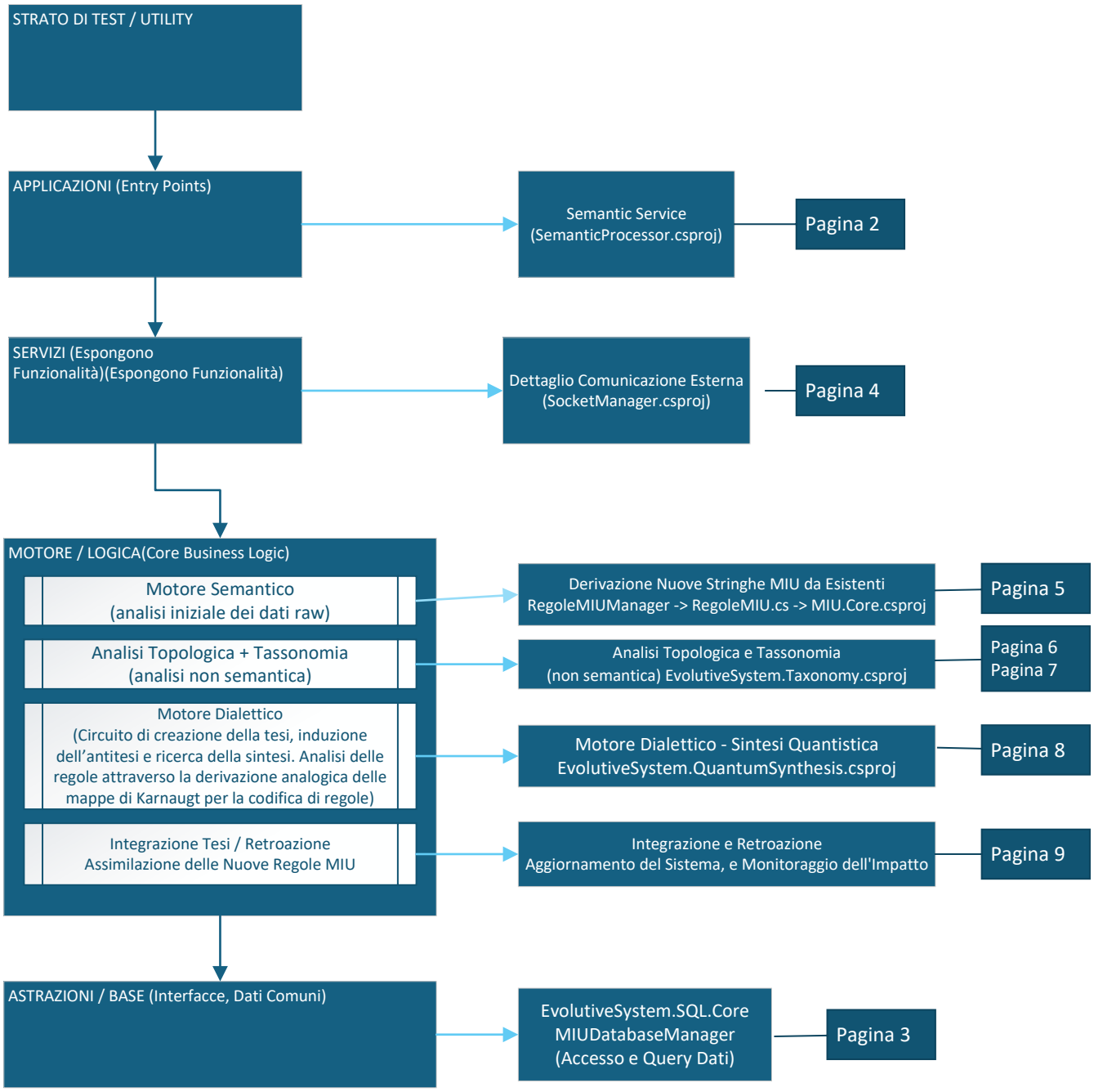
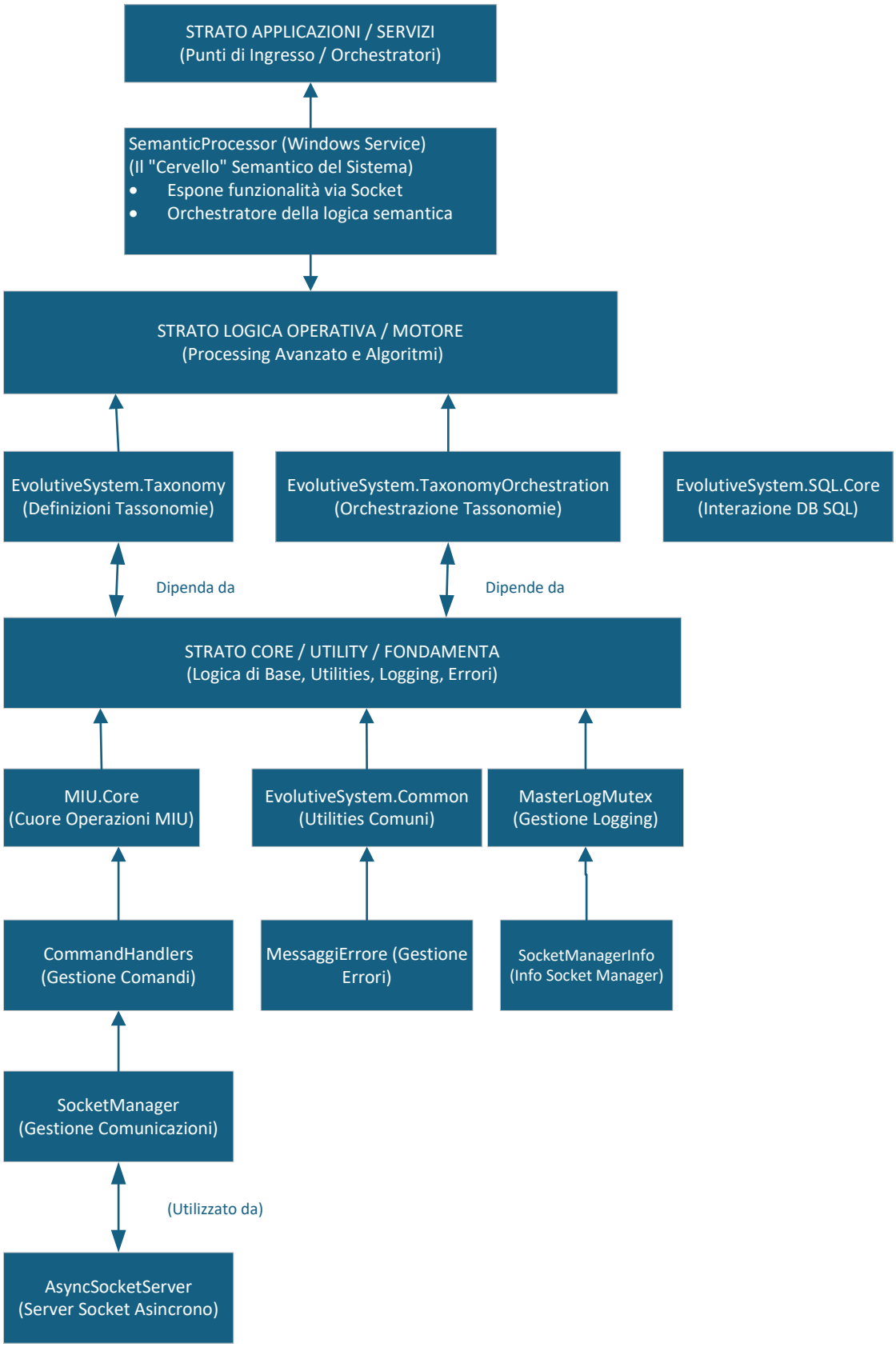
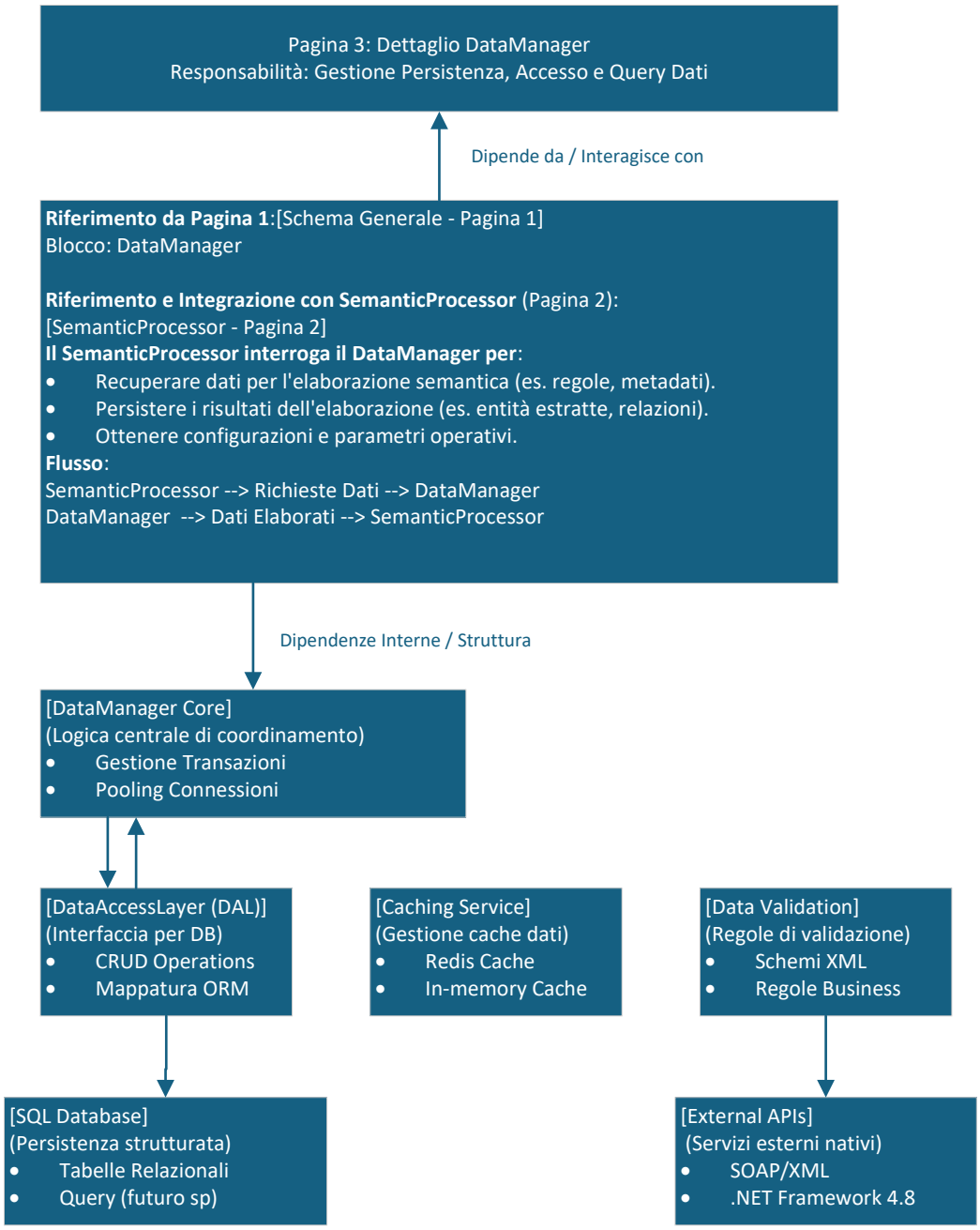
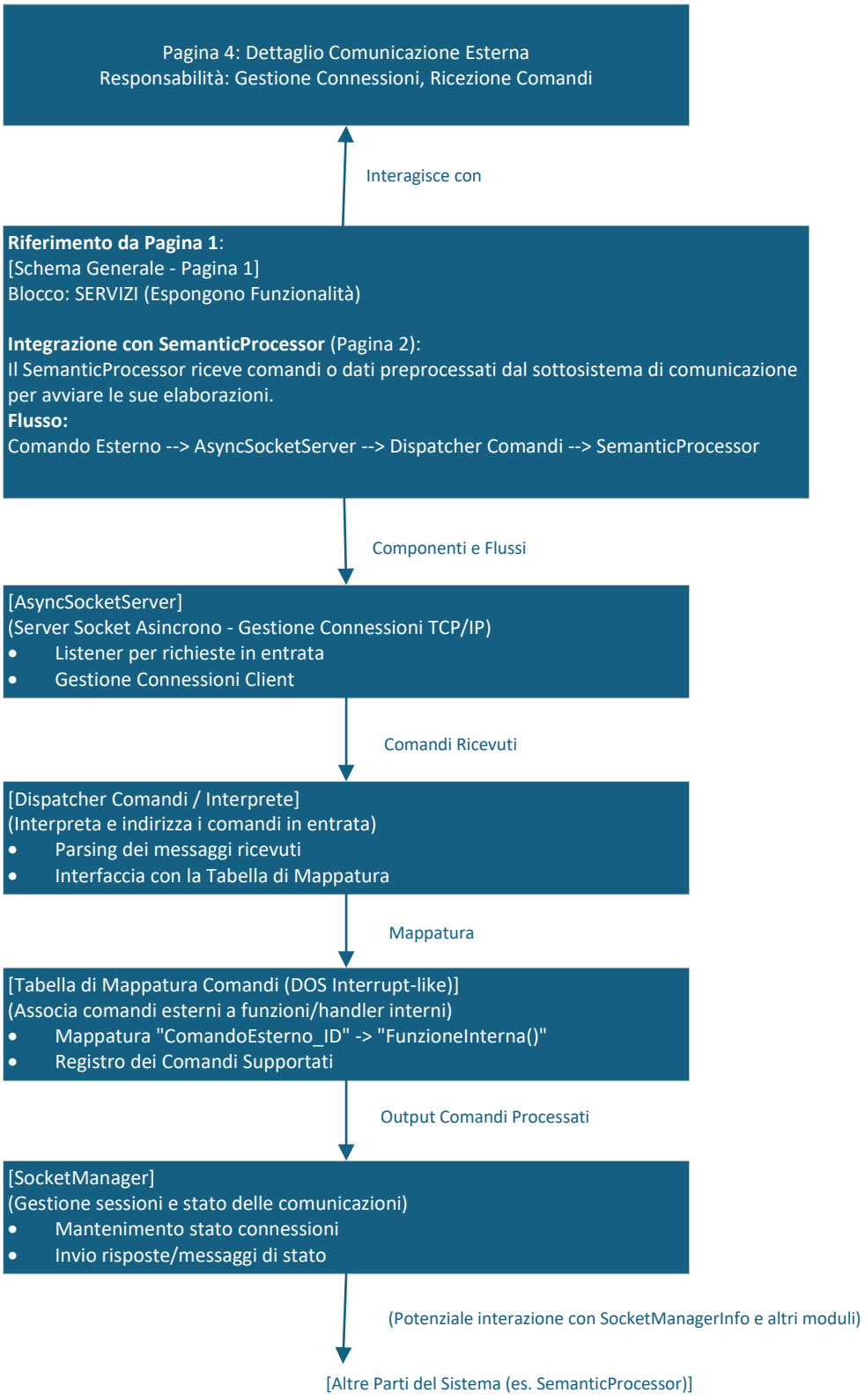

STRATO DI TEST / UTILITY
APPLICAZIONI (Entry Points)
Semantic Service
(SemanticProcessor.csproj)
Pagina 2
SERVIZI (Espongono
Funzionalità)(Espongono Funzionalità)
Dettaglio Comunicazione Esterna
(SocketManager.csproj)
Pagina 4
MOTORE / LOGICA(Core Business Logic)
Motore Semantico
(analisi iniziale dei dati raw)
Derivazione Nuove Stringhe MIU da Esistenti
RegoleMIUManager -> RegoleMIU.cs -> MIU.Core.csproj
Pagina 5
Analisi Topologica + Tassonomia
(analisi non semantica)
Analisi Topologica e Tassonomia
(non semantica) EvolutiveSystem.Taxonomy.csproj
Pagina 6
Pagina 7
Motore Dialettico
(Circuito di creazione della tesi, induzione dell'antitesi e ricerca della sintesi. Analisi delle regole attraverso la derivazione analogica delle mappe di Karnaugt per la codifica di regole)
Motore Dialettico - Sintesi Quantistica
EvolutiveSystem.QuantumSynthesis.csproj
Pagina 8
Integrazione Tesi / Retroazione
Assimilazione delle Nuove Regole MIU
Integrazione e Retroazione
Aggiornamento del Sistema, e Monitoraggio dell'Impatto
Pagina 9
ASTRAZIONI / BASE (Interfacce, Dati Comuni)
EvolutiveSystem.SQL.Core
MIUDatabaseManager
(Accesso e Query Dati)
Pagina 3


STRATO APPLICAZIONI / SERVIZI
(Punti di Ingresso / Orchestratori)
SemanticProcessor (Windows Service)
(Il "Cervello" Semantico del Sistema)
• Espone funzionalità via Socket
• Orchestratore della logica semantica
STRATO LOGICA OPERATIVA / MOTORE
(Processing Avanzato e Algoritmi)
EvolutiveSystem.Taxonomy
(Definizioni Tassonomie)
EvolutiveSystem.TaxonomyOrchestration
(Orchestrazione Tassonomie)
EvolutiveSystem.SQL.Core
(Interazione DB SQL)
Dipenda da
Dipende da
STRATO CORE / UTILITY / FONDAMENTA
(Logica di Base, Utilities, Logging, Errori)
MIU.Core
(Cuore Operazioni MIU)
EvolutiveSystem.Common
(Utilities Comuni)
MasterLogMutex
(Gestione Logging)
CommandHandlers
(Gestione Comandi)
MessaggiErrore (Gestione
Errori)
SocketManagerInfo
(Info Socket Manager)
SocketManager
(Gestione Comunicazioni)
(Utilizzato da)
AsyncSocketServer
(Server Socket Asincrono)


Pagina 3: Dettaglio DataManager
Responsabilità: Gestione Persistenza, Accesso e Query Dati
Dipende da / Interagisce con
Riferimento da Pagina 1:[Schema Generale - Pagina 1]
Blocco: DataManager
Riferimento e Integrazione con SemanticProcessor (Pagina 2):
[SemanticProcessor - Pagina 2]
Il SemanticProcessor interroga il DataManager per:
Recuperare dati per l'elaborazione semantica (es. regole, metadati).
Persistere i risultati dell'elaborazione (es. entità estratte, relazioni).
Ottenere configurazioni e parametri operativi.
Flusso:
SemanticProcessor --> Richieste Dati --> DataManager
DataManager --> Dati Elaborati --> SemanticProcessor
Dipendenze Interne / Struttura
[DataManager Core]
(Logica centrale di coordinamento)
Gestione Transazioni
Pooling Connessioni
[DataAccessLayer (DAL)]
(Interfaccia per DB)
CRUD Operations
Mappatura ORM
[Caching Service]
(Gestione cache dati)
Redis Cache
In-memory Cache
[Data Validation]
(Regole di validazione)
Schemi XML
Regole Business
[SQL Database]
(Persistenza strutturata)
Tabelle Relazionali
Query (futuro sp)
[External APIs]
(Servizi esterni nativi)
SOAP/XML
.NET Framework 4.8


Pagina 4: Dettaglio Comunicazione Esterna
Responsabilità: Gestione Connessioni, Ricezione Comandi
Interagisce con
Riferimento da Pagina 1:
[Schema Generale - Pagina 1]
Blocco: SERVIZI (Espongono Funzionalità)
Integrazione con SemanticProcessor (Pagina 2):
Il SemanticProcessor riceve comandi o dati preprocessati dal sottosistema di comunicazione per avviare le sue elaborazioni.
Flusso:
Comando Esterno --> AsyncSocketServer --> Dispatcher Comandi --> SemanticProcessor
Componenti e Flussi
[AsyncSocketServer]
(Server Socket Asincrono - Gestione Connessioni TCP/IP)
Listener per richieste in entrata
Gestione Connessioni Client
Comandi Ricevuti
[Dispatcher Comandi / Interprete]
(Interpreta e indirizza i comandi in entrata)
Parsing dei messaggi ricevuti
Interfaccia con la Tabella di Mappatura
Mappatura
[Tabella di Mappatura Comandi (DOS Interrupt-like)]
(Associa comandi esterni a funzioni/handler interni)
Mappatura "ComandoEsterno_ID" -> "FunzioneInterna()"
Registro dei Comandi Supportati
Output Comandi Processati
[SocketManager]
(Gestione sessioni e stato delle comunicazioni)
Mantenimento stato connessioni
Invio risposte/messaggi di stato
(Potenziale interazione con SocketManagerInfo e altri moduli)
[Altre Parti del Sistema (es. SemanticProcessor)]

Pagina 5: Derivazione Stringhe MIU
Responsabilità: Derivazione Nuove Stringhe MIU da Esistenti

Input: Stringhe MIU esistenti nel database

Riferimento da Pagina 1:
[Schema Generale - Pagina 1]
Blocco: Motore Semantico

Interazione Principale per l'Input: DataManager (Pagina 3)

- La logica di derivazione legge le stringhe MIU esistenti dalla tabella `MIU_Searches` (e altre tabelle pertinenti come statistiche di regole/transizioni) tramite `MIUDatabaseManager` in `EvolutiveSystem.SQL.Core.csproj`.
- Questo fornisce i "semi" su cui si basano gli algoritmi BFS/DFS per derivare nuove stringhe.

Flusso Input:
Database (SQLite) --- (tramite EvolutiveSystem.SQL.Core) ---> DataManager --- (tramite IMIUDataManager / IMIURepository) ---> Motore di Derivazione MIU

Persistenza degli Esiti (del Motore Semantico in generale):

- Il `SemanticProcessorService` (il "cervello centrale" di Pagina 2) orchestra l'elaborazione e persiste i suoi risultati (come regole, statistiche, anomalie, stati MIU) nel database SQLite utilizzando il`MIUDatabaseManager` in `EvolutiveSystem.SQL.Core.csproj`.
- Questa operazione non è diretta derivazione, ma persistenza dell'elaborazione generale del "Motore Semantico".

Output per il Prossimo Strato:
Fornisce le Stringhe MIU derivate e arricchite (risultato di BFS/DFS)
al blocco "Analisi Topologica + Tassonomia (analisi non semantica)"
(come definito in Pagina 1 e dettagliato in EvolutiveSystem.Taxonomy.cs

Componenti e Flussi Interni

[Input/Recupero Stringhe MIU Esistenti]
(Recupera le stringhe MIU dal database tramite DataManager)

- Query su tabella MIU_Searches
- Caricamento in memoria delle stringhe per l'elaborazione proj).

Stringhe MIU Caricate

[Motore di Derivazione MIU]
(Classe: RegoleMIU.cs in MIU.Core.csproj)
(Applica logiche combinatoriali per derivare nuove stringhe MIU)

- Algoritmi di ricerca: Depth-First Search (DFS)
- Algoritmi di ricerca: Breadth-First Search (BFS) (versione "Intelligent" con coda a priorità e euristiche)
- Logica di selezione algoritmo (TrovaDerivazioneAutomatica) basata su euristiche (es. lunghezza stringhe).
- Utilizzo di CurrentTransitionStatistics e CurrentRuleStatistics per guidare l'esplorazione.
- Potatura (pruning) tramite MAX_STRING_LENGTH e penalità lunghezza stringa.
- Tracciamento nodi visitati (HashSet<string> visitedStandard).
- Interazioni chiave: UpsertMIUState (persistenza stati), pubblicazione eventi (OnSolutionFound, OnRuleApplied, OnNewMiuStringDiscoveredInternal).

Nuove Stringhe MIU Derivate e Arricchite

[Output / Interfaccia per Elaborazione Successiva](
Prepara le stringhe MIU derivate per l'Analisi Topologica)

- Potenziale persistenza delle stringhe MIU derivate (tramite MIU.Core che usa IMIUDataManager).
- Interfaccia per il blocco "Analisi Topologica + Tassonomia"

Pagina 6: EvolutiveSystem.Taxonomy.csproj

Responsabilità: Analisi Topologica e Tassonomia,
Identificazione Inefficienze e Gap (Non Semantica)

Input: Nuove Stringhe MIU Derivate e Arricchite,Topologia delle Derivazioni, Regole MIU Esistenti

Riferimento da Pagina 1:
[Schema Generale - Pagina 1]
Blocco: Analisi Topologica + Tassonomia (analisi non semantica)

Interazione Principale per l'Input: EvolutiveSystem.MIUExplorer.csproj

- Il **`MIUExplorer`** (il motore di derivazione) pubblica eventi come:
- `NewMiuStringDiscoveredEventArgs`: Nuove stringhe MIU derivate.
- `SolutionFoundEventArgs`: Percorsi di derivazione riusciti (e topologia).
- Questi eventi forniscono al **`Taxonomy`** le stringhe e le relazioni di derivazione per costruire la mappa topologica.

Flusso Input:
`MIUExplorer` --- (Event Bus: `NewMiuStringDiscoveredEvent`, `SolutionFoundEvent`) ---> `Taxonomy`

Output per il Prossimo Strato:

- Informazioni strutturate sui **"gap"** (stringhe non derivabili, aree inesplorate)
- Informazioni sulle **"inefficienze"** (regole poco usate, percorsi ridondanti)
- Aggiornamenti sullo stato della tassonomia e statistiche sulle regole.
- Fornisce questi dati al modulo **`EvolutiveSystem.PetriNet.csproj`** (e indirettamente a `TaxonomyOrchestration`).

Componenti e Flussi Interni

[Input Listener / Consolidatore Dati]
(Sottoscrive eventi dall'Event Bus e raccoglie dati

- Ascolta `NewMiuStringDiscoveredEvent` per aggiungere nuove stringhe
- Ascolta `SolutionFoundEvent` per mappare le derivazioni e le regol
- Utilizza `IMIUDataManager` (via `EvolutiveSystem.SQL.Core`) per recuperare le regole attuali e le statistiche persistenti.

Dataset Aggiornato di Stringhe, Regole, Topologia

[Topological Analyzer]
(Analizza il grafo di derivazione delle stringhe MIU)

- **Costruzione del Grafo:** Mappa le relazioni "da stringa X a stringa Y tramite regola R".
- **Identificazione Nodi Isolari/Morti:** Trova stringhe MIU che non possono essere derivate da alcuna regola esistente (i "campioncini" per Karnaugh).
- **Identificazione Percorsi Inefficienti:** Rileva derivazioni lunghe o ridondanti, o regole che portano a stati senza uscita.
- **Metriche di Copertura:** Calcola la percentuale di spazio MIU coperto.
- Interazioni chiave: Fornisce la topologia analizzata alla Rete di Petri.

Risultati dell'Analisi Topologica (Gap, Inefficienze Strutturali)

[Rule Classifier / Tassonomista]
(Classifica le regole MIU e ne valuta l'efficacia non semantica)

- **Tassonomia Dinamica:** Organizza le regole in categorie (es. per pattern, per effetto sulla lunghezza, per frequenza di utilizzo).
- **Statistiche di Utilizzo:** Traccia quali regole sono applicate più/meno spesso.
- **Identificazione di "Regole Silenti":** Regole esistenti che non sono mai utilizzate o non portano a nuove derivazioni utili.
- **Generazione di `GapPattern` e `InefficiencyPattern`:** Trasforma le scoperte dell'analisi topologica e delle statistiche in oggetti specifici per l'Antitesi.

Feedback strutturato (GapPattern, InefficiencyPattern) per la Diagnostica

Output per il Prossimo Strato (Input per Rete di Petri e Orchestrazione):

- **Topologia aggiornata e Token:** La struttura del grafo di derivazione e i punti di "soffocamento" o "mancanza" (i precursori dei token della Petri Net).
- **Segnalazioni di `GapPattern`:** Indicano specifiche stringhe MIU che non possono essere derivate (i "mintermini analogici" da risolvere).
- **Segnalazioni di `InefficiencyPattern`:** Indicano problemi di performance o ridondanza nel set di regole.

Questi output alimenteranno direttamente il prossimo stadio di analisi:
il monitoraggio dinamico della **Rete di Petri**.

Pagina 7: EvolutiveSystem.TaxonomyOrchestration.csproj
Responsabilità: Orchestrazione Diagnostica,
Identificazione e Pubblicazione dell'Antitesi

Input: Report da Analisi Topologica (Taxonomy), Stato dei Token dalla Rete di Petri, Metriche di Performance del Sistema

Riferimento da Pagina 1:
[Schema Generale - Pagina 1]
Blocco: Motore Dialettico (Circuito di creazione della tesi, induzione...)

Interazione Principale per l'Input:
- **`EvolutiveSystem.Taxonomy.csproj` (Pagina 6):** Fornisce i `GapPattern` e`InefficiencyPattern` derivati dall'analisi statica della topologia.
- **`EvolutiveSystem.PetriNet.csproj`:** Fornisce lo stato dinamico dei "token" e segnalazioni di anomalie o squilibri nel flusso delle derivazioni MIU.

Flusso Input:
`Taxonomy` --- (Report strutturati) ---> `TaxonomyOrchestration`
`PetriNet` --- (Stato dinamico Token) ---> `TaxonomyOrchestration`

Output per il Prossimo Strato:
- Pubblicazione di `AntithesisIdentifiedEvent` (il trigger per la Sintesi).
- Invia i dettagli dell'Antitesi (es. le stringhe MIU non derivabili) al `QuantumSynthesis.RuleCandidateProposer`.
- Fornisce feedback per l'ottimizzazione del `MIUExplorer`.

Componenti e Flussi Interni

[Event Listener / Data Aggregator]
(Sottoscrive eventi da Taxonomy e PetriNet, aggrega dati)
- Monitora costantemente gli output di `Taxonomy` (per gap e inefficienze trutturali) e di `PetriNet` (per anomalie di flusso/token).
- Raccoglie e normalizza le diverse segnalazioni di problemi.
- Utilizza `MasterLogMutex` per registrare le anomalie rilevate.

Segnalazioni Aggregate di Problemi

[Antithesis Identification Engine / PID Feedback Control]
(Analizza le segnalazioni aggregate e determina l'Antitesi)
- **Logica PID Feedforward:** Non solo reagisce ai problemi, ma tenta di anticiparli o di indirizzare la ricerca di soluzioni. Questo è il "controllo" che traduce lo stato attuale (Process Variable) e i "setpoint" (gli obiettivi di copertura/efficienza) in una "Control Variable" che attiverà la sintesi.
- **Confronto con Target:** Valuta la discrepanza tra lo stato attuale del sistema (copertura, efficienza) e gli obiettivi desiderati.
- **Prioritizzazione:** Assegna una priorità alle diverse "antitesi" identificate.
- **Formulazione dell'Antitesi:** Compone un oggetto `AntithesisDefinition` che descrive il problema (es. `GapPattern` specifico, stringhe non derivabili).
- **Classe chiave:** **`AntithesisIdentifier.cs`** (in `EvolutiveSystem.Taxonomy.Antithesis.csproj`).

AntithesisDefinition (Problema Formalizzato)

[Event Publisher / Synthesis Trigger]
(Pubblica l'evento di Antitesi identificata sull'Event Bus)
- Dopo aver formalizzato l'Antitesi, l'Orchestrator pubblica un evento **`AntithesisIdentifiedEvent`** sull'Event Bus.
- Questo evento contiene tutti i dettagli necessari (`AntithesisDefinition`), inclusi i "campioncini" di stringhe non derivabili che il `QuantumSynthesis`dovrà risolvere.
- Questo è il punto di collegamento cruciale con il motore di Sintesi.

`AntithesisIdentifiedEvent` pubblicato

Output per il Prossimo Strato (Input per EvolutiveSystem.QuantumSynthesis.csproj):
- **`AntithesisIdentifiedEvent`**: Contiene la descrizione dettagliata dell'Antitesi (es. le precise stringhe MIU non derivabili o i pattern di inefficienza da risolvere).
- Questo evento sarà ascoltato dal **`QuantumSynthesisOrchestrator`**(nel progetto `EvolutiveSystem.QuantumSynthesis.csproj`), avviando il processo di generazione e valutazione delle regole.

Pagina 8: EvolutiveSystem.QuantumSynthesis.csproj

Responsabilità: Sintesi Quantistica di Nuove Regole MIU, Generazione e Valutazione di Candidati per la Risoluzione dell'Antitesi (Il Cuore del Motore Dialettico)

Input: `AntithesisIdentifiedEvent` (con `AntithesisDefinition` che include **`RuleFailureDetail`**), Regole MIU Esistenti, Set di Test MIU

Riferimento da Pagina 1:
[Schema Generale - Pagina 1]
Blocco: Motore Dialettico (Circuito di creazione della tesi, induzione...)

Interazione Principale per l'Input:

- **`EvolutiveSystem.TaxonomyOrchestration.csproj` (Pagina 7):** Pubblica l'evento `AntithesisIdentifiedEvent` che innesca il processo di sintesi. Questo evento contiene i dettagli dell'dell'**`AntithesisDefinition`**, inclusi:, inclusi:
  - I "campioncini" di stringhe MIU non derivabili o pattern di inefficienza.
  - Una collezione di **`IRuleFailureDetail`** (provenienti dalla diagnostica approfondita di **`EvolutiveSystem.Analog.DeepScan.csproj`**) che descrivono i fallimenti specifici delle regole.

Flusso Input:
TaxonomyOrchestration` --- (`AntithesisIdentifiedEvent` contenente **`IAntithesisWithDiagnosticDetails`**) ---> `QuantumSynthesisOrchestrator`

Persistenza degli Esiti:

- Le regole candidate validate e approvate vengono proposte per la persistenza nel database tramite `EvolutiveSystem.SQL.Core.csproj` e `MIUDatabaseManager`.
- I risultati delle simulazioni e le metriche di valutazione possono essere registrati per l'analisi post-mortem o per il logging via `MasterLogMutex`.

Output per il Prossimo Strato:

- **Nuove Regole MIU Validate:** Regole che hanno superato i test e sono pronte per essere aggiunte al set di regole attivo del sistema.
- Report di Validazione/Efficacia delle regole proposte per il Semantic Engine.

`AntithesisDefinition` (Stringhe MIU bersaglio, **`IRuleFailureDetail`**, ecc.)

[QuantumSynthesisOrchestrator.cs]
(Orchestratore del processo di sintesi)

- **Ascolto Antitesi:** Sottoscrive `AntithesisIdentifiedEvent` e ne estrae l'**`AntithesisDefinition`** (con **`IRuleFailureDetail`** e stringhe MIU bersaglio).
- **Coordinamento:** Dirige il flusso di lavoro tra `RuleCandidateProposer` e RuleCandidateEvaluator`.
- **Gestione Iterazioni:** Supervisiona il ciclo di Proposta-Valutazione fino al raggiungimento di una soluzione soddisfacente o all'esaurimento delle risorse/tentativi.
- Interagisce con `MIU.Core` per accedere alle regole esistenti e al contesto MIU.

`AntithesisDefinition` (Stringhe MIU bersaglio, **`IRuleFailureDetail`**, ecc.)

[IRuleCandidateProposer.cs / RuleCandidateProposer.cs]
(Generatore di regole candidate - La "Mappa di Karnaugh Analogica")

- **Input:** `AntithesisDefinition` (specialmente le stringhe MIU non derivabili e la **collezione di `IRuleFailureDetail`**).
- **Logica "Mappa di Karnaugh Analogica":**
  - Analizza le **`IRuleFailureDetail`** come "mintermini" (o "macchie") in uno "spazio analogico" di derivazione, usando i dati di fallimento (regola fallita, stringa iniziale, ragione del fallimento) per una diagnosi approfondita del problema.
  - Utilizza euristiche e pattern matching per identificare schemi ricorrenti o "aree vuote" nello spazio delle derivazioni che le nuove regole devono "coprire".
  - Propone solo nuove regole MIU (candidate) che potrebbero "coprire" questi fallimenti o migliorare l'efficienza senza modificare le regole esistenti.
  - Potrebbe usare tecniche combinatoriali avanzate o algoritmi evolutivi per generare un set di regole diverse.
- **Output:** Un elenco di `RuleCandidate` (regole MIU proposte).

`RuleCandidate` (Regole MIU proposte)

[IMiuSimulationEnvironment.cs / MiuSimulationEnvironment.cs]
(Ambiente di Simulazione per la Validazione)

- **Ambiente Controllato:** Fornisce un'istanza isolata del motore di derivazione MIU (`MIUExplorer` mock/wrapper) caricato con le regole attuali e le nuove `RuleCandidate`.
- **Simulazione:** Permette al `RuleCandidateEvaluator` di eseguire simulazioni di derivazione MIU utilizzando le regole candidate contro un set di test, incluso il tentativo di risolvere i problemi indicati dai `IRuleFailureDetail`.
- **Metriche:** Raccoglie metriche di performance della simulazione, come:
  - `StimaProfonditaMedia`: La profondità media delle derivazioni per raggiungere le stringhe target.
  - `TargetAntithesisResolutionScore`: Punteggio che indica quanto efficacemente le regole candidate risolvono l'Antitesi.
  - Conflitti, ridondanze, ecc.
- **Output:** Risultati dettagliati della simulazione.

Risultati della Simulazione (Metriche, Risoluzione Antitesi)

[IRuleCandidateEvaluator.cs / RuleCandidateEvaluator.cs]
(Valutatore di regole candidate)

- **Input:** `RuleCandidate` e i risultati di `MiuSimulationEnvironment`.
- **Criteri di Valutazione:**
  - **Validazione Strutturale:** Verifica la correttezza formale delle regole (es. sintassi, coerenza interna).
  - **Validazione Funzionale:** Utilizza `MiuSimulationEnvironment` per verificare se le regole candidate risolvono effettivamente l'Antitesi (es. derivano le stringhe MIU bersaglio e risolvono i fallimenti in **`IRuleFailureDetail`**).
  - **Valutazione di Efficienza:** Analizza le metriche di simulazione (`StimaProfonditaMedia`, ecc.) per determinare se le nuove regole migliorano le performance o introducono ridondanze/conflitti.
- **Filtering/Ranking:** Seleziona le regole più promettenti. Non le modifica, ma scarta quelle non idonee o inefficienti.
- **Output:** `ValidatedRule` (regole pronte per l'approvazione) o feedback per il `Proposer` per una nuova iterazione.

Nuove Regole MIU Validate

Output per il Motore Semantico (Pagina 2):

- Le **nuove regole MIU validate** vengono inviate al `EvolutiveSystem.SemanticProcessorService.csproj` (il "Cervello Centrale").
- Il Semantic Processor si occuperà di integrare queste nuove regole nel sistema, aggiornando il set di regole attivo e potenzialmente innescando nuovi cicli di derivazione.
- Questo chiude il ciclo Dialettico: l'Antitesi ha generato una Nuova Tesi.

Pagina 9: Integrazione e Retroazione (Feedback)

Responsabilità: Assimilazione delle Nuove Regole MIU, Aggiornamento del Sistema, e Monitoraggio dell'Impatto (Chiusura del Ciclo Dialettico e Preparazione per il Successivo)

Input: Nuove Regole MIU Validate da QuantumSynthesis.csproj, Statistiche di Utilizzo del MIUExplorer.

Riferimento da Pagina 1:
[Schema Generale - Pagina 1]
Blocco: Integrazione Tesi / Retroazione

Interazione Principale per l'Input:
- **`EvolutiveSystem.QuantumSynthesis.csproj` (Pagina 8):** Fornisce le **`ValidatedRule`** (le nuove regole MIU che hanno superato i test).

Flusso Input:
`QuantumSynthesis` --- (`ValidatedRule`) ---> `SemanticProcessorService`

Output per il Prossimo Ciclo:
- **Set di Regole MIU Aggiornato:** Il motore di derivazione (`MIUExplorer`) opererà con un set di regole potenziato.
- **Metriche di Performance Aggiornate:** Feedback sul miglioramento per i moduli di diagnosi (Taxonomy, PetriNet).
- **Nuovi Punti di Partenza per l'Antitesi:** Le nuove regole potrebbero rivelare nuove aree di inefficienza o gap, innescando un nuovo ciclo.

Componenti e Flussi Interni

[SemanticProcessorService.cs (Aggiornamento Regole)]
(Componente nel cuore del sistema che gestisce il set di regole)
- **Recezione Regole:** Riceve le `ValidatedRule` dal `QuantumSynthesisOrchestrator`.
- **Persistenza:** Utilizza **`EvolutiveSystem.SQL.Core.csproj`** per salvare le nuove regole nel database (gestite da `IMIUDataManager`).
- **Aggiornamento In-Memory:** Carica le nuove regole nel contesto di esecuzione del `MIUExplorer`, rendendole immediatamente disponibili per nuove derivazioni.
- **Broadcast Aggiornamento:** Potrebbe notificare altri moduli dell'aggiornamento del set di regole.

Regole MIU Aggiornate e Integrate nel Sistema

[MIUExplorer (Riavvio / Riconfigurazione)]
(Il motore di derivazione opera con il nuovo set di regole)
- **Utilizzo Regole:** Inizia a utilizzare le regole appena integrate per le future derivazioni di stringhe MIU.
- **Generazione Nuovi Dati:** Crea nuove stringhe derivate e nuove topologie di derivazione che riflettono l'influenza delle nuove regole.
- **Statistiche Aggiornate:** Genera nuove metriche di performance e di utilizzo delle regole, che saranno poi analizzate dal sistema.

Nuovi Dati, Nuove Topologie, Nuove Statistiche

[Ciclo di Monitoraggio Continuo]
(Taxonomy e PetriNet riprendono l'analisi sul sistema aggiornato)
- **`EvolutiveSystem.Taxonomy.csproj` (Pagina 6):** Rileva le nuove topologie di derivazione e identifica se i "gap" precedenti sono stati colmati o se ne sono emersi di nuovi (la **"nuova Tesi"** viene valutata).
- **`EvolutiveSystem.PetriNet.csproj`:** Monitora il flusso dei token con le nuove regole, verificando se gli squilibri sono stati risolti o se si sono creati nuovi "colli di bottiglia".
- Questo feedback continuo porta all'identificazione di una **nuova Antitesi** (se presente), chiudendo il ciclo e innescando una nuova iterazione del Motore Dialettico.

Rilevamento di Nuove Antitesi (se presenti)

Chiusura del Ciclo Dialettico:
- Le nuove regole (Tesi) sono state generate per risolvere l'Antitesi precedente.
- Vengono integrate nel sistema.
- Il sistema si evolve e la sua nuova configurazione viene nuovamente monitorata e analizzata.