
Week:  1    2    3    4    5    6    7    8    9   10   11   12   13   14   15

        Concept    Design        Implementation              Validation
                                                             & Delivery

      SRR       PDR            Sprints           CDR              Final
     (Wk2)    (Wk4)                                    (Wk12)           (Wk15)
```

5.3 Technical Reviews

Review	Week	Purpose	Participants
SRR (System Requirements Review)	2	Baseline requirements, approve SEP	Andrew Dunn, Greg Pappas, Dr. Rapp

Review	Week	Purpose	Participants
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

5.4 Review Entry/Exit Criteria

5.4.1 SRR

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

5.4.2 PDR

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

5.4.3 CDR

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

5.5 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

Week	Dates	Phase	Key Activities	Deliverables
4	Feb 2-8	Design	Design review, V&V planning	PDR: ADD v1, VVP v1
5	Feb 9-15	Impl	Phase 1: GitLab integration	gitlab_read_file, gitlab_list_models
6	Feb 16-22	Impl	Phase 1 complete, Phase 2 start	GitLab tools working
7	Feb 23-Mar 1	Impl	SysML API integration	API client
8	Mar 2-8	Impl	GVSETS draft due (Mar 5), validation tools	Draft paper submitted
9	Mar 9-15	Impl	Phase 2: validation, query tools	sysml_validate, sysml_query
10	Mar 16-22	Impl	Phase 2 complete, HTTP transport	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

5.6 Key Milestones

Date	Milestone
Jan 12	Concept phase begins (Week 1)
Jan 18	Plan review with Greg Pappas and Dr. Rapp
Jan 25	SRR complete
Feb 8	PDR complete
Mar 5	GVSETS draft paper submitted
Apr 5	CDR complete
Apr 12	GVSETS final paper submitted
Apr 25	Capstone deliverables complete

5.7 Configuration Management

5.7.1 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

5.7.2 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)
Container Images	Git SHA + SemVer tags

5.7.3 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)

5.8 Risk Management

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

5.8.1 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

5.8.2 Risk Scoring

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

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

Risk Score: Likelihood × Impact (1-9)

5.8.3 Risk Register

ID	Risk Description	Category	L	I	Score	Treatment Strategy	Owner	Status
R1	SysML v2 API server difficult to deploy locally	Technical	2	3	6	**Avoid**: Implement GitLab-only tools first; API integration is Phase 2. Provide mock server for testing.	Andrew	Open
R2	GVSETS paper deadline aggressive given parallel implementation	Schedule	2	2	4	**Accept**: Submit draft even if incomplete; iterate on final version.	Andrew	Open
R3	Stakeholder availability for reviews limited	External	1	2	2	**Mitigate**: Schedule reviews early; use asynchronous review via MR comments.	Greg	Open

ID	Risk Description	Category	L	I	Score	Treatment Strategy	Owner	Status
R4	Go MCP SDK has undiscovered limitations	Technical	1	3	3	**Accept**: SDK is mature (Google co-maintained); fallback to TypeScript SDK if critical issue found.	Andrew	Open
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 Go binaries.	Andrew	Open
R6	Scope creep from additional feature requests	Schedule	2	2	4	**Avoid**: Defer AI benchmarking, multi-agent features to future work. Strict change control after SRR.	Greg	Open
R7	SysML v2 specification changes during project	External	1	2	2	**Accept**: Track upstream releases; design for extensibility. July 2025 OMG adoption provides stability.	Andrew	Open

5.8.4 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.

5.9 Review Status

This section tracks actual review completion status. Entry/exit criteria are defined in Section 5.3.

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)

5.9.1 Status Summary

Review	Target Date	Entry Criteria	Exit Criteria	Status
SRR	Jan 25, 2026	Pending	Pending	Not Started
PDR	Feb 8, 2026	Pending	Pending	Not Started
CDR	Apr 5, 2026	Pending	Pending	Not Started

5.9.2 Action Items

Review	Item	Owner	Due	Status
-	-	-	-	-

Action items will be recorded during and after each review.

Chapter 6

Work Breakdown Structure

6.1 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. Status reflects project state as of the current week.

6.2 WBS Tree

- **1.0 SysML v2 MCP Server 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**
    - 1.3.1 Phase 0: Core MCP Server
      - 1.3.1.1 Server scaffold (Go, MCP SDK)
      - 1.3.1.2 Basic sysml_parse tool
      - 1.3.1.3 Example resources
    - 1.3.2 Phase 1: GitLab Integration
      - 1.3.2.1 GitLab API client
      - 1.3.2.2 gitlab_read_file tool

        · 1.3.2.3 gitlab_list_models tool
        · 1.3.2.4 Authentication (PAT)
    * 1.3.3 Phase 2: SysML v2 API Integration
        · 1.3.3.1 SysML v2 API client
        · 1.3.3.2 sysml_validate tool
        · 1.3.3.3 sysml_query tool
        · 1.3.3.4 Write operations (commit, MR)
    * 1.3.4 Phase 3: HTTP Transport
        · 1.3.4.1 Streamable HTTP server
        · 1.3.4.2 CORS configuration
- **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
- **1.6 External Deliverables**
    * 1.6.1 GVSETS Abstract
    * 1.6.2 GVSETS Draft Paper (Mar 5)
    * 1.6.3 GVSETS Final Paper (Apr)
    * 1.6.4 Capstone Submission

6.3 1.1 Project Management

6.3.1 1.1.1 Project Planning

Per [1, Sec. 2.3.4.1].

- ☐ Define project objectives, scope, constraints - Section 5.1
- ☒ Develop breakdown structures (WBS) - This chapter
- ☐ Establish schedule with milestones - Section 5.5
- ☐ Generate SEMP/SEP - Section 5.2

6.3.2 1.1.2 Technical Reviews

Per [1, Sec. 2.1.4].

- ☐ SRR (Week 2) - Section 5.9
- ☐ PDR (Week 4) - Section 5.9
- ☐ CDR (Week 12) - Section 5.9

6.3.3 1.1.3 Risk Management

Per [1, Sec. 2.3.4.4].

- ☐ Identify risks and opportunities - Section 5.8
- ☐ Establish risk thresholds and categories - Section 5.8
- ☐ Define treatment strategies - Section 5.8

6.4 1.2 Systems Engineering

6.4.1 1.2.1 SEP

Per [1, Sec. 2.3.4.1].

- ☐ Life cycle model definition - Section 5.2
- ☐ Technical review entry/exit criteria - Section 5.3
- ☐ Configuration management approach - Section 5.7

6.4.2 1.2.2 Stakeholder Analysis

Per [1, Sec. 2.3.5.2].

- ☐ Identify stakeholders with interests - Section 7.1
- ☐ Establish stakeholder management approach - Section 7.2
- ☐ Develop operational concept - Section 7.3
- ☐ Define stakeholder needs - Section 7.4
- ☐ Transform needs to stakeholder requirements - Section 7.5

6.4.3 1.2.3 SyRS

Per [1, Sec. 2.3.5.3].

- ☐ Define functional boundary of system - Section 8.2
- ☐ Define system functions with performance - Section 8.3.1
- ☐ Define constraints (operational, regulatory) - Section 8.5
- ☐ Define verification criteria per requirement - Section 8.6
- ☐ Analyze requirements (complete, consistent, feasible) - Section 8.7

6.4.4 1.2.4 ADD

Per [1, Sec. 2.3.5.4].

- ☐ Identify architecture viewpoints - Section 9.1
- ☐ Define system context and boundary - Section 9.2
- ☐ Synthesize candidate architectures - Section 9.3
- ☐ Select architecture via trade study - Section 9.4
- ☐ Define interfaces (internal/external) - Section 9.8
- ☐ Allocate requirements to elements - Section 9.9

6.4.5 1.2.5 VVP

Per [1, Secs. 2.3.5.9, 2.3.5.11].

- ☐ Define verification scope and strategy - Section 10.1
- ☐ Select verification methods per requirement - Section 10.2
- ☐ Define verification success criteria - Section 10.3
- ☐ Plan enabling systems (test tools, CI) - Section 10.4
- ☐ Define validation approach (stakeholder acceptance) - Section 10.8

6.4.6 1.2.6 RTM

Per [1, Sec. 3.2.3].

- ☐ Stakeholder needs → Stakeholder requirements - Section 15.1
- ☐ Stakeholder requirements → System requirements - Section 15.2
- ☐ System requirements → Architecture elements - Section 15.3
- ☐ System requirements → Test cases - Section 15.4

6.5 1.3 Software Development

6.5.1 Phase 0: Core MCP Server (Complete)

- ☒ **1.3.1.1** Server scaffold (Go, MCP SDK)
- ☒ **1.3.1.2** Basic `sysml_parse` tool
- ☒ **1.3.1.3** Example resources

6.5.2 Phase 1: GitLab Integration

- ☐ **1.3.2.1** GitLab API client - Week 5
- ☐ **1.3.2.2** `gitlab_read_file` tool - Week 5
- ☐ **1.3.2.3** `gitlab_list_models` tool - Week 5
- ☐ **1.3.2.4** Authentication (PAT support) - Week 6

6.5.3 Phase 2: SysML v2 API Integration

- ☐ **1.3.3.1** SysML v2 API client - Week 7
- ☐ **1.3.3.2** `sysml_validate` tool - Week 8
- ☐ **1.3.3.3** `sysml_query` tool - Week 8
- ☐ **1.3.3.4** Write operations (`gitlab_commit`, `gitlab_create_mr`) - Week 9

6.5.4 Phase 3: HTTP Transport

- ☐ **1.3.4.1** Streamable HTTP server - Week 9
- ☐ **1.3.4.2** CORS configuration - Week 9

6.6 1.4 Infrastructure

- ☒ **1.4.1** Repository Setup - Complete
- ☐ **1.4.2** CI/CD Pipeline (software)
- ☒ **1.4.3** CI/CD Pipeline (documentation) - Complete (HTML + PDF)
- ☐ **1.4.4** Container Build & Registry

6.7 1.5 Documentation

- ☒ **1.5.1** Quarto Book Setup - Complete
- ☐ **1.5.2** Chapter Authoring - Ongoing
- ☒ **1.5.3** GitLab Pages Deployment - Complete
- ☐ **1.5.4** Software README/CONTRIBUTING

6.8 1.6 External Deliverables

- ☐ **1.6.1** GVSETS Abstract
- ☐ **1.6.2** GVSETS Draft Paper - Due Mar 5
- ☐ **1.6.3** GVSETS Final Paper - Due Apr 7
- ☐ **1.6.4** Capstone Submission - Due Apr 14

6.9 Milestones

Week	Date	Milestone	Status
1	Jan 12-18	Plan finalized, repos set up	Complete
2	Jan 19-25	**SRR** - SEP, SyRS baselined	In Progress
4	Feb 2-8	**PDR** - ADD, VVP approved	Pending
7	Mar 2-8	GVSETS draft submitted	Pending
10	Mar 23-29	Full tool suite complete	Pending
12	Mar 30-Apr 5	**CDR** - V&V complete	Pending
15	Apr 20-25	Final delivery	Pending

6.10 Risk Summary

See Section 5.8 for full risk register.

ID	Risk	Likelihood	Impact	Mitigation
R1	SysML v2 API server difficult to deploy	Medium	High	GitLab-only tools first; API is Phase 2
R2	GVSETS deadline aggressive	Medium	Medium	Submit draft even if incomplete
R4	Go MCP SDK limitations	Low	High	SDK is mature; fallback to TypeScript
R5	Container testing blocked locally	High	Low	CI-only validation; document in VVP
R6	Scope creep	Medium	Medium	Defer benchmarking to future work

Chapter 7

Stakeholder Analysis

7.1 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	Category	Interest	Influence	Success Criteria
GitLab	Sponsor	Market positioning in SE/defense space	High	Visible GitLab integration, open source contribution
Academic Advisor	Authority	SE process rigor, academic standards	High	Complete SE artifacts, proper methodology
Capstone Collaborator	Team	Course completion, DoD relevance	High	Shared workload, defensible deliverables
Open Source Community	User	Usable tool, extensibility	Medium	Working software, good documentation
Defense/Aerospace Users	User	Practical utility, compliance	Medium	Solves real workflow problems
INCOSE/SE Community	Influencer	Advancing AI4SE	Low	Novel contribution, reproducible results

Stakeholder	Category	Interest	Influence	Success Criteria
SysML v2 Implementers	Supplier	Adoption of standard	Low	Correct API usage, spec compliance

7.1.1 Stakeholder Analysis Matrix

Stakeholder	Power	Interest	Strategy
GitLab	High	High	Manage Closely
Academic Advisor	High	High	Manage Closely
Capstone Collaborator	High	High	Manage Closely
Open Source Community	Low	High	Keep Informed
Defense/Aerospace Users	Low	High	Keep Informed
INCOSE/SE Community	Low	Medium	Monitor
SysML v2 Implementers	Low	Low	Monitor

7.2 Team Roles

Role	Person	Affiliation	Primary Responsibilities
Technical Lead	Andrew Dunn	GitLab Public Sector	Software implementation, CI/CD, architecture
SE Lead	Greg Pappas	DoD, Army, AFC-DEVCOM	Requirements, V&V Plan, SEP, review facilitation
Advisor	Dr. Stephen Rapp	Wayne State University, ISE	Technical reviews, capstone evaluation

7.2.1 Responsibility Matrix (RACI)

WBS Element	Andrew	Greg	Dr. Rapp
1.1.1 Planning	R	C	I
1.1.2 Technical Reviews	R	R	A
1.2.1 SEP	C	R	A
1.2.2 Stakeholder Analysis	C	R	I
1.2.3 SyRS	C	R	A
1.2.4 ADD	R	C	A
1.2.5 VVP	C	R	A
1.2.6 RTM	C	R	I
1.3.x Software Dev	R	I	I
1.4.x Infrastructure	R	I	I

WBS Element	Andrew	Greg	Dr. Rapp
1.5.1-3 Quarto Book	C	R	I
1.5.4 Software Docs	R	C	I
1.6.x Papers	R	R	C

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

7.3 Operational Concept

Per [1, Sec. 2.3.5.2], the operational concept describes how users will interact with the system in its intended environment.

7.3.1 System Context

The SysML v2 MCP Server operates as middleware between AI assistants (LLM clients) and MBSE infrastructure (GitLab repositories, SysML v2 API servers).

```
                      User Environment

      Claude            VS Code          Custom AI
      Desktop         + Continue        Application



                              MCP Protocol


                      SysML v2 MCP
                         Server
                     (stdio or HTTP)









      GitLab          SysML v2          Local Files
     (gitlab.com      API Server         (.sysml)
     or self)         (optional)
```

7.3.2 Use Cases

UC-1: AI-Assisted Model Review

1. Systems engineer opens Claude Desktop with MCP server configured
2. Engineer asks: "List all requirement definitions in the vehicle model"
3. MCP server calls `gitlab_read_file` to fetch model from GitLab
4. MCP server calls `sysml_parse` to extract elements
5. Claude presents findings and suggests improvements
6. Engineer requests changes; Claude uses `gitlab_commit` to save

UC-2: Model Validation in CI/CD

1. Developer commits SysML v2 model changes to GitLab
2. CI pipeline starts MCP server in HTTP mode
3. Pipeline calls `sysml_validate` via HTTP
4. Validation results reported in merge request
5. Reviewer sees AI-generated model summary

UC-3: Exploratory Model Query

1. New team member needs to understand existing model
2. Opens AI assistant with MCP server connected
3. Asks natural language questions about model structure
4. MCP server uses `sysml_query` to search elements
5. AI explains model architecture, relationships

7.3.3 Operational Modes

Mode	Transport	Use Case	Authentication
Local Development	stdio	Individual engineer with Claude/VS Code	GitLab PAT in environment
Team Server	HTTP	Shared server for team access	GitLab PAT per request
CI/CD Pipeline	HTTP	Automated validation in GitLab CI	GitLab CI_JOB_TOKEN

7.4 Stakeholder Needs

Per [1, Sec. 2.3.5.2], stakeholder needs are statements of what stakeholders require from the system.

7.4.1 Need Statement Format

[**SN-XXX**] As a [stakeholder], I need [capability] so that [benefit].

7.4.2 GitLab (Sponsor)

[**SN-001**] As GitLab, I need the MCP server to integrate with GitLab APIs so that GitLab is positioned as the platform for AI-augmented MBSE.

[**SN-002**] As GitLab, I need the project to be open source so that it contributes to the GitLab ecosystem and community.

[**SN-003**] As GitLab, I need CI/CD integration showcased so that the DevSecOps value proposition extends to systems engineering.

7.4.3 Academic/Capstone (Authority)

[**SN-004**] As the academic advisor, I need the project to follow INCOSE SE processes so that students demonstrate proper methodology.

[**SN-005**] As the academic advisor, I need formal technical reviews (SRR, PDR, CDR) so that the capstone meets academic rigor requirements.

[**SN-006**] As the capstone collaborator, I need shared workload distribution so that both team members contribute equitably.

7.4.4 Technical Users

[**SN-007**] As a systems engineer, I need easy installation (single binary) so that I can start using the tool without complex setup.

[**SN-008**] As a systems engineer, I need clear documentation with examples so that I understand how to use the MCP tools.

[**SN-009**] As a DevOps engineer, I need container deployment support so that I can integrate the server into existing infrastructure.

[**SN-010**] As a systems engineer, I need to query SysML v2 models through natural language so that I can explore models without learning query syntax.

7.4.5 Defense/Aerospace Users

[**SN-011**] As a defense contractor, I need support for self-hosted GitLab so that I can use the tool in air-gapped environments.

[**SN-012**] As a defense systems engineer, I need model validation against SysML v2 spec so that I ensure model compliance.

7.4.6 Needs to Requirements Traceability

Stakeholder Need	Stakeholder Requirement(s)	Rationale
SN-001	SR-001	Direct derivation
SN-002	SR-002	Direct derivation
SN-003	SR-003	Direct derivation
SN-004	SR-004	Direct derivation
SN-005	SR-005	Direct derivation
SN-006	-	Process constraint, not system requirement
SN-007	SR-006	Direct derivation
SN-008	SR-007, SR-008	Direct derivation
SN-009	SR-009	Direct derivation
SN-010	SR-012	Direct derivation
SN-011	SR-010	Direct derivation
SN-012	SR-011	Direct derivation

Complete traceability matrix in Section 15.1.

7.5 Stakeholder Requirements

Per [1, Sec. 2.3.5.2], stakeholder requirements are derived from stakeholder needs and expressed in technical terms.

7.5.1 Requirement Format

[**SR-XXX**] The system shall [capability] [condition] [constraint].

Trace: Derived from [SN-XXX]

7.5.2 Platform Requirements

[**SR-001**] The system shall integrate with GitLab REST API for repository operations.
Trace: SN-001

[**SR-002**] The system shall be licensed under the MIT open source license.
Trace: SN-002

[**SR-003**] The system shall provide GitLab CI/CD integration examples.
Trace: SN-003

7.5.3 Process Requirements

[**SR-004**] The project shall produce SE artifacts per INCOSE Handbook guidance (SEP, SyRS, ADD, VVP, RTM).
Trace: SN-004

[**SR-005**] The project shall conduct SRR, PDR, and CDR technical reviews with documented entry/exit criteria.
Trace: SN-005

7.5.4 Usability Requirements

[**SR-006**] The system shall be distributable as a single static binary requiring no external dependencies.
Trace: SN-007

[**SR-007**] The system shall include README with installation and configuration instructions.
Trace: SN-008

[**SR-008**] The system shall provide example SysML v2 models demonstrating tool capabilities.
Trace: SN-008

[**SR-009**] The system shall be distributable as an OCI-compliant container image.
Trace: SN-009

7.5.5 Functional Requirements

[**SR-010**] The system shall support self-hosted GitLab instances via configurable base URL.
Trace: SN-011

[**SR-011**] The system shall validate SysML v2 model syntax via API integration.
Trace: SN-012

[**SR-012**] The system shall parse SysML v2 textual notation and extract element information.
Trace: SN-010

Chapter 8

System Requirements Specification

8.1 Overview

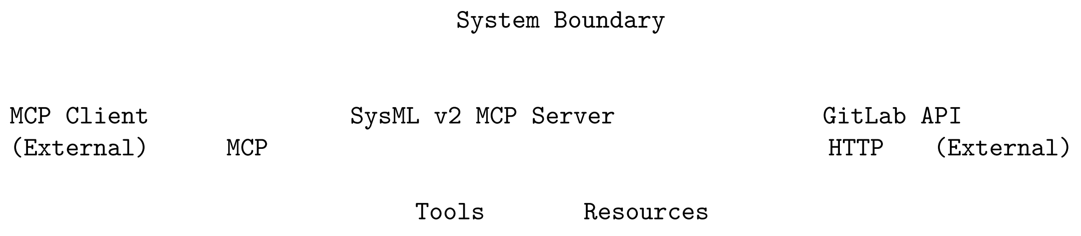
This chapter defines the system requirements for the SysML v2 MCP Server per [1, Sec. 2.3.5.3]. Requirements are organized by functional area and traced to stakeholder requirements.

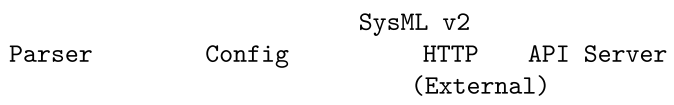
8.2 System Scope and Boundary

8.2.1 System Definition

The SysML v2 MCP Server is a software system that implements the Model Context Protocol (MCP) to provide AI assistants with programmatic access to SysML v2 models stored in GitLab repositories and managed by SysML v2 API servers.

8.2.2 System Boundary





8.2.3 External Interfaces

Interface	Type	Protocol	Description
MCP Client	Input	MCP over stdio/HTTP	AI assistant sending requests
GitLab API	Output	REST/HTTP	Repository file operations
SysML v2 API	Output	REST/HTTP	Model validation and queries
Configuration	Input	Environment variables	Server configuration

⚠ TODO: Interface Requirements

Interface requirements need formal IR-xxx identifiers and detailed specifications. This will be addressed in the requirements pass.

8.3 Functional Requirements

8.3.1 MCP Protocol

8.3.2 GitLab Integration

8.3.3 SysML v2 Operations

8.4 Non-Functional Requirements

8.4.1 Performance

ID	Requirement	Priority	Verification	Trace
FR-MCP-001	The server SHALL implement MCP protocol version 2024-11-05	High	Test	SR-001
FR-MCP-002	The server SHALL support stdio transport	High	Test	SR-006
FR-MCP-003	The server SHALL support HTTP transport	Medium	Test	SR-003
FR-MCP-004	The server SHALL respond to initialize requests with server capabilities	High	Test	SR-001
FR-MCP-005	The server SHALL list available tools via tools/list	High	Test	SR-001
FR-MCP-006	The server SHALL list available resources via resources/list	High	Test	SR-001

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

ID	Requirement	Priority	Verification	Trace
FR-SYS-001	The server SHALL parse SysML v2 textual notation	High	Test	SR-012
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

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	-

8.4.2 Security

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	-

8.4.3 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

ID	Requirement	Priority	Verification	Trace
NFR-DEP-004	The server SHALL support macOS operating systems (amd64, arm64 architectures)	High	Test	SR-006

8.4.4 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

8.5 Constraints and Assumptions

8.5.1 Design Constraints

ID	Constraint	Rationale
DC-001	The server SHALL be implemented in Go	Aligns with GitLab ecosystem, single binary deployment
DC-002	The server SHALL use the official MCP Go SDK	Ensures protocol compliance, Google co-maintained
DC-003	The server SHALL use go-gitlab client library	Mature library, supports gitlab.com and self-hosted
DC-004	Container builds SHALL use Buildah/Podman	OCI-compliant, rootless, CI-friendly

8.5.2 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	GitLab PAT required for private repositories	Public repos accessible without authentication

8.5.3 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	Go MCP SDK supports required features	May need SDK contributions or workarounds
A-004	GitLab API stable for file operations	Low risk - mature API

8.6 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

8.6.1 Verification Summary

Category	Test	Demonstration	Inspection	Analysis	Total
FR-MCP	6	0	0	0	6
FR-GL	6	0	1	0	7
FR-SYS	4	0	1	0	5
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	**22**	**2**	**6**	**0**	**30**

8.7 Requirements Analysis

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

8.7.1 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

8.7.2 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

8.7.3 Feasibility Assessment

Requirement Area	Feasibility	Risk
MCP Protocol	High	SDK provides implementation
GitLab Integration	High	Mature go-gitlab library
SysML v2 Parsing	Medium	Basic parser feasible; full parser out of scope
SysML v2 API	Medium	Depends on API server availability
Container Deployment	High	Standard Go cross-compilation

8.7.4 TBD Items

Item	Target Resolution	Owner
OAuth authentication scope	PDR (Week 4)	Andrew
SysML v2 API error handling patterns	Week 7	Andrew
HTTP transport security (TLS) requirements	PDR (Week 4)	Andrew

8.8 Tool Definitions

8.8.1 Phase 0 (Complete)

Tool	Description	Status
`sysml_parse`	Parse SysML v2 textual notation and extract element information	Complete

8.8.2 Phase 1 (GitLab)

Tool	Description	Status
`gitlab_read_file`	Read .sysml file from GitLab repository	Planned
`gitlab_list_models`	List .sysml files in a repo/directory	Planned

8.8.3 Phase 2 (SysML API)

Tool	Description	Status
`sysml_validate`	Full validation via SysML v2 API server	Planned
`sysml_query`	Query model elements by type/properties	Planned
`gitlab_commit`	Commit changes to GitLab	Planned
`gitlab_create_mr`	Create merge request	Planned

8.9 Resource Definitions

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

Chapter 9

Architecture Design Description

9.1 Architecture Viewpoints

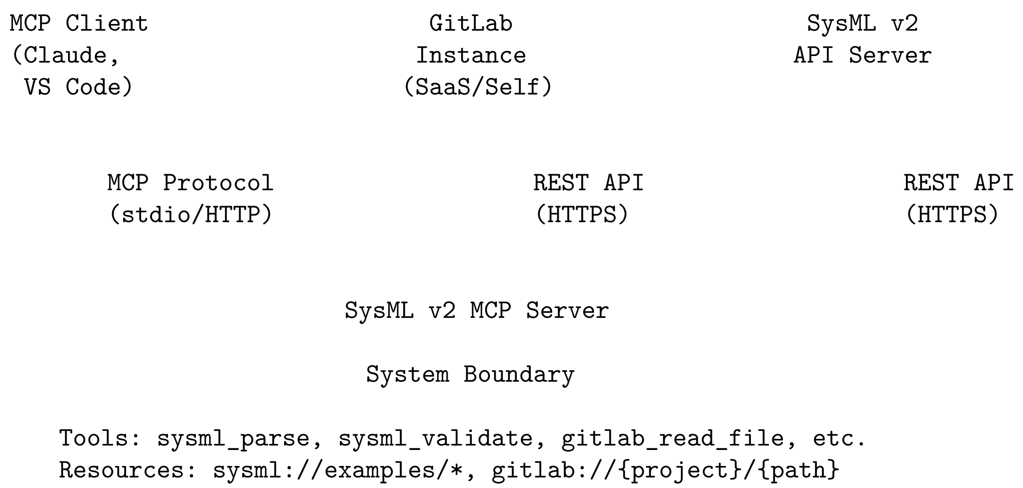
Per [1, Sec. 2.3.5.4], architecture viewpoints frame stakeholder concerns.

Viewpoint	Stakeholder Concern	Addressed In
Functional	What functions does the system perform?	Section 9.7
Information	What data flows through the system?	Section 9.8
Physical	What components exist and how deployed?	Section 9.10
Development	How is the system built and maintained?	Section 9.5, Section 9.6

9.2 System Context Diagram

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

```
External Systems
```


MCP Client (Claude, VS Code)
GitLab Instance (SaaS/Self)
SysML v2 API Server
MCP Protocol (stdio/HTTP)
REST API (HTTPS)
REST API (HTTPS)
SysML v2 MCP Server
System Boundary
Tools: sysml_parse, sysml_validate, gitlab_read_file, etc.
Resources: sysml://examples/*, gitlab://{project}/{path}

External Interfaces:

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

9.3 Architecture Alternatives

Per [1, Sec. 2.3.5.4], candidate architectures were evaluated before selection. Detailed analysis in Section 3.6.

Alternative	Description	Evaluation
Python + FastMCP	Python-based MCP server	Rejected: additional runtime dependency
TypeScript + official SDK	Node.js-based server	Rejected: heavier deployment footprint

Alternative	Description	Evaluation
Go + go-sdk	Single static binary	**Selected**: minimal dependencies, fast builds
Rust + custom impl	Rust-based server	Rejected: no official SDK, higher complexity

9.4 Architecture Selection Rationale

The Go-based architecture was selected based on:

Criterion	Weight	Go	Python	TypeScript
Single binary deployment	High			
Container size	Medium	~20MB	~200MB	~150MB
GitLab client maturity	High	(go-gitlab)		
Official MCP SDK	High	(Google co-maintained)		
Team expertise	Medium			

Decision: Go provides optimal balance of deployment simplicity, performance, and SDK support.

9.5 Technology Stack

Component	Technology	Rationale
Language	Go 1.23+	Single static binary, fast builds, excellent GitLab client library
MCP SDK	github.com/modelcontextprotocol/go-sdk v1.2.0	Official SDK, Google co-maintained
GitLab Client	github.com/xanzy/go-gitlab	Mature, supports both gitlab.com and self-hosted
Transport	stdio + HTTP	stdio for local dev, HTTP for remote/CI deployment
Container	Buildah/Podman	OCI-compliant, rootless, CI-friendly

Component	Technology	Rationale
Documentation	Quarto	Markdown-native, GitLab Pages compatible, PDF export

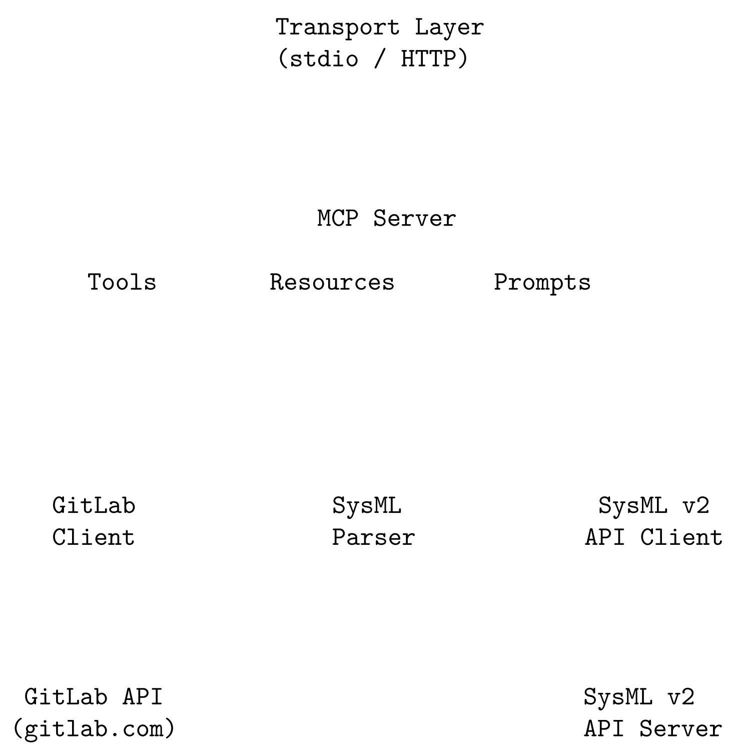
9.6 Repository Structure

```
open-mcp-sysml/                     # GitLab Group
  plan/                            # Capstone SE Documentation (Quarto Book)
      _quarto.yml
      index.qmd
      chapters/
      appendices/
      .gitlab-ci.yml

  open-mcp-sysml/                 # Software Product
      cmd/
         sysmlv2-mcp/
             main.go
      internal/
         server/                  # MCP server implementation
         gitlab/                  # GitLab API integration
         sysml/                   # SysML v2 API client
         config/                  # Configuration handling
      examples/
         models/                  # Example .sysml files
      testdata/
      Containerfile
      .gitlab-ci.yml
      go.mod
      README.md
      CONTRIBUTING.md
      LICENSE
```

9.7 Component Architecture

```
                MCP Client (Claude, etc.)
```



9.8 Interface Definitions

9.8.1 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

9.8.2 GitLab Interface

```go
type GitLabClient interface {
    ReadFile(project, path, ref string) ([]byte, error)
    ListFiles(project, path, ref string) ([]string, error)
    CreateCommit(project, branch, message string, actions []FileAction) error
    CreateMergeRequest(project, source, target, title string) (*MR, error)
}
```

9.8.3 SysML v2 API Interface

```
type SysMLAPIClient interface {
    ListProjects() ([]Project, error)
    GetElement(projectID, commitID, elementID string) (*Element, error)
    Query(projectID, commitID string, query Query) ([]Element, error)
    Validate(content string) (*ValidationResult, error)
}
```

9.9 Requirements Allocation

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

Requirement	Architecture Element	Package
FR-MCP-001	MCP Server	cmd/sysmlv2-mcp
FR-MCP-002	Tools Handler	internal/server
FR-MCP-003	HTTP Transport	internal/server
FR-GL-001, FR-GL-002	GitLab Client	internal/gitlab
FR-SYS-001, FR-SYS-002	SysML Parser	internal/sysml
FR-SYS-003, FR-SYS-004	SysML API Client	internal/sysml
NFR-DEP-001	Build Configuration	go.mod, Makefile
NFR-DEP-002	Container Image	Containerfile

9.10 Deployment Architecture

Per [1, Sec. 2.3.5.4], deployment architecture defines how the system operates in its environment.

9.10.1 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

9.10.2 Container Deployment

```
                  Host System

               Container (OCI)

  sysmlv2-mcp binary
  - Listens on :8080
  - Env: GITLAB_TOKEN, SYSML_API_URL


                    Port 8080


                 Port Mapping


               External Clients
```

9.11 Development Environment Constraints

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

Mitigation:

- Local development uses `go build` and `go test` directly
- MCP protocol testing via stdio (no containers required)
- Container builds run exclusively in GitLab CI
- HTTP transport testing deferred to CI or Linux machine

This constraint is documented in the V&V Plan with acceptance criteria adjusted accordingly.

Chapter 10

Verification & Validation Plan

10.1 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).

Method	Scope	Environment
Unit Testing	Go packages	Local (go 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

10.2 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

Method	Abbreviation	Description	When Used
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

10.2.1 Verification Method Assignment

Requirement	Method	Rationale
FR-MCP-001	T, D	Test server initialization, demonstrate with client
FR-MCP-002	T	Test tool enumeration and execution
FR-GL-001, FR-GL-002	T	Test file read from GitLab repositories
FR-SYS-001	T	Test parsing of SysML v2 syntax
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

10.3 Acceptance Criteria

Requirement Category	Verification Method	Acceptance Criteria
MCP Protocol Compliance	Integration test	Server initializes, lists tools/resources, executes tools
GitLab Integration	Integration test	Read files from gitlab.com and self-hosted
SysML v2 Validation	System test	Validates correct/incorrect SysML syntax

Requirement Category	Verification Method	Acceptance Criteria
Container Deployment	CI pipeline	Image builds, runs, responds to MCP requests
Documentation	Inspection	Quarto renders, deploys to GitLab Pages

10.4 Enabling Systems

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

Enabling System	Purpose	Responsibility
Go Test Framework	Unit and integration testing	Built into Go toolchain
GitLab CI/CD	Automated pipeline execution	GitLab SaaS 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

10.4.1 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

10.5 Test Cases

10.5.1 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
TC-MCP-005	Read sysml://examples/hello	Returns vehicle model content	T

10.5.2 GitLab Integration Tests

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

10.5.3 SysML Parsing Tests

ID	Test Case	Expected Result	Method
TC-SYS-001	Parse package declaration	Extracts package name	T
TC-SYS-002	Parse part definition	Extracts part def name	T

ID	Test Case	Expected Result	Method
TC-SYS-003	Parse requirement definition	Extracts requirement name	T
TC-SYS-004	Parse nested elements	Extracts all element names	T
TC-SYS-005	Parse empty input	Returns empty element list	T

10.6 Known Limitations

1. **Container testing**: Cannot be performed locally on macOS; relies on CI
2. **HTTP transport**: Requires CI service containers or Linux machine
3. **SysML v2 API**: Requires running API server; may use mock for some tests

10.7 CI/CD Verification Pipeline

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

10.7.1 Pipeline Stages

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

10.7.2 Test Stage

```
test:
  stage: test
  image: golang:1.23-alpine
  script:
    - go test -v -race ./...
  rules:
    - if: $CI_PIPELINE_SOURCE == "merge_request_event"
    - if: $CI_COMMIT_BRANCH == "main"
```

10.7.3 Integration Test Stage

```yaml
integration:
  stage: integration
  image: golang:1.23-alpine
  variables:
    GITLAB_TOKEN: $CI_JOB_TOKEN
  script:
    - go test -v -tags=integration ./...
  rules:
    - if: $CI_COMMIT_BRANCH == "main"
```

10.7.4 Container Test Stage

```yaml
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"
```

10.8 Validation Approach

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

10.8.1 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

10.8.2 Validation Schedule

Milestone	Week	Validation Activity
SRR	2	Requirements validated with stakeholders
PDR	4	Architecture validated against requirements
CDR	12	Implementation validated, acceptance tests pass
Final	15	Stakeholder acceptance, capstone submission

10.9 Review Verification

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

Chapter 11

Implementation

11.1 Status

Implementation details will be documented here as the software is developed during Phases 1-3.

For project structure and task tracking, see Section 6.2.

11.2 Phase Summary

Phase	Description	Target	Status
Phase 0	Core MCP Server	Complete	Complete
Phase 1	GitLab Integration	Week 5-6	Planned
Phase 2	SysML v2 API Integration	Week 7-9	Planned
Phase 3	HTTP Transport	Week 9	Planned

11.3 Phase 0: Core MCP Server (Complete)

Phase 0 established the basic MCP server scaffold:

- Go project with MCP SDK integration
- `sysml_parse` tool for basic element extraction
- Example resources (`sysml://examples/hello`, `sysml://examples/requirements`)
- Containerfile for deployment
- stdio transport

11.4 Phases 1-3

Detailed implementation notes will be added as each phase is executed. See Section 6.5.2, Section 6.5.3, and Section 6.5.4 for task tracking.

Chapter 12

Conclusions

12.1 Summary

This project delivers an open source SysML v2 MCP server that bridges AI assistants with Model-Based Systems Engineering workflows. The key contributions are:

1. **Working Software**: MCP server with GitLab integration and SysML v2 API support
2. **Academic Deliverables**: SE documentation demonstrating INCOSE principles
3. **External Publication**: NDIA GVSETS paper on AI-augmented MBSE

12.2 Lessons Learned

To be completed after project execution.

12.3 Future Work

12.3.1 Deferred to Future Releases

- AI benchmarking framework for MBSE tasks
- Multi-agent architectures with MCP communication
- Additional Git providers (GitHub, Gitea)
- Full SysML v2 parser implementation
- OAuth/OIDC authentication

12.3.2 Research Directions

- SysML v2-specific evaluation metrics

- Requirements-to-model generation
- Natural language model queries
- CI/CD integration patterns

12.4 References

See Section 14.1 for complete bibliography.

12.5 Acknowledgments

To be added.

Chapter 13

Glossary

Term	Definition
ADD	Architecture Design Description
CDR	Critical Design Review
CI/CD	Continuous Integration / Continuous Deployment
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
V&V	Verification and Validation
VVP	Verification and Validation Plan
WBS	Work Breakdown Structure

Chapter 14

References

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

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

[6] Eclipse Foundation, "Eclipse SysON." 2025. Available: https://eclipse.dev/syson/

[7] Sensmetry, "SysIDE." 2025. Available: https://sensmetry.com/syside/

[8] Systems Modeling, "SysML v2 pilot implementation." 2025. Available: https://github.com/Systems-Modeling/SysML-v2-Pilot-Implementation

[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/

Chapter 15

Requirements Traceability Matrix

Note

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

15.1 Stakeholder Needs to Stakeholder Requirements

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

15.2 Stakeholder Requirements to System Requirements

15.3 System Requirements to Architecture Elements

System Requirement	Architecture Element	Package
FR-MCP-001 through FR-MCP-006	MCP Server	cmd/sysmlv2-mcp, internal/server

System Requirement	Architecture Element	Package
FR-GL-001 through FR-GL-007	GitLab Client	internal/gitlab
FR-SYS-001, FR-SYS-002	SysML Parser	internal/sysml
FR-SYS-003, FR-SYS-004	SysML API Client	internal/sysml
NFR-DEP-001	Build Configuration	go.mod, Makefile
NFR-DEP-002	Container Image	Containerfile

15.4 System Requirements to Test Cases

15.5 WBS to Requirements

WBS	Requirements Addressed
1.3.1	FR-MCP-001 through FR-MCP-006, FR-SYS-001, FR-SYS-002
1.3.2	FR-GL-001 through FR-GL-007
1.3.3	FR-SYS-003, FR-SYS-004
1.3.4	FR-MCP-003
1.4.4	NFR-DEP-002

Stakeholder Need	Stakeholder Requirement	Rationale
SN-001 (GitLab API integration)	SR-001	Direct derivation
SN-002 (Open source)	SR-002	Direct derivation
SN-003 (CI/CD integration)	SR-003	Direct derivation
SN-004 (INCOSE process)	SR-004	Direct derivation
SN-005 (Technical reviews)	SR-005	Direct derivation
SN-006 (Shared workload)	-	Process constraint, not system requirement
SN-007 (Single binary)	SR-006	Direct derivation
SN-008 (Documentation)	SR-007, SR-008	Direct derivation
SN-009 (Container deployment)	SR-009	Direct derivation
SN-010 (Natural language query)	SR-012	Direct derivation
SN-011 (Self-hosted GitLab)	SR-010	Direct derivation
SN-012 (Model validation)	SR-011	Direct derivation

Stakeholder Requirement	System Requirement	Allocation
SR-001 (GitLab API)	FR-GL-001, FR-GL-002, FR-GL-003, FR-GL-004, FR-GL-005	GitLab 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 GitLab)	FR-GL-004	GitLab 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

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-GL-001	TC-GL-001, TC-GL-002	Test
FR-GL-002	TC-GL-003	Test
FR-GL-004	TC-GL-004	Test
FR-GL-005	TC-GL-005	Test
FR-SYS-001	TC-SYS-001 through TC-SYS-005	Test
NFR-DEP-001	CI build job	Test, Analysis
NFR-DEP-002	CI container job	Test
NFR-DOC-001	CI pages job	Inspection