**Documento di Progettazione Architetturale: L'Analisi dei Fallimenti**

**1. Il Problema Iniziale**

Tutto è iniziato con una domanda fondamentale: **"Come fare a capire perché una regola non ha funzionato?"**

L'obiettivo era creare un componente, il RuleCandidateProposer, che non si limitasse a constatare un fallimento, ma ne analizzasse la causa. L'idea era di costruire una sorta di "microscopio logico" capace di identificare il "buco" che ha impedito a un processo di derivazione di raggiungere l'obiettivo.

**2. La Logica del "Microscopio"**

La soluzione concettuale, nata dalla nostra discussione, è stata l'applicazione di una logica ricorsiva per isolare il problema:

1. **Individuare il "Buco"**: Confrontando la stringa che ha fallito (sourceString) con quella attesa (targetString), si individua la porzione di stringa non corrispondente (il mintermine).
2. **"Distillare" il Mintermine**: Questo mintermine viene poi sottoposto a un'analisi. Se una qualsiasi delle regole esistenti è applicabile a questo pattern, la si applica.
3. **Iterazione Ricorsiva**: L'applicazione di una regola produce un nuovo pattern, solitamente più semplice. Il processo di analisi ricorsiva continua su questo nuovo pattern.
4. **Caso Base**: Il processo si conclude quando nessuna delle regole esistenti può essere applicata al pattern. Quel pattern finale, il più piccolo e non coperto, è il "mintermine" di fallimento che il sistema deve risolvere.

Questa logica è il cuore del metodo AnalyzeFailures.

**3. Le Sfide Tecniche e la Soluzione**

Durante l'implementazione, sono emerse due sfide chiave:

1. **Incompatibilità di Tipi**: La logica del "microscopio" era chiara, ma la sua implementazione richiedeva di far comunicare il RuleCandidateProposer, che lavora con **RegolaMIU**, con il **MIURulesEngine**, che accetta solo oggetti di tipo **MIURule**.
2. **Evitare Modifiche Invasive**: Modificare la classe RegolaMIU per farle implementare l'interfaccia MIURule avrebbe causato un effetto a cascata in tutto il progetto.

Per superare questi ostacoli, abbiamo adottato il pattern di progettazione **Adapter**. Questo ci ha permesso di creare un "ponte" software:

* È stata creata la classe **RegolaMIUAdapter**.
* Questa classe incapsula un oggetto RegolaMIU e, allo stesso tempo, implementa l'interfaccia MIURule.
* Ha fornito le implementazioni necessarie dei metodi IsApplicable e Apply basate sull'uso di espressioni regolari.

Questo approccio ha consentito di mantenere intatto il tuo modello di dati esistente e, allo stesso tempo, di far funzionare perfettamente l'engine di regole.

**4. L'Implementazione Finale**

Il risultato finale è un codice che unisce l'idea della distillazione ricorsiva con un'architettura robusta e pulita:

* Il modulo RuleCandidateProposer ora contiene il metodo **AnalyzeFailures** che implementa la logica del "microscopio".
* Il codice crea un'istanza di **RegolaMIUAdapter** all'interno del suo ciclo foreach per ogni regola esistente.
* Questa istanza viene poi passata al **MIURulesEngine**, che si occupa dell'effettiva applicazione della regola.
* La ricorsione continua finché non viene trovato il mintermine di fallimento.

Questo documento cattura l'essenza del nostro percorso: dalla tua intuizione iniziale sulla logica di analisi fino alla soluzione tecnica che l'ha resa possibile. Spero che sia utile per il tuo progetto.