UNIVERSITÀ DEGLI STUDI DI SALERNO

Dipartimento di Informatica

Corso di Laurea Magistrale in Informatica

PRE-MAINTENANCE REPORT

INGEGNERIA, GESTIONE ED EVOLUZIONE DEL SOFTWARE

DOCENTI

Prof. Andrea De Lucia

TUTOR

Dott. Stefano Lambiase

Università degli Studi di Salerno

STUDENTI

Francesco Maria Torino

(0522501879)

Francesco A. Pinto

(0522501981)

Stefano Guida

(0522502054)

Anno Accademico 2024-2025

Indice

Elenco delle figure

Elenco delle tabelle

CAPITOLO 1

Introduzione

1.1 Contesto del Progetto

La crescente diffusione di team di sviluppo distribuiti, spesso eterogenei per background culturale e linguistico, ha evidenziato l'importanza di strumenti capaci di analizzare non solo gli aspetti tecnici, ma anche le dinamiche socio-organizzative delle community di sviluppo software. In tale contesto, si inserisce il concetto di *community smells*, ovvero indicatori di problematiche ricorrenti nella comunicazione e collaborazione tra sviluppatori, che possono compromettere la sostenibilità e la qualità dei progetti software nel lungo termine.

1.1.1 GUIDO

GUIDO (Gathering and Understanding Socio-Technical Aspects in Development Organizations) è una piattaforma software progettata per supportare i project manager nell'identificazione e nella gestione dei community smells nelle community di sviluppo su GitHub. Inizialmente concepito come applicazione desktop autonoma, GUIDO consente l'esecuzione locale di **CADOCS** (Conversational Agent for the Detection Of Community Smells), un agente conversazionale sviluppato originariamente da

Voria et al. [1] e presentato alla conferenza ICSME 2022. CADOCS rappresenta un'evoluzione del tool CSDetector, migliorandone l'accessibilità grazie a un'interfaccia conversazionale e all'integrazione con Slack.

Nel corso del tempo, GUIDO è stato esteso e potenziato. Attualmente integra al suo interno due strumenti principali.

- **CADOCS**: chatbot basato su modelli di NLP/NLU, che permette agli utenti di richiedere informazioni sui community smells e di eseguire analisi su repository GitHub. Le funzionalità di rilevamento sono basate su CSDetector, un tool che utilizza metriche socio-tecniche per identificare dieci diversi community smells attraverso l'analisi storica delle interazioni tra sviluppatori.
- **Culture Inspector**: modulo sviluppato per analizzare la dispersione culturale e geografica di un team di sviluppo. A partire dalle informazioni relative alla nazionalità dei membri del team, il tool è in grado di calcolare metriche utili a valutare il rischio di insorgenza di community smells derivanti da disallineamenti culturali o geografici.

1.1.2 TOAD

TOAD (Tool for Organizational and Activity Diagnosis) TOAD è uno strumento open-source progettato per analizzare lo stato di salute delle community di sviluppo open-source. Il suo obiettivo è rilevare fino a 8 community pattern nella community di sviluppo analizzando caratteristiche chiave, quali structure, geodispersion, formality, engagement e longevity. TOAD permette di valutare lo stato della community lungo tre dimensioni teoriche principali, ognuna delle quali è rappresentata da un insieme specifico di metriche:

- **Activity**: Misura l'engagement della community tramite metriche come numero di commit, numero di contributors attivi, frequenza delle modifiche nel tempo
- **Cohesion**: Analizza le connessioni sociali tra sviluppatori tramite la densità del grafo, il numero di componenti connesse, ecc.

- **Resilience**: Valuta la capacità della community di resistere a shock o perdita di sviluppatori chiave

1.2 Obiettivi del Progetto

L'obiettivo generale di questo progetto riguarda proprio l'integrazione del tool TOAD all'interno della piattaforma GUIDO, al fine di arricchire le funzionalità analitiche offerte dal sistema e fornire una visione più completa dello stato di salute delle community di sviluppo software. A tal fine, gli obiettivi specifici del progetto sono articolati come segue:

- **Realizzazione di un wrapper RESTful**, che consenta di esporre le funzionalità di TOAD tramite interfacce web, rendendolo accessibile come servizio esterno e facilmente integrabile nel flusso operativo di GUIDO;
- **Dockerizzazione del tool TOAD**, con l'obiettivo di assicurarne la portabilità, la replicabilità e una gestione semplificata dell'infrastruttura, facilitando così il suo utilizzo in ambienti eterogenei;
- **Integrazione dei risultati analitici di TOAD all'interno della GUI di GUIDO**, garantendo la visualizzazione chiara, contestuale e centralizzata degli output generati, in modo da supportare efficacemente gli utenti nell'interpretazione dello stato di salute delle community analizzate.

CAPITOLO 2

Reverse engineering GUIDO

2.1 Analisi di GUIDO

Prima di procedere con le modifiche e la stesura delle change request, abbiamo ritenuto fondamentale analizzare gli artefatti prodotti dagli sviluppatori precedenti, al fine di ottenere una visione completa del prodotto software **GUIDO**.

La nostra prima azione è stata quella di esaminare attentamente la documentazione pregressa di GUIDO. Questo ci ha permesso di acquisire una panoramica generale della sua struttura e del suo funzionamento. Una prima overview dell'architettura di GUIDO è illustrata nella Figura 2.1, che rappresenta il package diagram del sistema. Successivamente, abbiamo intrapreso un'analisi approfondita del codice sorgente, modulo per modulo. Poiché la documentazione esistente non includeva un diagramma delle classi che rappresentasse l'ultima versione di GUIDO, abbiamo ritenuto indispensabile crearne uno. Questo nuovo diagramma, rappresentato nella Figura 2.2 delinea in modo preciso le relazioni tra i vari componenti all'interno del sistema, fornendo una rappresentazione chiara della sua struttura interna.

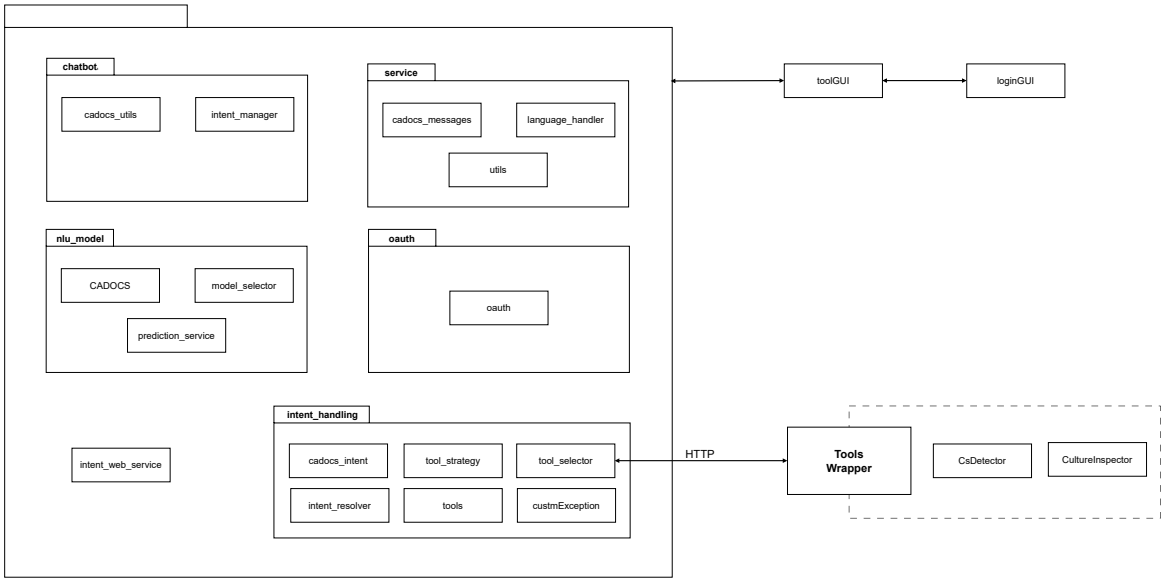

Figura 2.1: Package Diagram di GUIDO

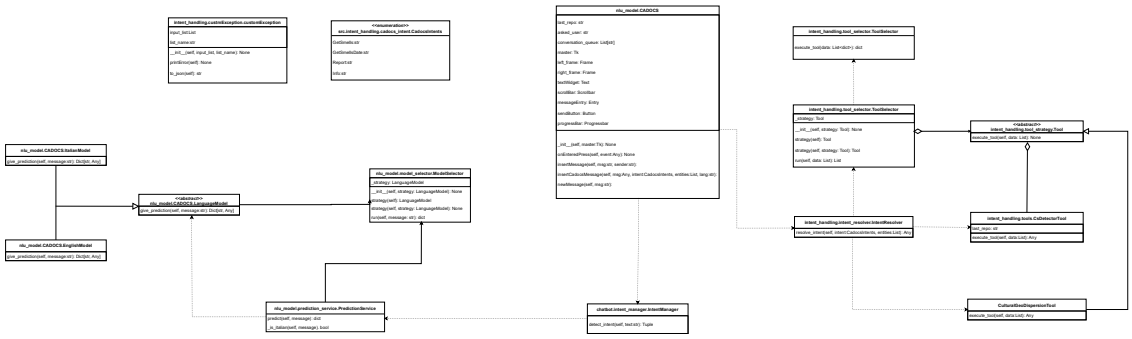

Figura 2.2: Class Diagram di GUIDO

Di seguito, presentiamo l'analisi dei principali componenti del sistema:

- **Servizio Web Flask** (`src/intent_web_service.py`): Questo componente funge da punto di ingresso per tutte le richieste in arrivo. La sua responsabilità principale è orchestrare l'intero processo di risoluzione degli intenti dell'utente.
- **Intent Manager** (`src/chatbot/intent_manager.py`): Utilizzando tecniche di Natural Language Understanding (NLU), questo gestore rileva con precisione l'intento e le entità chiave dal testo fornito dall'utente.
- **Modello NLU** (`src/nlu_model/CADOCS.py`): Il cuore della comprensione linguistica di GUIDO. Questo modulo si avvale di *modelli RoBERTa* pre-

addestrati per classificare gli intenti, garantendo il supporto sia per la lingua italiana che per quella inglese.

- **Intent Resolver** (`src/intent_handling/intent_resolver.py`): Una volta identificato l'intento, questo risolutore ha il compito di selezionare ed eseguire lo strumento più appropriato tra quelli disponibili, basandosi sull'intento riconosciuto.

- **Tools** (`src/intent_handling/tools.py`): Si tratta di classi specifiche che implementano azioni distinte, come il rilevamento di code smell e l'analisi culturale.

2.2 Analisi delle dipendenze tra moduli

Per ottenere una visione più dettagliata della struttura del sistema e delle interazioni tra i suoi moduli, è stata calcolata una matrice delle dipendenze relativa ai file sorgente del progetto GUIDO. Tale analisi consente di valutare il grado di accoppiamento tra i moduli e di condurre una prima *impact analysis* ad alto livello, utile nella fase preliminare di definizione delle Change Request. La Tabella 2.1 riporta la matrice delle dipendenze: righe e colonne rappresentano i file Python costituenti il sistema, e ogni cella (i, j) contiene il numero di chiamate effettuate dal modulo i verso funzioni o classi definite nel modulo j.

Dall'analisi complessiva si osserva che il sistema è composto da 18 moduli, e la matrice risulta relativamente sparsa: questo suggerisce un grado di accoppiamento contenuto, elemento positivo in termini di manutenibilità e modularità. Tuttavia, si evidenziano alcune considerazioni degne di nota:

- Alcuni moduli, come `intent_manager.py` e `intent_resolver.py`, presentano un numero consistente di dipendenze in uscita. Tuttavia, in questi casi, l'elevato numero di interazioni risulta giustificato e coerente con il ruolo centrale di orchestrazione logica che tali componenti ricoprono all'interno del sistema.

- È opportuno monitorare con attenzione i moduli molto utilizzati da altri componenti (i.e., con molte dipendenze in ingresso), come `cadocs_intent.py`, poiché eventuali modifiche a tali moduli potrebbero avere un impatto significativo sull'intero sistema.

- Alcuni file, tra cui `oauth.py` e `custmException.py`, non mostrano alcuna dipendenza in ingresso o in uscita. Un'analisi più approfondita ha confermato che tali moduli non sono attualmente utilizzati: si tratta verosimilmente di codice obsoleto o non più mantenuto.

- Durante la costruzione della matrice sono emersi diversi `import` non utilizzati, oppure utilizzati in porzioni di codice commentate. Questo potrebbe indicare la necessità di un'attività di refactoring per rimuovere dipendenze ridondanti o inattive, migliorando la chiarezza e la pulizia del codice.

Tabella 2.1: GUIDO: Matrice delle Dipendenze

File	loginGui.py	toolGui.py	cadocs_utils.py	intent_manager.py	cadocs_intent.py	custmException.py	intent_resolver.py	tool_selector.py	tool_strategy.py	tools.py	CADOCS.py	model_selector.py	prediction_service.py	oauth.py	cadocs_messages.py	language_handler.py	utils.py	intent_web_service.py
loginGui.py	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
toolGui.py	1	0	0	1	5	0	0	0	0	0	0	0	0	0	3	0	0	1
cadocs_utils.py	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
intent_manager.py	0	0	1	0	4	0	0	0	0	0	0	0	5	0	0	0	0	0
cadocs_intent.py	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
custmException.py	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
intent_resolver.py	0	0	0	0	5	0	0	2	0	2	0	0	0	0	0	0	0	0
tool_selector.py	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0
tool_strategy.py	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
tools.py	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0
CADOCS.py	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
model_selector.py	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0
prediction_service.py	0	0	0	0	0	0	0	0	0	0	2	2	0	0	0	0	0	0
oauth.py	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
cadocs_messages.py	0	0	0	0	5	0	0	0	0	0	0	0	0	0	0	0	0	0
language_handler.py	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
utils.py	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
intent_web_service.py	0	0	0	1	2	0	1	0	0	0	0	0	0	0	1	0	0	0

2.3 Requisiti Funzionali

Nella Tabella 2.2 sono indicati i requisiti funzionali di GUIDO estratti a seguito di un'analisi del prodotto finale, della documentazione preesistente e del codice.

ID	Nome	Descrizione
RF01	Info	Permette all'utente di richiedere al chatbot informazioni sulle sue funzionalità e sui community smells che è in grado di rilevare.
RF02	Get Smells	Consente all'utente di richiedere al chatbot un'analisi di una repository github per ottenere un report con tutti i Community Smells rilevati.
RF03	Get Smells By Date	Consente all'utente di richiedere al chatbot un'analisi di una repository github per ottenere un report con tutti i Community Smells rilevati a partire da un determinato periodo temporale.
RF02	Get Cultural Dispersion	Consente all'utente di richiedere al Culture Inspector informazioni sulla dispersione culturale e geografica di una community o di un team di sviluppatori.

Tabella 2.2: GUIDO: Requisiti Funzionali

2.4 Use Case

2.4.1 Info

Identificativo	UC01
Nome	Info
Descrizione	Lo Use Case si riferisce alla funzionalità del sistema di fornire informazioni generali relative ai Community Smells.
Attore Principale	Utente
Attori Secondari	CADOCS NLU
Entry Condition	L'utente deve aver installato il tool e deve trovarsi nella finestra dedicata al chatbot di GUIDO.
Exit Condition (successo)	L'utente visualizza tutte le informazioni sui community smells.
Exit Condition (fallimento)	L'utente non visualizza le informazioni sui community smells.
Rilevanza	Media
Flusso principale	1. L'utente richiede informazioni sui community smells. 2. Il sistema inoltra la richiesta a CADOCS NLU. 3. Il sistema riceve la risposta da CADOCS NLU contenente l'intent della richiesta. 4. Il sistema mostra le informazioni sui community smells.
Flusso alternativo	3.a1 Il modello non riesce a interpretare la richiesta: il sistema non mostra nulla.

Tabella 2.3: GUIDO: Use Case 01 - Info

2.4.2 Get Smells

Identificativo	UC02
Nome	Get Smells
Descrizione	Lo Use Case fa riferimento alla funzionalità del sistema che permette di analizzare una repository GitHub per identificare i community smells al suo interno.
Attore Principale	Utente
Attori Secondari	CADOCS NLU, CS Detector
Entry Condition	L'utente si trova nella finestra di GUIDO dedicata al Chatbot e vuole trovare i community smells di una repository.
Exit Condition (successo)	L'utente visualizza i community smells identificati.
Exit Condition (fallimento)	L'utente non visualizza i community smells.
Rilevanza	Alta
Flusso principale	1. L'utente richiede di analizzare una repository github fornendo il link della repository. 2. Il sistema rileva il linguaggio della richiesta. 3. Il sistema inoltra la richiesta al modello di NLU corretto (ITA o ENG). 4. Il sistema riceve la risposta da CADOCS NLU contenente l'intent della richiesta. 5. Il sistema inoltra la richiesta di analisi a CS Detector. 6. Il sistema riceve la risposta da CS Detector e i risultati dell'analisi all'utente.
Flusso alternativo	5.a1 Il modello non riesce a interpretare la richiesta: il sistema non mostra nulla.

Tabella 2.4: GUIDO: Use Case 02 - Get Smells

2.4.3 Get Smells By Date

Identificativo	UC03
Nome	Get Smells By Date
Descrizione	Lo Use Case si riferisce alla funzionalità che permette di analizzare i community smells presenti in una repository GitHub a partire da una certa data.
Attore Principale	Utente
Attori Secondari	CADOCS NLU, CS Detector
Entry Condition	L'utente si trova nella finestra di GUIDO dedicata al Chatbot vuole trovare i community smells di una repository da una certa data in poi.
Exit Condition (successo)	L'utente visualizza i community smells identificati a partire dalla data specificata.
Exit Condition (fallimento)	L'utente non visualizza i community smells.
Rilevanza	Alta
Flusso principale	1. L'utente richiede di analizzare una repository github fornendo il link della repository e la data da cui partire per trovare i community smells. 2. Il sistema rileva il linguaggio della richiesta. 3. Il sistema inoltra la richiesta al modello di NLU corretto (ITA o ENG). 4. Il sistema riceve la risposta da CADOCS NLU contenente l'intent della richiesta e l'entity della data. 5. Il sistema inoltra la richiesta a CS Detector. 6. Il sistema riceve la risposta da CS Detector e mostra i risultati dell'analisi all'utente.
Flusso alternativo	5.a1 Il modello non riesce a interpretare la richiesta: il sistema non mostra nulla.

Tabella 2.5: GUIDO: Use Case 03 - Get Smells By Date

2.4.4 Get Cultural Dispersion

Identificativo	UC04
Nome	Get Cultural Dispersion
Descrizione	Lo Use Case fa riferimento alla funzionalità che permette di calcolare la dispersione culturale e geografica di una community di sviluppatori.
Attore Principale	Utente
Attori Secondari	Culture Inspector
Entry Condition	L'utente si trova nella finestra di GUIDO dedicata al Culture Inspectore e vuole calcolare metriche di dispersione culturale e geografica di un team di sviluppatori.
Exit Condition (successo)	L'utente visualizza le informazioni sulla dispersione culturale e geografica del team.
Exit Condition (fallimento)	L'utente non visualizza le metriche richieste.
Rilevanza	Alta
Flusso principale	1. L'utente richiede di calcolare la dispersione culturale di un team inserendo la composizione del team di sviluppo e selezionando la nazionalità e il numero dei vari team member. 2. Il sistema inoltra la richiesta al Culture Inspector. 3. Il sistema riceve la risposta dal Culture Inspector. 4. Il sistema mostra all'utente le Metriche di Dispersione Culturale e Geografica del team di sviluppo.
Flusso alternativo	4.a1 Il tool non riesce a interpretare la richiesta: il sistema non mostra nulla.

Tabella 2.6: GUIDO: Use Case 04 - Get Cultural Dispersion

2.5 Sequence Diagrams

2.5.1 Info

Il diagramma di sequenza riportato in Figura 2.3 illustra il flusso di interazioni tra i principali componenti del sistema durante l'esecuzione della funzionalità di richiesta di informazioni generali sui *community smells*, attivata dall'utente tramite l'interfaccia chatbot di GUIDO.

1. L'utente invia una richiesta informativa relativa ai *community smells* attraverso l'interfaccia conversazionale di GUIDO.
2. La richiesta viene presa in carico dal modulo **Intent Manager**, il quale invoca il **Language Handler** per determinare automaticamente la lingua del messaggio, inoltra il messaggio al componente NLU (**CADOCS_NLU**) per la predizione dell'intento, trasmette i risultati al modulo **Intent Resolver** per la generazione della risposta contestuale.
3. In questo specifico scenario, il **Resolver** produce direttamente una risposta testuale, senza richiedere l'interazione con strumenti esterni, che viene infine restituita all'utente e visualizzata nella chat.

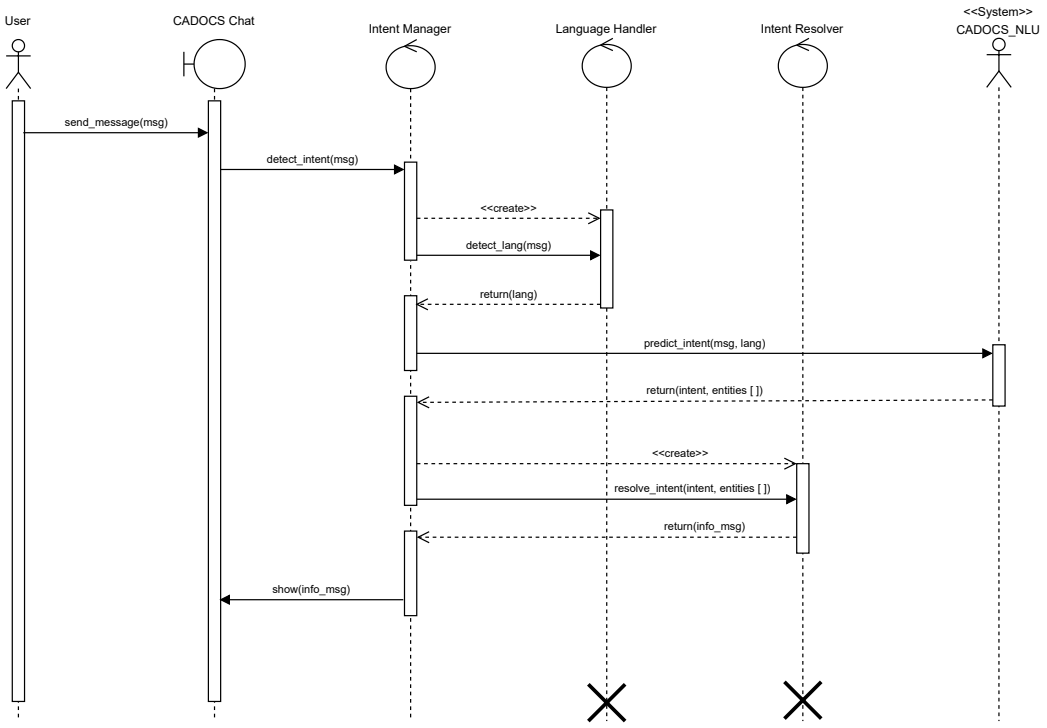

Figura 2.3: Sequence Diagram: Info

2.5.2 Get Smells (By Date)

Il diagramma di sequenza riportato in Figura 2.4 illustra il flusso di interazioni tra i principali componenti del sistema durante l'esecuzione della funzionalità di richiesta di analisi dei *community smells* all'interno di una repository GitHub, attivata dall'utente mediante l'interfaccia conversazionale di GUIDO:

1. L'utente invia, tramite la chat di GUIDO, una richiesta per l'analisi di una repository GitHub ai fini dell'individuazione di eventuali *community smells*.
2. La richiesta viene gestita dal modulo **Intent Manager**, che invoca il **Language Handler** per rilevare automaticamente la lingua del messaggio, inoltra il contenuto al componente NLU (**CADOCS_NLU**) per la predizione dell'intento e l'estrazione delle entità rilevanti e trasmette i risultati ottenuti al modulo **Intent Resolver**.
3. L' **Intent Resolver** elabora l'intento ricevuto e, tramite il componente **Tool Selector**, inoltra la richiesta al **CS Detector**, responsabile dell'esecuzione dell'analisi sulla repository. I risultati ottenuti vengono formattati e propagati a ritroso nel flusso per essere visualizzati all'interno della chat.

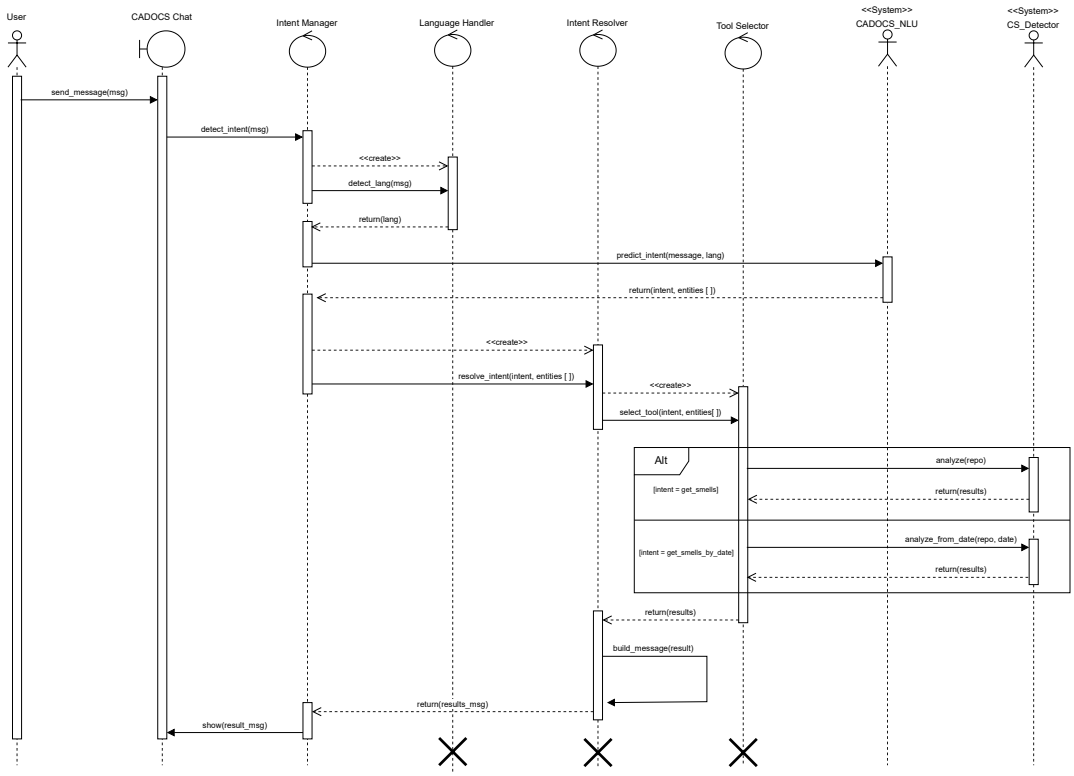

Figura 2.4: Sequence Diagram: Get Smells e Get Smells By Date

2.5.3 Get Cultural Dispersion

Il diagramma di sequenza riportato in Figura 2.5 rappresenta il flusso di interazioni tra i principali componenti del sistema durante l'esecuzione della funzionalità di calcolo della dispersione culturale all'interno di una community di sviluppo software.

1. L'utente, attraverso l'interfaccia grafica di GUIDO dedicata al **Culture Inspector**, seleziona la composizione del team e le informazioni relative alla distribuzione geografica dei membri. Successivamente, invia la richiesta di analisi al sistema.
2. La richiesta viene gestita dal modulo **Intent Manager**, che in questo caso specifico assegna direttamente l'intento associato all'analisi culturale (senza coinvolgere il componente di NLU) e inoltra la richiesta al modulo **Intent Resolver**.
3. L'**Intent Resolver** interpreta l'intento ricevuto e, tramite il componente **Tool Selector**, instrada la richiesta al modulo **Culture Inspector**, responsabile dell'elaborazione dell'analisi sulla dispersione culturale e geografica del team. I risultati ottenuti vengono formattati e trasmessi a ritroso per essere visualizzati all'interno della sezione dedicata dell'interfaccia utente.

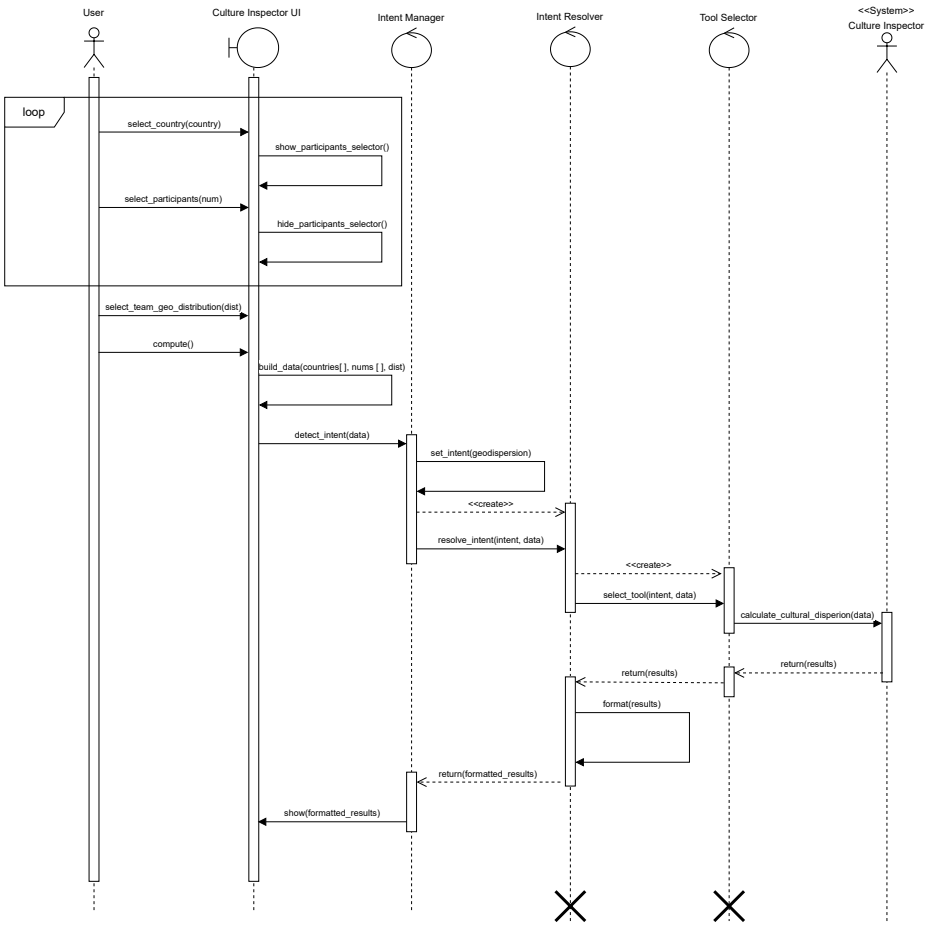

Figura 2.5: Sequence Diagram: Get Cultural Dispersion

CAPITOLO 3

Reverse Engineering TOAD

TOAD (*Tool for Organizational and Activity Diagnosis*) è uno strumento open-source progettato per analizzare lo stato di salute delle community di sviluppo software, rilevando ricorrenze comportamentali e organizzative (*community patterns*) sulla base di metriche socio-tecniche.

In vista della sua futura integrazione all'interno della piattaforma GUIDO, si rende necessaria un'analisi preliminare del funzionamento di TOAD, con l'obiettivo di garantire un'invocazione corretta e automatizzata dello strumento da parte di un wrapper esterno. Poiché non è prevista una modifica diretta sostanziale del codice sorgente di TOAD, l'attività di *reverse engineering* qui condotta si limita all'identificazione delle strutture essenziali del sistema, utili per orchestrarne l'esecuzione in modo efficace. L'analisi si concentra quindi su due aspetti principali:

- l'**organizzazione dei moduli**, per comprendere la suddivisione funzionale del codice e individuare i punti di ingresso rilevanti;
- i **meccanismi di input/output**, per analizzare il formato dei dati richiesti e prodotti, nonché la struttura delle directory e dei file coinvolti.

Queste informazioni risultano fondamentali per progettare un'integrazione robusta e sostenibile, che tenga conto anche delle caratteristiche operative dello strumento: per esempio, durante le prove sperimentali è emerso che l'analisi di una singola repository richiede un tempo minimo di circa **12 minuti**, rendendo necessaria una gestione asincrona ed efficiente del processo di invocazione.

3.1 Organizzazione dei Moduli

L'architettura di TOAD segue un paradigma modulare, in cui ciascun componente è responsabile di una fase distinta del processo di raccolta, elaborazione e analisi dei dati. Il diagramma ad alto livello è riportato nella Figura 3.1.

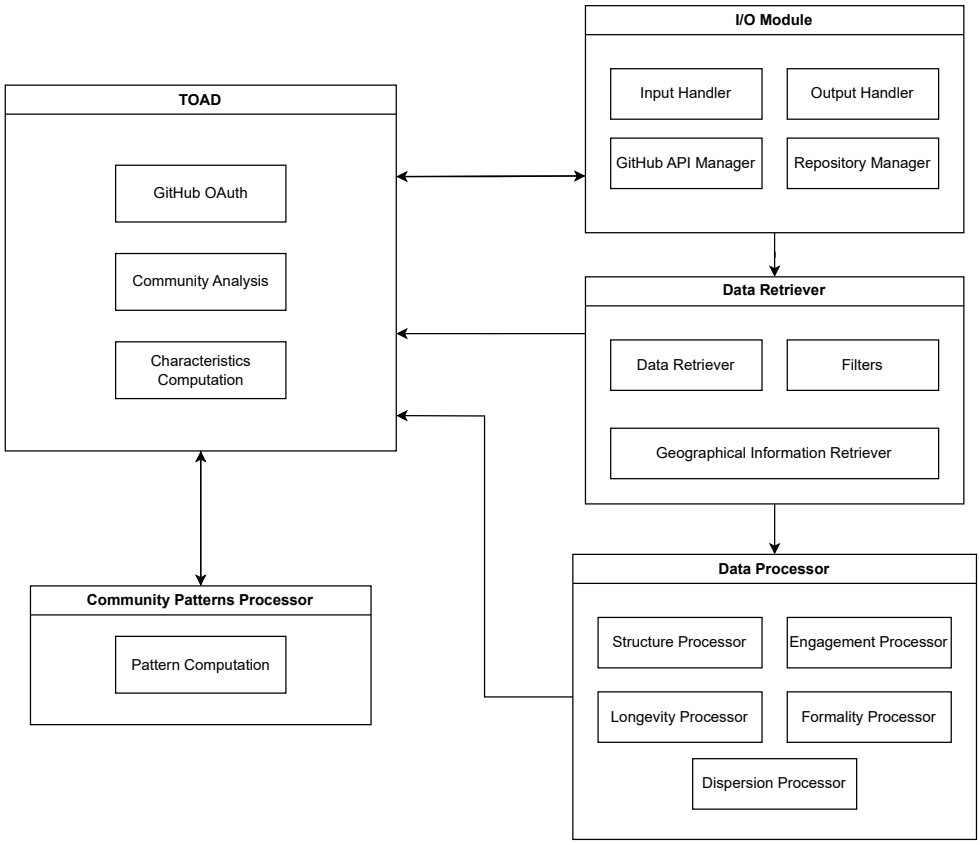

Figura 3.1: Architettura di TOAD

Di seguito sono descritte le principali macro-componenti:

- **TOAD (Core)**: rappresenta il nucleo applicativo, da cui viene avviato il processo di analisi. Contiene le logiche per l'autenticazione con GitHub, la gestione delle repository e il coordinamento delle fasi successive.
- **I/O Module**: responsabile della gestione dell'interazione con l'utente e con il file system. Comprende:
  - *Input Handler*: per la lettura e validazione dei file CSV in input;
  - *Output Handler*: per la scrittura dei risultati su file, in vari formati;
  - *GitHub API Manager*: per l'interrogazione delle API pubbliche di GitHub;
  - *Repository Manager*: per il tracciamento e la gestione delle repository analizzate.

- **Data Retriever**: modulo dedicato all'estrazione delle informazioni dalle repository GitHub. Include:
  - *Event Retriever*: per il recupero delle attività storiche;
  - *Filter Engine*: per l'applicazione di vincoli temporali e qualitativi;
  - *Geo Info Retriever*: per l'arricchimento dei dati con informazioni geografiche.
- **Data Processor**: modulo che elabora i dati grezzi e calcola le metriche fondamentali. Include *Structure Processor*, *Engagement Processor*, *Longevity Processor*, *Formality Processor*, *Dispersion Processor*, ciascuno specializzato su una metrica specifica.
- **Community Patterns Processor**: modulo che applica regole euristiche sulle metriche calcolate per rilevare la presenza di determinati pattern organizzativi nella community.

3.2 Meccanismi di Input/Output

Il processo di analisi in TOAD prevede una fase iniziale di raccolta dell'input, seguita da un'elaborazione iterativa e infine dalla produzione di molteplici file di output organizzati in directory specifiche.

3.2.1 Input

L'input principale è un file `.csv` contenente, per ciascuna repository da analizzare, le seguenti colonne:

- **Author**: nome dell'utente GitHub;
- **Repository**: nome del repository;
- **End Date**: data finale dell'intervallo di analisi (si estende per 90 giorni a ritroso).

Di seguito un esempio di come deve essere strutturato il file di input in formato CSV:

```
netty,netty,2017-05-01
bundler,bundler,2017-05-01
composer,composer,2017-05-01
```

Il punto di ingresso per l'avvio di TOAD è lo script `pattern_detection.py`, che esegue interazioni tramite riga di comando per raccogliere da tastiera i seguenti input:

- il percorso del file CSV in input;
- la directory di destinazione dei risultati;
- il nome del file CSV in cui memorizzare i risultati.

Dopo aver validato le informazioni fornite, TOAD inizia l'analisi iterativa di ciascuna repository. Durante questa fase, le repository vengono clonate localmente e sottoposte a una procedura di verifica che ne assicura l'idoneità (es. almeno 100 commit nel periodo analizzato, almeno una milestone, almeno due membri attivi). Solo in caso di superamento della verifica viene eseguita l'analisi vera e propria.

3.3 Output

Al termine dell'analisi, TOAD produce diversi artefatti distribuiti su più directory. Di seguito vengono descritti i principali output generati.

3.3.1 Output CSV

Il file CSV di output, posizionato nella directory specificata dall'utente, conterrà una riga per ogni repository analizzata, con le seguenti informazioni:

- Owner della repository;
- Nome del repository;
- Data di inizio e fine dell'analisi;
- Esito del rilevamento per ciascun community pattern (valori booleani).

Esempio di output in formato CSV:

```
owner,name,start_date,end_date,SN,NoP,IN,FN,CoP,PT,FG,IC
netty,netty,2017-01-31,2017-05-01,True,True,False,True,
False,False,False,True
bundler,bundler,2017-01-31,2017-05-01,True,False,False,
False,True,True,False,True
composer,composer,2017-01-31,2017-05-01,True,True,False,True,
False,False,False,False
```

3.3.2 Metriche JSON

Per ogni repository analizzata, TOAD salva un file `metrics.json` nella cartella `data/author/repo/`, contenente tutte le metriche calcolate, suddivise per tipologia. Di seguito un esempio di file JSON contenente le metriche calcolate da TOAD:

```
{
  "dispersion": {
    "geo_distance_variance": 17959025.225662425,
    "avg_geo_distance": 7122.194073794376,
    "cultural_distance_variance": 21.5851795725958
  },
  "engagement": {
    "m_comment_per_pr": 0.5,
    "mm_comment_dist": 2,
    "m_watchers": 0.0,
    "m_stargazers": 0.0,
    "m_active": 1.0,
    "mm_commit_dist": 0.0,
    "mm_filecollab_dist": 1.0
  },
  "formality": {
    "m_membership_type": 1.8372093023255813,
    "milestones": 271,
    "lifetime": 5466
  },
  "longevity": 21.205882352941178,
  "structure": {
    "repo_connections": true,
    "follow_connections": true,
    "pr_connections": true
  }
}
```


3.3.3 Grafi delle interazioni

TOAD genera anche una rappresentazione del grafo delle interazioni tra contributor, dove i nodi rappresentano i membri della community e gli archi rappresentano le interazioni tra i membri (eg. coomit condivisi, issue comuni). Inoltre, ad ogni arco è associato un peso che rappresenta la

"forza" dell'interazione. Tutte queste informazioni vengono salvate in `graphs/author/repo/` in due formati:

- `.png`: visualizzazione statica del grafo;
- `.gexf`: formato interoperabile per strumenti come Gephi.

In caso di grafo di grandi dimensioni, l'output viene suddiviso in più file. Nella Figura 3.2 è possibile osservare un esempio di png del grafo generato da TOAD, mentre di seguito viene mostrato un esempio semplificato di file GEXF:

```
<?xml version='1.0' encoding='utf-8'?>
<gexf xmlns="http://www.gexf.net/1.2draft"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.gexf.net/1.2draft
                          http://www.gexf.net/1.2draft/gexf.xsd"
      version="1.2">
  <meta lastmodifieddate="2023-08-28">
    <creator>NetworkX 3.1</creator>
  </meta>
  <graph defaultedgetype="undirected" mode="static" name="">
    <nodes>
      <!-- Esempio di nodo -->
      <node id="runspired" label="runspired" />
      <node id="bmac" label="bmac" />
      <!-- ... altri nodi ... -->
    </nodes>
    <edges>
      <!-- Esempio di arco -->
      <edge id="0" source="runspired" target="bmac" weight="13" />
      <edge id="1" source="runspired" target="pete" weight="6" />
      <!-- ... altri archi ... -->
    </edges>
  </graph>
</gexf>
```

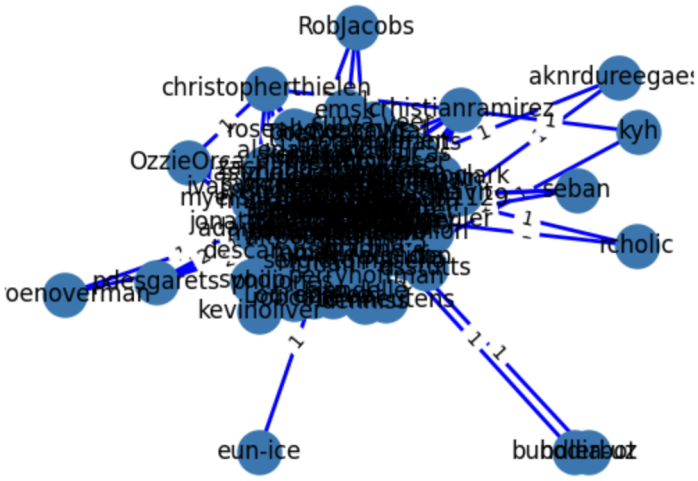

Figura 3.2: Grafo PNG restituito da TOAD

3.3.4 Gestione degli Errori

TOAD non dispone di un meccanismo strutturato per la gestione degli errori. In caso di repository non valida o di problemi durante l'esecuzione, lo strumento emette messaggi informativi sulla console standard, senza restituire eccezioni o codici di errore formali.

3.4 Considerazioni Finali

Il processo di reverse engineering condotto ha permesso di ottenere una panoramica dettagliata sul funzionamento interno di TOAD e di identificare con precisione i punti chiave per una sua futura integrazione e orchestrazione tramite wrapper esterno. L'analisi ha permesso di effettuare diverse osservazioni che guideranno la progettazione dei meccanismi di integrazione:

- TOAD richiede un'interazione diretta da tastiera per la raccolta dei parametri di input, non risultando immediatamente compatibile con esecuzioni automatizzate;
- è necessario predisporre un file CSV di input ben strutturato, la cui creazione potrebbe dover essere automatizzata da parte del wrapper;
- i risultati vengono distribuiti su molteplici file in formati eterogenei (CSV, JSON, GEXF, PNG), e potrebbero necessitare di un post-processing per essere arricchiti (es. con la descrizione delle metriche e dei community pattern rilevati);
- la visualizzazione del grafo in formato PNG risulta, nella maggior parte dei casi, illeggibile e non utile a fini analitici. Per eventuali funzionalità di visualizzazione e analisi integrate in GUIDO, è quindi raccomandato l'utilizzo del formato `.gexf`.

CAPITOLO 4

Change request

Il fulcro del presente progetto consiste nell'evoluzione della piattaforma **GUIDO** attraverso l'integrazione del tool **TOAD** e nel contestuale potenziamento delle funzionalità già esistenti. TOAD, sviluppato esternamente dal dott. Gianmario Voria, è uno strumento orientato all'analisi della salute delle community di sviluppo software, basato su un insieme di metriche socio-tecniche consolidate.

Le modifiche necessarie per raggiungere tale obiettivo sono state organizzate in quattro principali *Change Request* (CR), ciascuna delle quali rappresenta un intervento mirato volto a garantire un'integrazione funzionale, modulare ed efficiente di TOAD all'interno dell'ecosistema GUIDO. Per ogni Change Request vengono fornite tre dimensioni di analisi:

- **Priorità**: indica il livello di urgenza associato alla modifica, in relazione agli obiettivi generali del progetto e alla sua roadmap evolutiva.
- **Impatto**: descrive, a un livello architetturale e funzionale, le implicazioni che l'introduzione della modifica comporta sul sistema esistente, valutando sia effetti immediati che conseguenze a lungo termine.
- **Effort**: rappresenta la stima della complessità tecnica e del carico di lavoro richiesto per l'implementazione della modifica, sulla base di un'analisi preliminare delle dipendenze tra moduli, delle tecnologie coinvolte e dell'eventuale necessità di refactoring.

4.1 CR1: Integrazione di TOAD in GUIDO

Identificativo	CR1
Descrizione	Integrazione del tool esterno TOAD all'interno della piattaforma GUIDO, mediante la realizzazione di un *wrapper* e l'esposizione delle sue funzionalità tramite interfacce RESTful, in modo da consentirne l'utilizzo nel sistema in modo modulare e scalabile.
Motivazione	Rendere le funzionalità analitiche di TOAD accessibili agli utenti direttamente tramite GUIDO, ampliando la capacità della piattaforma di valutare lo stato di salute delle community di sviluppo.
Priorità	High[X] Medium[] Low[]
Effort	High[X] Medium[] Low[]
Conseguenze se non accettato	Le analisi socio-tecniche di TOAD non sarebbero fruibili attraverso GUIDO, riducendo la completezza delle informazioni disponibili per l'utente e limitando l'impatto decisionale del sistema.
Impatto	L'integrazione di TOAD richiederà l'introduzione di un nuovo servizio **backend**, accessibile tramite API REST. Sarà necessario: • progettare e implementare un **wrapper** per l'esposizione delle funzionalità di TOAD; • modificare il modulo **tool_selector.py** al fine di includere TOAD tra gli strumenti selezionabili da GUIDO; • aggiornare i moduli deputati alla gestione degli **intent**, in modo da abilitare l'utilizzo delle nuove funzionalità; • adeguare la **documentazione tecnica** per includere le modalità di interazione con il nuovo servizio; • progettare e sviluppare **test di integrazione** per garantire il corretto funzionamento delle interazioni tra GUIDO e TOAD. Le modifiche avranno un impatto principalmente sull'architettura *backend* e sulla logica di orchestrazione delle funzionalità offerte dal chatbot. L'integrazione sarà progettata per mantenere l'isolamento funzionale tra moduli, in modo da preservare la manutenibilità del sistema.

Tabella 4.1: Change Request 1: Integrazione di TOAD in GUIDO

4.2 CR2: Aggiunta di una GUI per utilizzare TOAD all'interno di GUIDO

Identificativo	CR2
Descrizione	Sviluppo di un'interfaccia grafica all'interno di GUIDO che consenta agli utenti di avviare analisi TOAD e di visualizzarne i risultati in modo interattivo e accessibile.
Motivazione	Migliorare l'usabilità e l'accessibilità della piattaforma, rendendo disponibili i risultati delle analisi TOAD anche a utenti non tecnici, senza necessità di interazione diretta con API o strumenti esterni.
Priorità	High[X] Medium[] Low[]
Effort	High[] Medium[X] Low[]
Conseguenze se non accettato	L'accesso ai dati prodotti da TOAD sarebbe limitato agli utenti con competenze tecniche in grado di interagire direttamente con l'API REST, riducendo l'impatto e la fruibilità complessiva della piattaforma.
Impatto	L'introduzione della GUI per TOAD comporta interventi sia lato frontend che nella logica di comunicazione con il backend. In particolare: • sarà necessario modificare il modulo di interfaccia tra **frontend e backend**, affinché supporti le chiamate alle nuove funzionalità offerte da TOAD; • dovrà essere progettata e implementata una nuova **sezione dell'interfaccia utente**, integrata nella UI esistente di GUIDO, per consentire la selezione dei repository da analizzare, l'avvio dell'analisi e la visualizzazione dei risultati; Tali modifiche avranno un impatto concentrato sul frontend, sulla logica di orchestrazione della UI e sulla comunicazione tra Frontend e Backend

Tabella 4.2: Change Request 2: Aggiunta di una GUI per TOAD

4.3 CR3: Estensione della GUI per la visualizzazione del grafo

Identificativo	CR3
Descrizione	Estensione dell'interfaccia grafica di GUIDO per includere la visualizzazione interattiva del grafo delle relazioni sociali all'interno della community, generato dai dati analizzati da TOAD.
Motivazione	Favorire una comprensione più immediata e intuitiva delle dinamiche relazionali e strutturali della community, tramite l'uso di rappresentazioni visuali esplicite e interattive.
Priorità	High[] Medium[X] Low[]
Effort	High[] Medium[X] Low[]
Conseguenze se non accettato	L'interpretazione delle informazioni relative alla struttura della community risulterebbe meno immediata, compromettendo la fruibilità dei dati TOAD da parte di utenti non esperti.
Impatto	La modifica avrà un impatto diretto sull'interfaccia utente realizzata nella **CR2**, richiedendo: • l'integrazione di **librerie di visualizzazione grafica** (es. D3.js, Cytoscape.js o analoghe) per la rappresentazione dinamica del grafo delle relazioni; • l'aggiornamento del **modulo frontend** al fine di includere una sezione dedicata alla visualizzazione interattiva del grafo, con funzionalità di zoom, selezione dei nodi e consultazione dei dettagli; • l'adattamento della **comunicazione con il backend** per ricevere i dati strutturati necessari alla costruzione del grafo, mantenendo compatibilità con le API TOAD; L'intervento, pur non modificando l'architettura backend, introduce nuove dipendenze nel frontend e una maggiore complessità nella gestione dello stato dell'interfaccia.

Tabella 4.3: Change Request 3: Estensione della GUI per la visualizzazione del grafo

4.4 CR4: Dockerizzazione del backend TOAD

Identificativo	CR4
Descrizione	Creazione di un'immagine Docker per il backend di TOAD, al fine di semplificarne la distribuzione, garantirne la portabilità e facilitarne l'integrazione all'interno dell'infrastruttura di GUIDO.
Motivazione	Standardizzare l'ambiente di esecuzione di TOAD, riducendo i problemi legati alla configurazione manuale e alle dipendenze software, e facilitare il suo utilizzo in ambienti eterogenei attraverso tecnologie containerizzate.
Priorità	High[X] Medium[] Low[]
Effort	High[] Medium[X] Low[]
Conseguenze se non accettato	L'integrazione di TOAD risulterebbe più complessa e soggetta a problemi di compatibilità tra ambienti, limitando la replicabilità e l'automazione del sistema complessivo.
Impatto	L'intervento richiederà la realizzazione di un **Dockerfile** dedicato per il backend di TOAD, comprensivo di tutte le dipendenze necessarie all'esecuzione del tool. Saranno richieste le seguenti attività: • analisi e, se necessario, **modifica delle directory** in cui TOAD memorizza i file temporanei e i risultati delle analisi, per assicurare compatibilità con il file system interno al container; • revisione degli **endpoint API**, nel caso in cui gli URL hardcoded o le configurazioni di rete non siano compatibili con un ambiente dockerizzato; • aggiornamento dei file di **docker-compose** o altri script già presenti in GUIDO per includere TOAD come servizio aggiuntivo all'interno dell'orchestrazione dell'ambiente; • eventuale modifica dei moduli di **comunicazione tra GUIDO e TOAD** per garantire la corretta interazione tra i container e la gestione degli indirizzi e delle porte esposte; L'intervento ha un impatto infrastrutturale significativo ma limitato nella logica applicativa, e rappresenta un prerequisito essenziale per garantire la scalabilità e la manutenibilità del sistema integrato.

Tabella 4.4: Change Request 4: Dockerizzazione del backend TOAD

4.5 Mockup

Questa sezione illustra l'interfaccia grafica prevista per **GUIDO** con l'integrazione di **TOAD**. La GUI è stata progettata per semplificare l'interazione dell'utente con le funzionalità di analisi offerte da TOAD e rendere immediata la consultazione dei risultati.

4.5.1 Schermata di inserimento dati

La Figura 4.1 mostra la sezione denominata *Community Inspector*, che costituisce l'ingresso principale alle funzionalità di TOAD integrate in GUIDO. In questa schermata l'utente può:

- inserire il link alla repository da analizzare;
- specificare la data di riferimento per l'analisi: il sistema utilizzerà tale data per recuperare i commit dei tre mesi antecedenti, come previsto dal funzionamento di TOAD.

Questa interfaccia è concepita per essere intuitiva e guidare l'utente nell'avvio dell'analisi, senza necessità di interazioni tecniche dirette con TOAD.

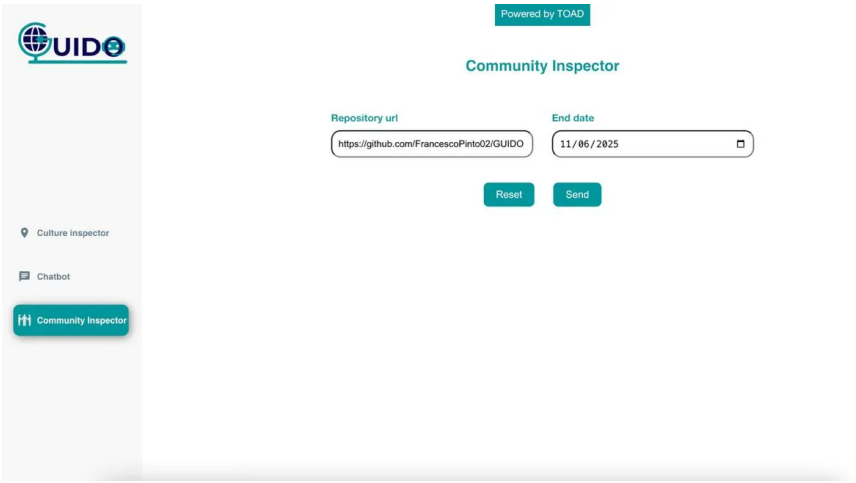

Figura 4.1: Mockup dell'interfaccia Community Inspector prima dell'avvio dell'analisi

4.5.2 Schermata di visualizzazione dei risultati

La Figura 4.2 mostra la schermata prevista per la visualizzazione dei risultati dell'analisi. L'interfaccia è suddivisa in due sezioni principali:

- sulla sinistra vengono presentate le metriche calcolate da TOAD relative alla community della repository analizzata;
- sulla destra è visualizzato un grafo interattivo che rappresenta le relazioni tra i membri della community, facilitando la comprensione delle dinamiche organizzative.

Questa disposizione consente all'utente di esplorare in modo combinato i dati numerici e la rappresentazione visuale della struttura sociale del progetto.

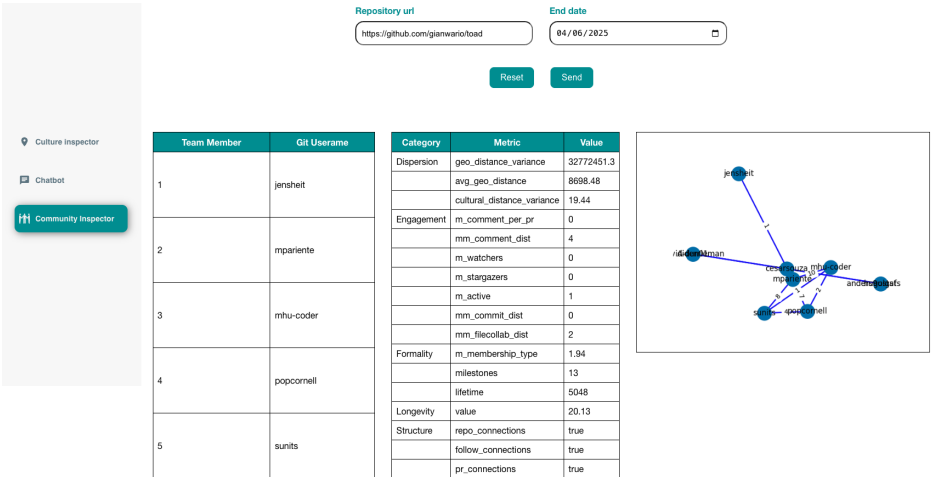

Figura 4.2: Mockup della visualizzazione dei risultati: metriche e grafo della community

Bibliografia

[1] G. Voria, V. Pentangelo, A. D. Porta, S. Lambiase, G. Catolino, F. Palomba, and F. Ferrucci, "Community smell detection and refactoring in slack: The cadocs project," in *2022 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2022, pp. 469–473. (Citato a pagina 2)