Università Campus Bio-Medico di Roma

Facoltà di Ingegneria

Corso di Laurea in Ingegneria dei Sistemi Intelligenti

Progetto per l'insegnamento di

Intelligent Architectures: Microservices Programming for AI

Domain Interviewer & Modeler

Agente 1 di una pipeline multi-agente per la scoperta del dominio
e la progettazione automatica di architetture a microservizi
tramite AI locale e workflow orchestrati

Autore:
Luca Scorza

Docente:
Prof. Floriano Caprio

Anno Accademico 2025/2026

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Executive Summary

1.1 Visione Strategica

Il Domain Interviewer & Modeler (DIM) è il primo agente di una pipeline multi-agente dedicata alla progettazione automatica di architetture a microservizi secondo i principi del Domain-Driven Design (DDD). Sviluppato come Agente 1 del sistema *Intelligent Domain Architect Agents*, il DIM trasforma una conversazione naturale con uno stakeholder di business in un documento architetturale strutturato e in un modello JSON formale del dominio, pronti per essere consumati dagli agenti downstream.

1.2 Value Proposition

A differenza di un semplice chatbot o di un questionario statico, il DIM risolve il problema della raccolta e strutturazione dei requisiti di dominio attraverso tre pilastri fondamentali:

- **Intervista Intelligente Multi-Fase:** Una catena di cinque agenti specializzati (Domain, Context, Event, Pattern, Resilience Interviewer) guida lo stakeholder attraverso le cinque fasi del Discovery DDD, ponendo una domanda alla volta e approfondendo le risposte vaghe.
- **Generazione Automatica del Modello:** Al termine dell'intervista, un agente DDD Analyst sintetizza l'intera conversazione in un documento architetturale Markdown strutturato secondo cinque pilastri (Strategic Analysis, Boundaries, EDA Integration, Tactical Patterns, Technical Excellence), che viene poi convertito in JSON formale da un agente JSON Coder.
- **Raffinamento Interattivo:** L'utente può rivedere, modificare e confermare il modello generato attraverso un flusso di Refinement conversazionale assistito da LLM, con meccanismo di Draft/Confirm/Discard.

1.3 Sintesi Tecnica

L'architettura è interamente basata su microservizi containerizzati orchestrati da Docker Compose. L'orchestrazione dei workflow di intervista, analisi e raffinamento è delegata a n8n, che coordina gli agenti LLM tramite Ollama (llama3) e gestisce lo stato conversazionale tramite Redis. L'osservabilità completa è garantita dallo stack OpenTelemetry Collector / Loki / Tempo / Prometheus / Grafana, che raccoglie log, tracce distribuite e metriche da tutti i container del sistema, incluso n8n stesso.

Capitolo 2

Analisi dello Scenario e Business Case

2.1 Razionale del Progetto

In un contesto aziendale reale, la definizione del dominio applicativo e la sua suddivisione in sottodomini funzionali rappresenta il primo passo critico nella progettazione di un sistema basato su microservizi. Questo processo, tipicamente condotto manualmente da architetti esperti attraverso sessioni di Event Storming e interviste con gli stakeholder, è lungo, soggetto a bias individuali e difficilmente ripetibile in modo sistematico.

Il Domain Interviewer & Modeler nasce per automatizzare e strutturare questa fase fondamentale: è l'agente che conduce l'intervista di scoperta del dominio, analizza le risposte e produce un modello architetturale formale secondo i principi del Domain-Driven Design.

2.2 Analisi dei Pain Points

Lo scenario tipico di una fase di discovery manuale presenta criticità strutturali:

- **Raccolta non strutturata:** Le interviste con gli stakeholder producono note informali, appunti sparsi e conoscenza implicita che non viene formalizzata in un modello coerente.
- **Assenza di copertura sistematica:** Senza una guida strutturata, l'architetto rischia di trascurare aspetti critici come i pattern di gestione dei fallimenti (Saga), i modelli di consistenza o le esigenze di integrazione con sistemi legacy.
- **Dipendenza dall'esperienza individuale:** La qualità del modello di dominio dipende interamente dall'esperienza dell'architetto che conduce la sessione.
- **Non ripetibilità:** Due sessioni di discovery sullo stesso progetto possono produrre modelli significativamente diversi.

2.3 Rischi e Impatto Operativo

Un modello di dominio incompleto o mal definito si traduce in bounded context errati, microservizi con responsabilità sovrapposte, e un'architettura event-driven con eventi mancanti o ridondanti. Il DIM mitiga questi rischi attraverso la sistematicità della pipeline a cinque fasi, la persistenza dello stato conversazionale in Redis, e la generazione automatica di un modello JSON strutturato e revisionabile.

Capitolo 3

Obiettivi e Requisiti del Sistema

3.1 Obiettivi Strategici

Il sistema opera come agente autonomo di Domain Discovery nella pipeline di progettazione architetturale. Gli obiettivi si articolano in:

- **Intervista Guidata Multi-Fase:** Conduzione automatica di un'intervista DDD strutturata in cinque fasi progressive, ciascuna gestita da un agente specializzato con competenze specifiche nel proprio pillar architetturale.
- **Generazione del Modello di Dominio:** Produzione automatica di un documento architetturale Markdown e della sua rappresentazione JSON strutturata, a partire dallo storico conversazionale completo.
- **Raffinamento Interattivo:** Supporto a un ciclo di revisione in cui lo stakeholder può modificare il modello generato attraverso conversazione naturale con un agente dedicato (JSON Patcher).
- **Osservabilità Completa:** Monitoraggio end-to-end di log, tracce distribuite e metriche per tutti i componenti del sistema.

3.2 Requisiti Funzionali

1. Pipeline di Intervista a 5 Fasi: Cinque agenti LLM specializzati (Domain, Context, Event, Pattern, Resilience Interviewer) che conducono sequenzialmente l'intervista, con transizione automatica basata su tag di completamento fase (`[FASE_N_COMPLETA]`).

2. Routing Intelligente: Un workflow di routing che determina automaticamente se instradare il messaggio dell'utente verso il Discovery Workflow (prima intervista) o verso il Refinement Workflow (modello già generato), basandosi sullo stato in Redis.

3. Generazione Architetturale: Due agenti in cascata — DDD Analyst (produce il documento Markdown) e JSON Coder (lo converte in JSON strutturato) — che processano l'intera conversazione al completamento della Fase 5.

4. Raffinamento del Modello: Un workflow dedicato con agente JSON Patcher che applica le modifiche richieste dall'utente al JSON architetturale esistente, e un meccanismo di Draft Decision (conferma/rifiuta) per gestire le bozze.

5. Frontend Chat: Interfaccia web HTML con comunicazione REST verso n8n, supporto a sessioni persistenti tramite `sessionId` e rendering differenziato per messaggi testuali e blocchi JSON architetturali.

6. Stato Conversazionale Persistente: Gestione dello stato tramite Redis con chiavi dedicate per fase di discovery (`discovery_phase`), JSON architetturale (`discovery_json`), bozze (`draft_json`), storico chat (`chat_history`) e stato delle bozze (`draft_status`).

3.3 Requisiti Non Funzionali

- **Osservabilità:** Raccolta centralizzata di log (Loki), tracce distribuite (Tempo) e metriche (Prometheus) tramite OpenTelemetry Collector, con dashboard Grafana preconfigurate.
- **Privacy e Indipendenza:** Inferenza LLM completamente locale tramite Ollama. Nessun dato inviato a servizi cloud esterni.
- **Portabilità:** Containerizzazione completa tramite Docker Compose per ambiente self-contained e riproducibile.
- **Modularità:** Ogni workflow n8n è un componente autonomo richiamabile indipendentemente, con interfacce ben definite (input/output via webhook e Redis).

Capitolo 4

Stakeholder e Ruoli

Il presente capitolo identifica gli attori, umani e sistemici, che interagiscono con il Domain Interviewer & Modeler. Per ogni stakeholder vengono definiti il perimetro di responsabilità e le modalità di interazione con il sistema.

4.1 Business Stakeholder (Utente Intervistato)

Ruolo: Rappresentante del business che partecipa all'intervista di scoperta del dominio. È la fonte primaria delle informazioni sul contesto aziendale, i processi, le entità e le regole di business.

Responsabilità: Rispondere alle domande degli agenti intervistatori, fornire dettagli sui processi aziendali, validare e confermare (o rifiutare) il modello architetturale generato.

Interfaccia: Chat web (`frontend/index.html`) su browser, comunicazione asincrona tramite webhook n8n.

4.2 Agenti Intervistatori (Discovery Pipeline)

Il sistema impiega cinque agenti specializzati, ciascuno responsabile di una fase distinta dell'intervista DDD:

Agente	Fase	Focus
Domain Interviewer	1	Strategic Analysis: Core, Generic e Supporting Subdomains
Context Interviewer	2	Boundaries e Language: Bounded Contexts, Aggregates, Ubiquitous Language
Event Interviewer	3	EDA Integration: Actors, Commands, Domain Events, Pub/Sub
Pattern Interviewer	4	Coordination Styles: Choreography/Orchestration, Sagas, CQRS, Event Sourcing
Resilience Interviewer	5	Technical Excellence: Cloud Scalability, ACL, Monitoring, Security

Tabella 4.1: Agenti Intervistatori e relative fasi

Dettaglio Tecnico: Meccanismo di Transizione tra Fasi

Ogni agente possiede un tag di completamento fase (es. `[FASE_1_COMPLETA]`). Quando l'agente ritiene che la propria fase sia sufficientemente dettagliata, emette il tag nell'output. Un nodo Switch nel workflow rileva il tag e aggiorna la chiave Redis `discovery_phase` al valore successivo, causando l'instradamento dei messaggi futuri verso l'agente della fase successiva. Prima di emettere il tag, l'agente è istruito a riassumere i risultati della propria fase e a porre la prima domanda della fase successiva, garantendo una transizione fluida.

4.3 Agenti di Elaborazione (Post-Discovery)

DDD Analyst: Agente che analizza l'intero storico conversazionale e produce il documento architetturale Markdown strutturato secondo i cinque pilastri DDD.

JSON Coder: Agente che converte il documento Markdown in un JSON formale con schema predefinito, preservando ogni dettaglio tecnico senza riassumere.

JSON Patcher: Agente del Refinement Workflow che applica le modifiche richieste dall'utente al JSON architetturale esistente, producendo un JSON aggiornato valido.

4.4 Stack di Osservabilità

Ruolo: Insieme di strumenti tecnici preposti al monitoraggio e alla telemetria del sistema.

Responsabilità: OpenTelemetry Collector raccoglie log da tutti i container Docker via `receiver_creator` con `docker_observer`, riceve telemetria OTLP da n8n, e fa scraping delle metriche Redis. I segnali vengono esportati verso Loki (log), Tempo (tracce) e Prometheus (metriche). Grafana visualizza i tre segnali con datasource preconfigurati e correlazione automatica log-tracce.

4.5 Matrice di Tracciabilità Ruoli/Funzioni

Ruolo/ Stakeholder	Discovery	Generazione	Refinement	Draft Mgmt	Osservabilità
Stakeholder	Risponde	No	Modifica	Conferma/ Rifiuta	No
Interviewer Agents	Intervista	No	No	No	No
DDD Analyst	No	Genera MD	No	No	No
JSON Coder	No	Genera JSON	No	No	No
JSON Patcher	No	No	Patcha JSON	No	No
Team DevOps	Sì	Sì	Sì	Sì	Sì
Stack Osservabilità	No	No	No	No	Sì

Tabella 4.2: Matrice di tracciabilità Ruoli/Funzioni

Capitolo 5

Architettura del Sistema

Questo capitolo descrive l'architettura logica e fisica del Domain Interviewer & Modeler, illustrando come i moduli cooperano per garantire un servizio di domain discovery scalabile, modulare e osservabile.

Figura 5.1: Architettura a microservizi del sistema Domain Interviewer & Modeler

5.1 Visione d'Insieme

Il DIM è un ecosistema modulare basato su microservizi containerizzati in cui ogni componente ha un compito preciso e delimitato:

- **L'Orchestratore (n8n):** Motore di workflow automation che coordina l'intera pipeline di intervista, analisi e raffinamento. Gestisce i webhook HTTP, il routing dei messaggi e l'invocazione degli agenti LLM.
- **Lo Stato Conversazionale (Redis):** Key-value store che mantiene lo stato persistente delle sessioni: fase corrente dell'intervista, storico chat, JSON architetturali generati, bozze in attesa di conferma.
- **Il Cervello (Ollama):** Runtime LLM locale che esegue il modello `llama3:latest` per tutte le operazioni di ragionamento semantico: intervista, analisi, generazione e patching del modello.
- **L'Interfaccia (Frontend HTML):** Chat web full-screen che consente allo stakeholder di interagire con gli agenti in linguaggio naturale, con rendering differenziato per testo e JSON architetturale.
- **Il Supervisore (Stack Osservabilità):** Pipeline completa OpenTelemetry Collector → Loki/Tempo/Prometheus → Grafana che raccoglie e correla log, tracce distribuite e metriche da tutti i container.

5.2 Stack Tecnologico

Componente	Tecnologia	Versione	Porta	Ruolo
n8n	n8n Workflow Engine	latest	5678	Orchestrazione workflow
redis	Redis Alpine	alpine	6379	Stato conversazionale
redis-exporter	Redis Exporter	latest	9121	Metriche Redis
ollama	Ollama	latest	11434	LLM locale
otel-collector	OTel Collector Contrib	0.133.0	4317/4318	Hub telemetria
loki	Grafana Loki	3.5.0	3100	Aggregazione log
tempo	Grafana Tempo	2.9.0	3200	Tracce distribuite
prometheus	Prometheus	2.53.0	9090	Metriche
grafana	Grafana	12.2	8080	Dashboard

Tabella 5.1: Stack tecnologico del sistema

5.3 I Pilastri Progettuali

Per garantire un sistema affidabile e manutenibile, sono stati seguiti quattro principi architetturali fondamentali:

Orchestrazione via Workflow (n8n): Tutta la logica applicativa è espressa come workflow n8n, non come codice custom. Questo approccio garantisce visibilità totale sul flusso di esecuzione, facilità di debug e modifica senza ricompilazione, e natività nell'integrazione con servizi esterni.

State Management Centralizzato (Redis): Ogni informazione di stato — fase di discovery, storico chat, JSON generati, bozze — è persistita in Redis con chiavi strutturate per `sessionId`. Questo disaccoppia completamente lo stato dalla logica di esecuzione e abilita la ripresa di sessioni interrotte.

Agenti Specializzati con System Prompt: Ogni agente LLM ha un system prompt rigorosamente definito che ne delimita il ruolo, le regole di output, il focus tematico e le condizioni di completamento fase. Questa separazione garantisce che ogni agente sia un esperto focalizzato sul proprio pillar architetturale.

Osservabilità Nativa: L'osservabilità non è un add-on ma un componente strutturale dell'architettura, con OTel Collector che raccoglie automaticamente i log di tutti i container Docker e la telemetria OTLP di n8n.

5.4 Perché queste tecnologie?

Dettaglio Tecnico: n8n, Redis e Ollama

n8n: Scelto come motore di orchestrazione per la capacità di definire workflow complessi con nodi condizionali (Switch), invocazione di sub-workflow (`executeWorkflow`), integrazione nativa con LLM (nodi `lmChatOllama` e `agent`), e gestione della memoria conversazionale tramite nodi `memoryRedisChat`. La UI di n8n offre inoltre visibilità completa sull'esecuzione di ogni workflow.

Redis: Scelto per la gestione dello stato conversazionale grazie alla bassa latenza, al supporto TTL nativo per la scadenza automatica delle sessioni (24h), e all'integrazione nativa con i nodi n8n per operazioni GET/SET/DELETE.

Ollama (llama3:latest): Runtime LLM locale che garantisce inferenza privata senza dipendenza da servizi cloud. Il modello `llama3` è utilizzato con temperatura variabile per ruolo (0.4 per gli intervistatori, 0.3 per il DDD Analyst, 0.0 per il JSON Coder) e context window di 8192 token.

Capitolo 6

Deployment e Containerizzazione Docker

L'intero sistema è definito in un singolo file `docker-compose.yml` che descrive tutti i microservizi e le relative dipendenze. Tutti i servizi comunicano tramite una rete Docker interna denominata `rete_unica`.

6.1 Servizi Definiti nel Docker Compose

Container	Immagine	Porte	Volumi Principali
loki	grafana/loki:3.5.0	3100	`./loki/loki-config.yml`
tempo	grafana/tempo:2.9.0	3200	`./tempo/tempo-config.yml`
otel-collector	otel/opentelemetry-collector-contrib:0.133.0	4317, 4318	Docker socket, config YAML
prometheus	prom/prometheus:v2.53.0	9090	`./prometheus/prometheus.yml`
grafana	grafana/grafana: 12.2-ubuntu	8080	Provisioning datasources
n8n	docker.n8n.io/n8nio/n8n	5678	`./Data_N8N`, workflows
redis	redis:alpine	6379	`./Data_Redis`
redis-exporter	oliver006/redis_exporter	9121	—
ollama	ollama/ollama:latest	11434	`./Data_Ollama`

Tabella 6.1: Dettaglio dei servizi Docker Compose

6.2 Rete e Dipendenze

Tutti i container sono connessi alla rete bridge `rete_unica`. Le dipendenze chiave sono:

- **n8n** dipende da `redis` e `ollama` per operare correttamente.
- **otel-collector** accede al Docker socket (`/var/run/docker.sock`) per la discovery automatica dei container e la raccolta dei log tramite `receiver_creator`.
- **redis-exporter** si connette a `redis:6379` per esporre le metriche in formato Prometheus.
- **grafana** è preconfigurato con datasource per Loki, Tempo e Prometheus tramite il provisioning automatico in `grafana/provisioning/datasources/datasources.yml`.

6.3 Variabili d'Ambiente n8n

n8n è configurato con variabili d'ambiente specifiche per l'osservabilità OTLP:

```
N8N_DIAGNOSTICS_ENABLED=true
N8N_OTEL_SDK_DISABLED=false
OTEL_EXPORTER_OTLP_ENDPOINT=http://otel-collector:4318
OTEL_EXPORTER_OTLP_PROTOCOL=http/protobuf
```

```
N8N_OTEL_SPAN_SAMPLING_RATE=1
N8N_LOG_OUTPUT=console
```

Dettaglio Tecnico: Telemetria OTLP di n8n

n8n supporta nativamente l'esportazione di tracce e log via protocollo OTLP verso un OpenTelemetry Collector. La configurazione `N8N_OTEL_SPAN_SAMPLING_RATE=1` garantisce il campionamento del 100% delle tracce, essenziale in ambiente di development per avere visibilità completa su ogni esecuzione di workflow.

Capitolo 7

Pipeline dei Workflow n8n

Il cuore logico del DIM è costituito da cinque workflow n8n interconnessi che implementano l'intera pipeline di discovery, analisi e raffinamento. Ogni workflow è un componente autonomo con interfacce well-defined.

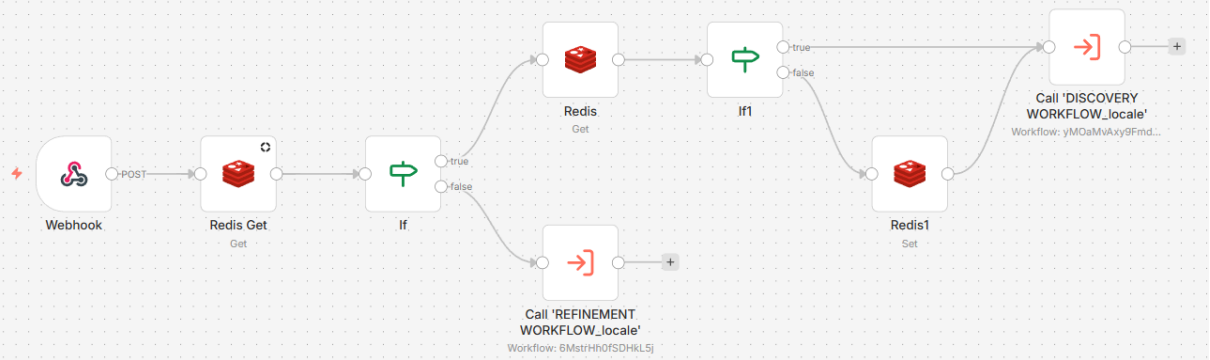

Figura 7.1: Routing Workflow e Pipeline di Gestione dei Messaggi

7.1 Routing Workflow

Il Routing Workflow è il punto di ingresso dell'intera pipeline. Riceve i messaggi dallo stakeholder tramite webhook HTTP e decide dove instradadarli.

Logica di Routing:

1. Riceve il messaggio con `sessionId` dal frontend via webhook POST.
2. Controlla in Redis la chiave `discovery_json:{sessionId}`:
   - **Se NON esiste:** La discovery non è ancora completata → inoltra al Discovery Workflow.
   - **Se esiste:** Il modello architetturale è già stato generato → inoltra al Refinement Workflow.
3. Restituisce la risposta dell'agente al frontend tramite webhook response.

7.2 Discovery Workflow

Il Discovery Workflow è il componente più complesso dell'intero sistema. Implementa la pipeline di intervista a cinque fasi con agenti specializzati.

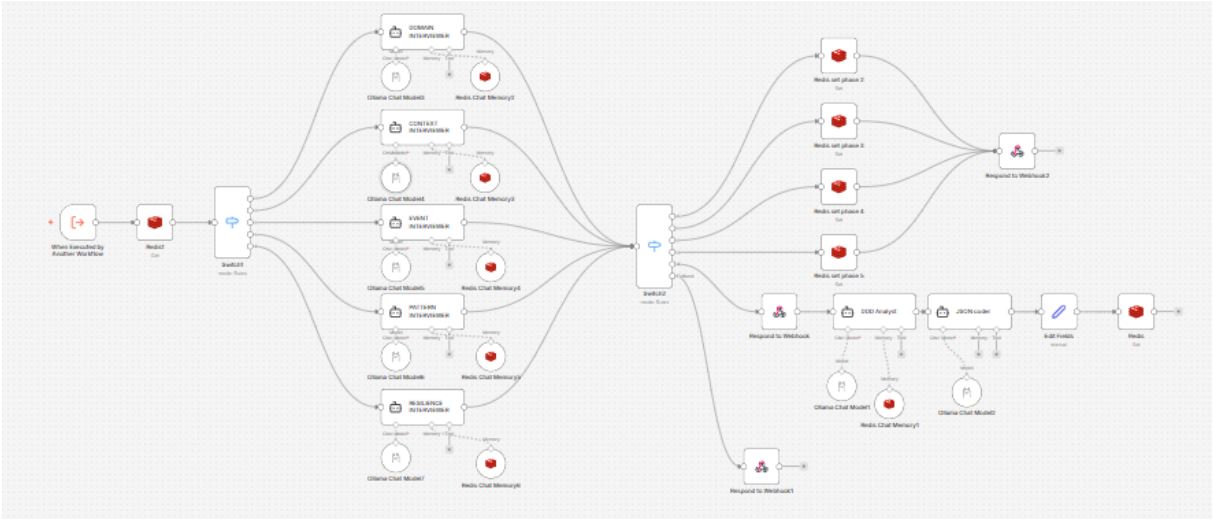

Figura 7.2: Discovery Workflow: Pipeline di intervista a 5 fasi

Flusso di Esecuzione:

1. Riceve `message` e `sessionId` dal Routing Workflow (invocazione sub-workflow).
2. Legge la fase corrente da Redis (`discovery_phase:{sessionId}`).
3. Un nodo Switch instrada verso l'agente della fase corrente (1→5).
4. Ogni agente LLM utilizza:
   - **Ollama Chat Model** con `llama3:latest` (temperatura 0.4, context 8192 token).
   - **Redis Chat Memory** con context window crescente per fase (10→20→30→40→50 messaggi) per gestire la crescente complessità della conversazione.
5. Un secondo nodo Switch analizza l'output per i tag `[FASE_N_COMPLETA]`:
   - Se il tag è presente: aggiorna la fase in Redis e restituisce la risposta.
   - Se il tag non è presente: restituisce la risposta direttamente (la fase non cambia).
6. Al completamento della Fase 5 (`[FASE_5_COMPLETA]`): attiva la catena DDD Analyst → JSON Coder → salvataggio in Redis.

Agente	Fase	Temp.	Memory Window	Tag Completamento
Domain Interviewer	1	0.4	10 msg	`[FASE_1_COMPLETA]`
Context Interviewer	2	0.4	20 msg	`[FASE_2_COMPLETA]`
Event Interviewer	3	0.4	30 msg	`[FASE_3_COMPLETA]`
Pattern Interviewer	4	0.4	40 msg	`[FASE_4_COMPLETA]`
Resilience Interviewer	5	0.4	50 msg	`[FASE_5_COMPLETA]`
DDD Analyst	Post	0.3	200 msg	—
JSON Coder	Post	0.0	—	—

Tabella 7.1: Configurazione degli agenti nel Discovery Workflow

7.3 Refinement Workflow

Una volta generato il modello architetturale, lo stakeholder può richiedere modifiche conversazionali al JSON tramite il Refinement Workflow.

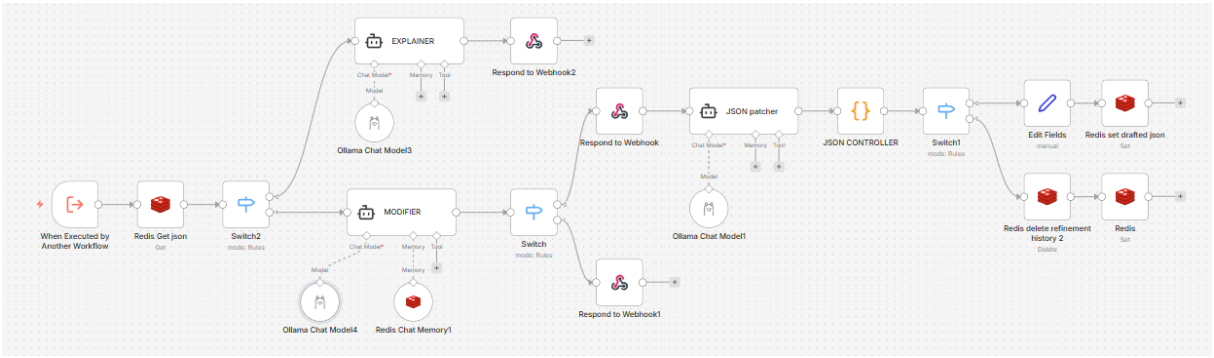

Figura 7.3: Refinement Workflow — Modifica interattiva del modello architetturale

Flusso di Esecuzione:

1. Riceve la richiesta di modifica dall'utente e carica il JSON architetturale corrente da Redis.
2. Un agente JSON Patcher (con system prompt dedicato) analizza la richiesta e produce un JSON aggiornato.
3. Il JSON modificato viene salvato come "bozza" in Redis (`draft_json:{sessionId}`) e lo stato bozza viene impostato a `pending` (`draft_status:{sessionId}`).
4. L'utente visualizza le differenze e può confermare o rifiutare la modifica.

7.4 Polling Workflow

Il Polling Workflow implementa un meccanismo di polling lato server per verificare lo stato delle operazioni asincrone, in particolare la completezza della generazione del modello architetturale dopo la Fase 5.

Logica: Il frontend invia richieste periodiche al webhook di polling per verificare se il JSON architetturale è stato generato e salvato in Redis dopo il completamento della pipeline Discovery.

7.5 Draft Decision Workflow

Il Draft Decision Workflow gestisce il ciclo di vita delle bozze di modifica al modello architetturale.

Azioni supportate:

- **Conferma (confirm):** Promuove la bozza (`draft_json`) a versione corrente (`discovery_json`) e rimuove lo stato bozza.
- **Rifiuto (discard):** Elimina la bozza e il suo stato, mantenendo la versione corrente invariata.

Dettaglio Tecnico: Gestione delle Chiavi Redis

Lo stato conversazionale del DIM è gestito tramite un insieme strutturato di chiavi Redis, tutte con namespace per `sessionId`:

`discovery_phase:{sessionId}`	Fase corrente (1–5)
`chat_history:{sessionId}`	Storico messaggi (gestito da n8n memoryRedisChat)
`discovery_json:{sessionId}`	Modello architetturale corrente (TTL 24h)
`draft_json:{sessionId}`	Bozza di modifica (TTL 24h)
`draft_status:{sessionId}`	Stato bozza: `pending` / vuoto

Capitolo 8

Frontend Chat

Il frontend è un'interfaccia web single-page (`frontend/index.html`) che consente allo stakeholder di interagire con il DIM attraverso una chat full-screen.

8.1 Caratteristiche Principali

- **Chat Full-Screen:** Interfaccia a schermo intero con layout responsive, barra di input fissa in basso e area messaggi scrollabile.
- **Sessioni Persistenti:** Ogni sessione è identificata da un `sessionId` univoco (UUID v4) generato lato client e utilizzato per tutte le interazioni con i webhook n8n.
- **Rendering Differenziato:** I messaggi dell'agente vengono analizzati per individuare blocchi JSON architetturali. Quando rilevati, il JSON viene formattato e presentato in un blocco collassabile con sintassi evidenziata, separato dal testo conversazionale.
- **Meccanismo di Polling:** Dopo il completamento della Fase 5 (rilevato dal tag `[FASE_5_COMPLETA]` nella risposta), il frontend avvia un polling periodico verso il Polling Workflow per attendere la generazione del modello architetturale.
- **Draft Management UI:** Quando l'utente riceve una bozza di modifica (durante il Refinement), vengono visualizzati pulsanti "Conferma" e "Rifiuta" che invocano il Draft Decision Workflow.
- **Indicatore di Fase:** Un indicatore visivo nella parte superiore della chat mostra la fase corrente dell'intervista (1–5 o "Modello Generato").

8.2 Comunicazione con il Backend

La comunicazione avviene tramite chiamate `fetch()` (REST) ai webhook n8n esposti sulla porta 5678:

Azione	Metodo	Endpoint	Payload
Invio messaggio	POST	`/webhook/chat`	`message, sessionId`
Polling modello	POST	`/webhook/poll`	`sessionId`
Draft Decision	POST	`/webhook/draft-decision`	`sessionId, action`

Tabella 8.1: Endpoint di comunicazione Frontend → n8n

Capitolo 9

Stack di Osservabilità

L'osservabilità è un pilastro architetturale del DIM, non un componente aggiuntivo. Il sistema implementa i tre pilastri dell'osservabilità moderna — Log, Tracce e Metriche — attraverso una pipeline centralizzata basata su OpenTelemetry Collector.

Figura 9.1: Architettura dello stack di osservabilità: OTel Collector → Loki / Tempo / Prometheus → Grafana

9.1 OpenTelemetry Collector: L'Hub Centrale

L'OpenTelemetry Collector (versione Contrib 0.133.0) è il componente centrale della pipeline di osservabilità. Raccoglie, processa e instrada tutti i segnali di telemetria verso i backend appropriati.

9.1.1 Receivers (Sorgenti Dati)

Il Collector è configurato con tre tipologie di receiver:

Receiver	Tipo Segnale	Funzione
OTLP (HTTP/gRPC)	Log + Tracce	Riceve telemetria strutturata da n8n via protocollo OTLP
receiver_creator con filelog	Log	Scopre automaticamente i container Docker tramite `docker_observer` e raccoglie i log di ogni container
prometheus (scrape)	Metriche	Fa scraping delle metriche di `redis-exporter:9121/metrics`

Tabella 9.1: Receivers configurati nell'OpenTelemetry Collector

9.1.2 Processors (Elaborazione)

- **batch:** Raggruppa i segnali in batch per ottimizzare le operazioni di rete verso i backend.
- **resource:** Arricchisce i dati con attributi aggiuntivi, in particolare aggiunge l'attributo `service.name` derivato dal nome del container Docker (`container.name`) per identificare univocamente la sorgente di ogni segnale in Grafana.

9.1.3 Exporters (Destinazioni)

Exporter	Backend	Configurazione
loki	Loki (3100)	Esporta log con attributi di risorsa come label Loki
otlphttp/tempo	Tempo (4318)	Esporta tracce distribuite via OTLP HTTP
prometheus remotewrite	Prometheus (9090)	Esporta metriche via Remote Write

Tabella 9.2: Exporters configurati nell'OpenTelemetry Collector

Dettaglio Tecnico: receiver_creator e docker_observer

L'extension `docker_observer` si connette al Docker socket (`unix:///var/run/docker.sock`) e monitora lo stato dei container in tempo reale. Per ogni container scoperto, il `receiver_creator` istanzia dinamicamente un receiver `filelog` che legge i log del container dal filesystem dell'host. Questo approccio garantisce la raccolta automatica dei log di tutti i container senza necessità di configurazione manuale per ogni nuovo servizio aggiunto allo stack.

La configurazione utilizza un operatore `container` nel parser filelog che estrae automaticamente metadati come `container_id`, `container_name` e il formato del log (JSON, CRI, Docker).

9.2 Loki: Aggregazione Log

Grafana Loki (v3.5.0) è il backend di log aggregation. Configurato in modalità single-instance per sviluppo locale, riceve i log dall'OTel Collector e li rende interrogabili tramite LogQL in Grafana.

Configurazione chiave:

- Storage su filesystem locale (`/loki/chunks`) con retention di 744 ore (31 giorni).
- Limite di ingestione configurato per gestire log di grandi dimensioni (10MB per messaggio gRPC).
- Autenticazione disabilitata per ambiente di sviluppo locale.

9.3 Tempo: Tracce Distribuite

Grafana Tempo (v2.9.0) è il backend per le tracce distribuite. Riceve gli span OTLP dall'OTel Collector (che a sua volta li riceve da n8n) e li rende navigabili in Grafana.

Configurazione chiave:

- Receiver OTLP su HTTP (4318) e gRPC (4317).
- Storage locale con blocchi compressi in `/var/tempo/traces`.
- SLO configurato a 5 secondi per ricerca tracce e lookup per TraceID.

9.4 Prometheus: Metriche

Prometheus (v2.53.0) è il backend per le metriche. Non esegue scraping diretto dei target: le metriche arrivano esclusivamente via Remote Write dall'OTel Collector, che a sua volta fa scraping del Redis Exporter.

Metriche disponibili: Metriche Redis esposte dal `redis-exporter` (connessioni attive, memoria utilizzata, operazioni al secondo, chiavi per database, etc.).

9.5 Grafana: Dashboard e Correlazione

Grafana (v12.2) è il frontend di visualizzazione, preconfigurato tramite provisioning automatico con tre datasource:

Datasource	**Backend**	**Default**	**Funzionalità Speciale**
Loki	`loki:3100`	Sì	Derived fields per correlazione log → tracce via `trace_id`
Tempo	`tempo:3200`	No	TraceQL, correlazione tracce → log via Loki
Prometheus	`prometheus:9090`	No	PromQL per metriche Redis

Tabella 9.3: Datasource Grafana preconfigurati

Inserire qui screenshot della dashboard Grafana con log Loki

Figura 9.2: Dashboard Grafana — Visualizzazione log aggregati da Loki

Figura 9.3: Grafana Tempo — Visualizzazione tracce distribuite di una esecuzione workflow n8n

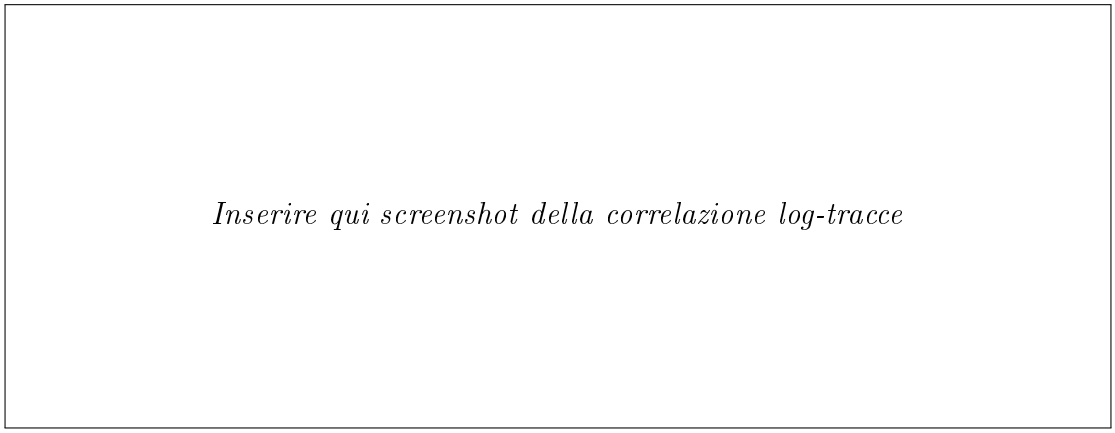

Figura 9.4: Correlazione log → tracce in Grafana tramite `trace_id`

Dettaglio Tecnico: Correlazione Log–Tracce

La correlazione tra log e tracce è implementata tramite *derived fields* nel datasource Loki. Il campo `trace_id` presente nei log JSON di n8n viene estratto tramite la regex `‘"trace_id":"(\w+)"‘` e linkato al datasource Tempo. In Grafana, questo si traduce in un link cliccabile accanto a ogni riga di log che contiene un `trace_id`, permettendo di navigare direttamente alla traccia corrispondente per analizzare l'intera esecuzione del workflow.

Capitolo 10

Design dei System Prompt

Il design dei system prompt è un aspetto critico del DIM. Ogni agente LLM è dotato di un system prompt ingegnerizzato secondo una struttura comune ma adattato alle specificità di ciascuna fase dell'intervista DDD.

10.1 Struttura Comune dei System Prompt

Tutti gli agenti intervistatori condividono una struttura a cinque sezioni:

1. **ROLE & PERSONA:** Definisce l'identità dell'agente come Senior Architect specializzato in un aspetto specifico del DDD. Ogni agente ha un titolo e una missione unici.
2. **OUTPUT RULES (STRICT ENFORCEMENT):** Regole rigide che governano il comportamento dell'agente:
   - Lingua: interazione esclusivamente in italiano.
   - Dialogo diretto con il cliente, senza meta-commenti.
   - Divieto assoluto di mostrare il ragionamento interno.
   - Divieto di usare gergo tecnico DDD nelle domande (traduzione in scenari di business).
   - Stile professionale, diretto e autorevole.
3. **BEHAVIORAL RULES:** Regole comportamentali operative:
   - Una domanda alla volta.
   - Progressione sequenziale nella propria fase.
   - Drill-down obbligatorio su risposte vaghe.
   - Reindirizzamento se l'utente devia verso aspetti UI/UX.
4. **DISCOVERY FLOW (Fase Specifica):** Definizione precisa del goal, del focus tematico e dell'anteprima della fase successiva (per il "bridge" nel handover).
5. **HANDOVER RULE (Phase Completion):** Condizioni precise per emettere il tag di completamento fase, incluso l'obbligo di riassumere i risultati e porre la prima domanda della fase successiva prima del tag.

10.2 Agenti Post-Discovery

DDD Analyst: Il system prompt istruisce l'agente a trasformare lo storico conversazionale in un Architectural Design Document strutturato in 5 pilastri (Strategic Analysis, Boundary Definition, EDA Integration, Tactical Patterns, Technical Excellence). Segue

una "Zero-Loss Policy" che impone la preservazione di ogni dettaglio tecnico menzionato durante l'intervista. L'output è esclusivamente in italiano e in formato Markdown puro.

JSON Coder: Il system prompt definisce uno schema JSON target rigido e istruisce l'agente a eseguire una traduzione 1:1 dal Markdown al JSON, senza aggiungere, riassumere o omettere informazioni. L'output deve essere JSON valido puro, senza wrapper o commenti.

Capitolo 11

Conclusioni e Sviluppi Futuri

11.1 Sintesi del Lavoro Svolto

Il Domain Interviewer & Modeler rappresenta l'implementazione completa dell'Agente 1 della pipeline *Intelligent Domain Architect Agents*. Il sistema dimostra come sia possibile automatizzare la fase di Domain Discovery del DDD attraverso una catena di agenti LLM specializzati, orchestrati da n8n e supportati da un'infrastruttura a microservizi containerizzata e completamente osservabile.

I risultati principali del progetto sono:

- **Pipeline di intervista strutturata:** Cinque agenti specializzati che coprono sistematicamente tutti i pilastri del DDD Problem Space, garantendo una copertura completa e ripetibile.
- **Generazione automatica del modello:** Trasformazione automatica dello storico conversazionale in un documento architetturale Markdown e in un JSON strutturato formale.
- **Raffinamento interattivo:** Ciclo di revisione con meccanismo Draft/Confirm/Discard che permette allo stakeholder di perfezionare iterativamente il modello generato.
- **Osservabilità completa:** Stack OpenTelemetry end-to-end con correlazione automatica log-tracce che garantisce visibilità totale sull'esecuzione del sistema.
- **Privacy by design:** Inferenza LLM completamente locale tramite Ollama, senza dipendenze da servizi cloud.

11.2 Sviluppi Futuri

- **Integrazione con Agente 2 e Agente 3:** Collegamento a valle con gli agenti downstream della pipeline (Requirement Analyzer, Microservice Designer) per completare la catena automatica dal domain discovery al design architetturale.
- **Multi-tenancy e autenticazione:** Implementazione di autenticazione utente e isolamento delle sessioni per supportare l'uso simultaneo da parte di più stakeholder.
- **Miglioramento dei modelli LLM:** Sostituzione di llama3 con modelli più potenti o fine-tuned per il dominio DDD, potenzialmente con supporto a context window più ampie per gestire interviste più lunghe.
- **Dashboard di osservabilità dedicate:** Creazione di dashboard Grafana specifiche per il monitoraggio delle performance degli agenti LLM (latenza per fase, qualità delle risposte, tasso di drill-down).
- **Export e integrazione:** Esportazione del modello JSON in formati standard (AsyncAPI, EventCatalog) per integrazione diretta con strumenti di sviluppo e documentazione architetturale.