0x06tudio
Progetto didattico di Ingegneria del Software
Università degli Studi di Padova

0x06tudio@gmail.com

Studio Tecnologico: Ingestion e Parsing di Diagrammi Architetturali

Approfondimento Tecnologico Modulo 3 – Progetto SpecCheck (Capitolato C9)

Informazioni sul documento

Versione:	0.1.0
Stato:	Approvato
Data ultimo aggiornamento:	2026/11/17
Uso:	Interno
Destinatari:	0x06tudio

Registro delle modifiche

Versione	Data	Descrizione	Autore	Verificatore
0.1.0	2026/11/17	Stesura del documento	Nicolò Beltrame	Catalin Chirilov

Indice

1 Introduzione **3**
1.1 Scopo del documento . . . 3
1.2 Riferimenti . . . 3
1.2.1 Riferimenti normativi . . . 3
1.2.2 Riferimenti informativi . . . 3

2 Contesto e Sfide nel Capitolato C9 **3**

3 Analisi delle Tecnologie Chiave **4**
3.1 Formati text-based per diagrammi UML . . . 4
3.1.1 Mermaid . . . 4
3.1.2 PlantUML . . . 5
3.1.3 XMI (XML Metadata Interchange) . . . 5
3.2 Vision Language Models (VLM) . . . 5
3.3 Pipeline di estrazione strutturata . . . 6
3.4 Collegamento con gli altri moduli . . . 7

4 Architettura Proposta e Proof of Concept (PoC) **7**
4.1 Flusso operativo del PoC . . . 7
4.2 Esempio di diagramma in ingresso . . . 8

5 Valutazione Comparativa e Analisi dei Rischi **9**
5.1 Confronto tra Strategie di Ingestion . . . 9
5.2 Matrice dei Rischi e Mitigazioni . . . 10

6 Conclusioni e Raccomandazioni per il Progetto **10**

1 Introduzione

1.1 Scopo del documento

Il presente documento costituisce lo studio tecnologico di dettaglio relativo al Modulo 3: Ingestion e Parsing di Diagrammi Architetturali per il progetto SpecCheck (Capitolato C9, proposto da Zucchetti S.p.A.).

L'obiettivo dell'analisi è definire formati, metodologie e librerie per l'acquisizione (*ingestion*) dei diagrammi UML dei casi d'uso e per la loro conversione in una rappresentazione strutturata, deterministica e riutilizzabile dalle *skill* di OpenCode. Il modulo costituisce lo stadio iniziale dell'intera catena di verifica: la qualità dei dati estratti in questa fase condiziona direttamente l'affidabilità della tracciabilità semantica tra Casi d'Uso e Specifiche (Modulo 4) e, a cascata, della verifica del codice sorgente (Modulo 5).

In particolare, lo studio approfondisce tre ambiti:

- i **formati text-based** per la descrizione di diagrammi UML (Mermaid, XMI) e le relative strategie di parsing;
- l'impiego di *Vision Language Models*$_G$ (VLM) per la comprensione di diagrammi esportati in formato raster o vettoriale (PNG, JPEG);
- la progettazione di una **pipeline di estrazione strutturata** che produca oggetti JSON contenenti attori, azioni e relazioni.

1.2 Riferimenti

1.2.1 Riferimenti normativi

- Capitolato d'appalto C9: *SpecCheck – Sviluppo di Skills per OpenCode per la verifica della tracciabilità end-to-end da casi d'uso a requisiti e codice sorgente* (Zucchetti S.p.A., A.A. 2026/2027);
- *Norme di Progetto*, redatte dal gruppo 0x06tudio.

1.2.2 Riferimenti informativi

- Documentazione e architettura agenti OpenCode: `https://opencode.ai/`;
- **Formati per diagrammi:**
  - *Mermaid*: `https://mermaid.js.org/`;
  - *PlantUML*: `https://plantuml.com/`;
  - *OMG, XML Metadata Interchange (XMI) Specification* e *OMG, Unified Modeling Language (UML) Specification*: `https://www.omg.org/spec/`;
- **Librerie di parsing e validazione:**
  - *lxml*: `https://lxml.de/`;
  - *Lark* (parsing toolkit per Python): `https://github.com/lark-parser/lark`;
  - *Pydantic*: `https://docs.pydantic.dev/`;
- **Piattaforme e modelli linguistici:** *Hugging Face* (raccolta di modelli LLM, SLM e multimodali): `https://huggingface.co/`.

2 Contesto e Sfide nel Capitolato C9

I requisiti obbligatori 1–3 del capitolato C9 prevedono che SpecCheck sia in grado di leggere diagrammi dei casi d'uso (Mermaid, XMI ed eventualmente immagini), documentazione aggiuntiva

e specifiche. Nella riunione del 2026/11/13 il referente aziendale ha chiarito che l'obiettivo principale del progetto è sostituire il prompt in linguaggio naturale con la lettura di un diagramma dei casi d'uso, evitando l'ambiguità del linguaggio naturale, e verificare poi che il prodotto finale soddisfi tali casi d'uso. L'ingestion è quindi il passaggio che trasforma il diagramma in un input affidabile per l'agente.

L'acquisizione dei diagrammi presenta le seguenti criticità:

1. **Eterogeneità dei formati.** I diagrammi possono essere forniti in formati testuali (Mermaid, XMI), vettoriali (SVG) o raster (PNG, JPEG). Ogni formato offre un diverso livello di strutturazione e quindi richiede una strategia di estrazione dedicata, ma l'output deve essere uniforme per tutte le fasi successive.

2. **Variabilità dei dialetti.** Anche all'interno dello stesso standard esistono differenze: i file XMI prodotti da strumenti di modellazione diversi adottano versioni, namespace ed estensioni proprietarie differenti, mentre Mermaid non dispone di un tipo di diagramma dedicato ai casi d'uso e viene quindi utilizzato tramite convenzioni, ad esempio con la sintassi dei *flowchart*.

3. **Perdita di informazione nei formati raster.** Un'immagine non conserva la semantica del modello: attori, frecce e stereotipi (`«include»`, `«extend»`) devono essere ricostruiti per via visiva, con rischio di errori di riconoscimento.

4. **Vincolo sui modelli e determinismo.** Come indicato dal proponente Zucchetti, SpecCheck deve poter operare con modelli locali di dimensioni contenute (**Gemma 4 26B** e **Qwen 3.8 27B**) senza dipendere da API cloud proprietarie. Inoltre il capitolato prescrive che le attività di analisi strutturata non siano affidate al solo ragionamento non-deterministico dell'LLM, ma supportate da applicativi esterni deterministici richiamati tramite *skill* e *tool*.

3 Analisi delle Tecnologie Chiave

3.1 Formati text-based per diagrammi UML

I formati testuali rappresentano la via preferenziale per l'ingestion: il contenuto è già strutturato, è versionabile con Git, può essere analizzato con strumenti deterministici e non richiede inferenza probabilistica.

3.1.1 Mermaid

Mermaid$_G$ è un linguaggio di descrizione di diagrammi basato su testo, supportato nativamente dai principali renderer Markdown (GitHub, GitLab, editor). Poiché le specifiche del progetto sono redatte in Markdown, i diagrammi Mermaid compaiono come blocchi di codice delimitati (```` ```mermaid ````) all'interno dello stesso documento, rendendo naturale l'ingestion congiunta di testo e diagrammi.

- **Punti di forza:** sintassi compatta e leggibile; integrazione diretta con i documenti Markdown; assenza di strumenti di modellazione proprietari.

- **Limiti:** la semantica UML è debole, perché Mermaid descrive principalmente la rappresentazione grafica e non un modello formale; per i casi d'uso si ricorre a convenzioni (nodi con forma di attore, sottografi come confine di sistema, etichette sugli archi per *include* ed *extend*) che il parser deve conoscere ed applicare in modo coerente.

- **Strategie di parsing:** (i) parser basato su grammatica formale (ad esempio con la libreria *Lark*), che garantisce errori di sintassi espliciti; (ii) espressioni regolari per sottoinsiemi ristretti e controllati della sintassi, più rapide da realizzare ma più fragili; (iii) utilizzo del parser ufficiale in JavaScript (richiede un runtime Node.js, da valutare rispetto al vincolo di distribuzione del Modulo 6).

3.1.2 PlantUML

PlantUML$_G$ dispone di una sintassi dedicata ai diagrammi dei casi d'uso (attori, casi d'uso, confine di sistema, relazioni *include*/*extend*/generalizzazione), più vicina alla semantica UML rispetto a Mermaid. Pur non essendo richiesto esplicitamente dal capitolato, il suo supporto è un'estensione a basso costo se il parser è progettato in modo modulare.

3.1.3 XMI (XML Metadata Interchange)

XMI$_G$ è lo standard OMG per la serializzazione di modelli UML in XML e rappresenta il formato di scambio supportato dai principali strumenti di modellazione (ad esempio Papyrus, StarUML, Visual Paradigm).

- **Punti di forza:** contiene il modello UML completo e non solo la sua rappresentazione grafica; ogni elemento è identificato da un identificativo univoco (`xmi:id`) e le relazioni tra elementi sono espresse tramite riferimenti espliciti.
- **Limiti:** verbosità elevata; esistenza di più versioni dello standard e di varianti specifiche di ciascuno strumento; le informazioni grafiche (posizione, dimensione) possono essere separate dal modello o assenti.
- **Strategie di parsing:** lettura con parser XML (*lxml* o `xml.etree`), gestione esplicita dei namespace, selezione degli elementi di interesse tramite XPath (attori, casi d'uso, associazioni, relazioni *include*/*extend*/generalizzazione) e risoluzione dei riferimenti incrociati tramite l'identificativo. Il parser deve essere tollerante rispetto a elementi sconosciuti, che vanno ignorati e segnalati senza interrompere l'estrazione.

3.2 Vision Language Models (VLM)

Quando il diagramma è disponibile solo come immagine (PNG, JPEG), l'unica via di ingestion è la comprensione visiva. I *Vision Language Models*$_G$ sono modelli multimodali in grado di ricevere in ingresso un'immagine e un prompt testuale e di produrre una descrizione o un output strutturato.

- **Immagini vettoriali (SVG).** Un file SVG è in realtà un documento XML: il testo delle etichette è direttamente leggibile e la geometria degli elementi consente di ricostruire connessioni e posizioni senza inferenza visiva. Prima di ricorrere a un VLM, è quindi opportuno verificare se il formato vettoriale consenta un parsing deterministico.
- **Immagini raster.** Il VLM deve riconoscere attori, casi d'uso, confini di sistema, direzione delle frecce e stereotipi. Si tratta del caso più incerto e il rischio principale è quello delle *allucinazioni*: relazioni inventate, frecce invertite, etichette lette in modo errato o elementi omessi.
- **Strategie di mitigazione:** (i) vincolare l'output a uno schema JSON; (ii) fornire immagini ad alta risoluzione o suddivise in porzioni (*tiling*) per le etichette di piccole dimensioni; (iii) associare a ogni elemento estratto un grado di confidenza e un'indicazione

del metodo di estrazione; (iv) eseguire più passaggi indipendenti e confrontarne i risultati (*self-consistency*); (v) segnalare all'utente gli elementi a bassa confidenza affinché li confermi.

- **Disponibilità sui modelli del progetto.** I modelli locali messi a disposizione (Gemma 4 26B, Qwen 3.8 27B) sono in grado di ricevere immagini in ingresso, quindi l'ingestion dei diagrammi raster può essere svolta direttamente in locale. Resta da verificare sperimentalmente durante il PoC la qualità effettiva del riconoscimento sui diagrammi UML; in caso di risultati insufficienti, l'ingestion di immagini può essere delegata ai modelli frontier (soggetti a tetto di spesa) oppure considerata una funzionalità opzionale a basso grado di affidabilità.

3.3 Pipeline di estrazione strutturata

Indipendentemente dal formato di partenza, l'obiettivo è produrre un'unica *rappresentazione intermedia*$_G$ (IR) in formato JSON, indipendente dalla sorgente, che le *skill* dei Moduli 4 e 5 possano consumare senza conoscere i dettagli dell'ingestion. Le entità chiave da estrarre sono:

- **Attori:** soggetti, umani o sistemi esterni, che interagiscono con il sistema;
- **Casi d'uso (azioni):** funzionalità offerte dal sistema;
- **Relazioni:** associazione attore–caso d'uso, *include*, *extend*, generalizzazione;
- **Confine di sistema** (*system boundary*), quando presente;
- **Metadati di provenienza:** file sorgente, posizione nel file, formato originale, metodo di estrazione, confidenza.

I passi della pipeline sono i seguenti:

1. **Rilevamento del formato:** basato su estensione e contenuto (blocco `mermaid` in Markdown, radice XML con namespace XMI, firma di un file immagine).
2. **Selezione del parser:** dispatch verso il parser deterministico (Mermaid, PlantUML, XMI, SVG) oppure verso il percorso VLM per i formati raster.
3. **Normalizzazione:** conversione dell'output nella rappresentazione intermedia, con identificativi stabili e nomi normalizzati (spaziatura, maiuscole, sinonimi del confine di sistema).
4. **Validazione:** controllo dell'output rispetto a uno schema formale (JSON Schema o modelli Pydantic) e verifica di coerenza (ogni relazione deve riferirsi a elementi esistenti, nessun identificativo duplicato).
5. **Persistenza e consegna:** salvataggio dell'artefatto JSON e restituzione all'agente tramite il tool del Sidecar.

```
from pydantic import BaseModel
from typing import Literal, List, Optional

class Provenance(BaseModel):
    source_file: str
    source_format: Literal["mermaid", "plantuml", "xmi", "svg", "raster"
        ]
    extraction_method: Literal["parser", "vlm"]
    confidence: float          # 1.0 per i parser deterministici
```

```python
class Actor(BaseModel):
    id: str
    name: str
    provenance: Provenance

class UseCase(BaseModel):
    id: str
    name: str
    description: Optional[str] = None
    provenance: Provenance

class Relation(BaseModel):
    source_id: str
    target_id: str
    kind: Literal["association", "include", "extend", "generalization"]
    provenance: Provenance

class DiagramIR(BaseModel):
    system_name: Optional[str] = None
    actors: List[Actor]
    use_cases: List[UseCase]
    relations: List[Relation]
```

Listing 1: Schema concettuale della rappresentazione intermedia dei diagrammi.

3.4 Collegamento con gli altri moduli

Il modulo di ingestion non produce un risultato finale, ma un artefatto intermedio. Nell'architettura di integrazione del Modulo 6 la funzionalità è esposta da un tool deterministico del Sidecar (`speccheck_parse_artifacts`), invocato dalla skill di OpenCode, che restituisce l'IR in formato JSON. Tale artefatto costituisce l'ingresso della pipeline di tracciabilità del Modulo 4 (confronto Casi d'Uso – Specifiche) ed evita di caricare i diagrammi grezzi nel contesto dell'LLM, in linea con le raccomandazioni di gestione del contesto (Modulo 1).

4 Architettura Proposta e Proof of Concept (PoC)

In conformità con la separazione tra ragionamento probabilistico dell'LLM ed esecuzione deterministica richiesta dal capitolato, l'ingestion è implementata come componente del Sidecar, con i soli passi non deterministici (VLM) isolati e chiaramente marcati nell'output.

Il referente aziendale ha inoltre precisato la differenza tra *skill* e *tool*: la skill è un file `.md` in linguaggio naturale che entra nel contesto dell'LLM, che la interpreta liberamente (risultato non deterministico), e di cui viene normalmente iniettata solo una breve descrizione; il tool è invece uno strumento descritto in modo strutturato, con parametri definiti, la cui esecuzione è deterministica, anche se è l'LLM a decidere quando e con quali parametri invocarlo. Per questo motivo il parsing dei diagrammi è collocato in un tool del Sidecar e non nelle istruzioni di una skill, mentre la skill si limita a descrivere quando e come invocarlo.

4.1 Flusso operativo del PoC

1. L'agente OpenCode riceve il comando di analisi e individua i file dei diagrammi indicati dall'utente (Markdown con blocchi Mermaid, file XMI, immagini);
2. la skill invoca il tool `speccheck_parse_artifacts` del Sidecar;

3. il Sidecar rileva il formato di ciascun file e seleziona il parser corrispondente;
4. per i formati testuali il parsing è interamente deterministico e produce la rappresentazione intermedia con confidenza massima;
5. per le immagini raster il Sidecar invia l'immagine al VLM con un prompt strutturato e con output vincolato allo schema JSON; gli elementi estratti sono marcati con il metodo `vlm` e con la relativa confidenza;
6. l'output viene validato, persistito e restituito all'agente sotto forma di JSON compatto, pronto per le fasi di tracciabilità.

4.2 Esempio di diagramma in ingresso

```
flowchart LR
    Utente([Utente])
    Admin([Amministratore])
    subgraph Sistema
        UC1(UC-01 Effettua login)
        UC2(UC-02 Gestisci utenti)
    end
    Utente --> UC1
    Admin --> UC2
    UC2 -.->|include| UC1
```

Listing 2: Esempio di caso d'uso espresso in Mermaid tramite convenzioni su flowchart.

```
{
  "system_name": "Sistema",
  "actors": [
    {"id": "Utente", "name": "Utente"},
    {"id": "Admin", "name": "Amministratore"}
  ],
  "use_cases": [
    {"id": "UC1", "name": "UC-01␣Effettua␣login"},
    {"id": "UC2", "name": "UC-02␣Gestisci␣utenti"}
  ],
  "relations": [
    {"source_id": "Utente", "target_id": "UC1", "kind": "association"},
    {"source_id": "Admin",  "target_id": "UC2", "kind": "association"},
    {"source_id": "UC2",    "target_id": "UC1", "kind": "include"}
  ]
}
```

Listing 3: Estratto della rappresentazione intermedia ottenuta dal diagramma precedente (JSON).

5 Valutazione Comparativa e Analisi dei Rischi

5.1 Confronto tra Strategie di Ingestion

Criterio	**Parser Deterministico (Mermaid / XMI)**	**Solo VLM (su immagini)**	**Pipeline Ibrida (Parser + VLM come fallback)**
Accuratezza	Alta (estrazione esatta se il formato è valido)	Media-Bassa (soggetta ad allucinazioni)	Alta sui formati testuali, media sulle immagini
Determinismo e riproducibilità	Totale	Bassa	Alta (la parte non deterministica è isolata e marcata)
Copertura dei formati	Limitata ai formati supportati	Ampia (qualsiasi immagine)	Ampia
Costo e risorse	Trascurabile	Alto (inferenza multimodale)	Basso (VLM invocato solo quando necessario)
Spiegabilità	Alta (ogni elemento è riconducibile al sorgente)	Bassa	Alta (provenienza e confidenza per ogni elemento)
Complessità implementativa	Media (un parser per formato)	Bassa	Media-Alta

5.2 Matrice dei Rischi e Mitigazioni

N.	Rischio	Impatto	Probabilità	Strategia di Mitigazione
1	Allucinazioni del VLM su diagrammi raster (relazioni inventate o frecce invertite)	Alto	Media	Output vincolato a schema JSON, campo di confidenza per ogni elemento, conferma esplicita dell'utente sugli elementi a bassa confidenza; preferire sempre il formato testuale quando disponibile.
2	Varianti e dialetti di XMI prodotti da strumenti diversi	Medio	Alta	Parser tollerante con gestione dei namespace, ignorando e segnalando gli elementi sconosciuti; test su file reali esportati da più strumenti.
3	Uso non uniforme delle convenzioni Mermaid per i casi d'uso	Medio	Media	Definire e documentare nelle Norme di Progetto una convenzione unica per attori, confine di sistema e relazioni; segnalare le costruzioni non riconosciute.
4	Qualità insufficiente del riconoscimento visivo dei modelli locali sui diagrammi UML	Medio	Media	Verifica sperimentale della qualità nel PoC; in alternativa, delegare ai modelli frontier o considerare il percorso raster come funzionalità opzionale.
5	Diagrammi di grandi dimensioni che saturano il contesto	Medio	Bassa	Restituire all'agente solo la rappresentazione intermedia compatta e non il sorgente del diagramma.

6 Conclusioni e Raccomandazioni per il Progetto

Lo studio condotto evidenzia che l'acquisizione dei diagrammi deve privilegiare, ovunque possibile, il parsing deterministico dei formati testuali, riservando l'inferenza visiva ai soli casi in cui il diagramma sia disponibile esclusivamente come immagine.

Si raccomanda al gruppo 0x06tudio di adottare le seguenti linee guida per il Proof of Concept (PoC) e per il Minimum Viable Product (MVP):

1. **Priorità ai formati testuali:** implementare per primi i parser per Mermaid (blocchi in Markdown) e XMI, che coprono i requisiti obbligatori del capitolato in modo deterministico e riproducibile;
2. **Rappresentazione intermedia unica:** adottare uno schema JSON condiviso (attori, casi d'uso, relazioni, provenienza) validato formalmente, da concordare con i Moduli 4 e 6 per garantire l'interoperabilità;

3. **VLM come fallback marcato:** trattare l'ingestion di immagini come percorso a minore affidabilità, con confidenza esplicita per ogni elemento e conferma dell'utente sugli elementi incerti;
4. **Architettura estendibile:** strutturare il Sidecar con un parser per formato dietro un'interfaccia comune, così da poter aggiungere in seguito PlantUML, SVG o altri formati senza modificare le fasi successive;
5. **Convenzioni di progetto:** definire nelle Norme di Progetto le convenzioni per rappresentare i casi d'uso in Mermaid, riducendo l'ambiguità a monte.