Contents

Problem Space Taxonomy per Architetture Agentiche

Overview

Non esiste un'architettura "one-size-fits-all". La scelta architettuale dipende criticamente dalle caratteristiche del problema da risolvere.

Questa taxonomy classifica problemi agentici lungo **5 dimensioni fondamentali**, permettendo selezione sistematica di pattern architetturali appropriati.

Le 5 Dimensioni di Classificazione

Dimensione 1: Complessità Decisionale

Metrica: Numero di step di ragionamento richiesti per soluzione

```
Level 0: REATTIVO
- Decisione istantanea da input
- Nessuna decomposizione richiesta
- Esempio: "Qual è il capitale della Francia?"

Level 1: SEMPLICE
- 1-3 step di ragionamento
- Decomposizione minima
- Esempio: "Riassumi questo articolo"

Level 2: INTERMEDIO
- 4-10 step di ragionamento
- Decomposizione moderata
- Backtracking occasionale
- Esempio: "Scrivi un programma che calcola numeri primi"

Level 3: COMPLESSO
- 10-50 step di ragionamento
- Decomposizione gerarchica necessaria
- Backtracking frequente
- Planning esplicito benefico
- Esempio: "Progetta un'architettura software scalabile"

Level 4: MOLTO COMPLESSO
- 50+ step di ragionamento
- Decomposizione multi-livello
- Esplorazione spazio soluzioni ampio
- Planning e replanning essenziali
- Esempio: "Ricerca scientifica su nuovo problema"
```

Dimensione 2: Requisiti di Affidabilità

Metrica: Conseguenze di fallimento

```
Class A: BEST-EFFORT
- Fallimento accettabile
- Retry disponibile
- Nessun costo critico
- Esempio: Generazione contenuto creativo

Class B: ALTA AFFIDABILITÀ
- Fallimento indesiderabile ma gestibile
- Recovery possibile
```

```
- Costo moderato di fallimento
- Esempio: Assistente customer service

Class C: MISSION-CRITICAL
- Fallimento dannoso
- Recovery difficile
- Alto costo di fallimento
- Esempio: Assistente medico diagnostico

Class D: SAFETY-CRITICAL
- Fallimento catastrofico
- Rischio per vita/safety
- Recovery impossibile
- Esempio: Controllo sistemi autonomi in ambienti pericolosi
```

Dimensione 3: Vincoli Temporali

Metrica: Latenza massima accettabile

```
Real-Time: <100ms
- Decisioni istantanee richieste
- Nessun tempo per elaborazione complessa
- Esempio: Controllo robotico real-time

Interactive: 100ms-2s
- Risposta percepita come istantanea
- Ragionamento limitato possibile
- Esempio: Chatbot conversazionale

Responsive: 2s-30s
- Utente aspetta attivamente
- Ragionamento moderato accettabile
- Esempio: Coding assistant

Batch: 30s-10min
- Utente fa altro nel frattempo
- Ragionamento esteso possibile
- Esempio: Analisi codice complessa

Offline: >10min
- Nessuna aspettativa immediata
- Elaborazione estensiva accettabile
- Esempio: Ricerca scientifica automatizzata
```

Dimensione 4: Necessità di Tracciabilità

Metrica: Requisiti di spiegabilità e audit

```
Level 0: NESSUNA
```

- Black box accettabile
- Nessun audit requirement
- Esempio: Generazione testo creativo

Level 1: BASIC LOGGING
- Log di alto livello sufficiente
- Debugging capability base
- Esempio: Task automation interno

Level 2: DECISION TRACKING
- Rationale decisioni documentato
- Replay capability richiesta
- Esempio: Business process automation

Level 3: FULL AUDITABILITY
- Ogni decisione tracciata e giustificata
- Compliance requirements
- Deterministic replay essenziale
- Esempio: Sistemi finanziari

Level 4: FORMAL VERIFICATION
- Proof of correctness richiesta
- Proprietà formali verificabili
- Esempio: Safety-critical systems

Dimensione 5: Criteri di Safety

Metrica: Necessità di safety bounds

UNCONSTRAINED
- Nessun bound richiesto
- Comportamento emergente accettabile
- Esempio: Brainstorming creativo

SOFT CONSTRAINTS
- Linee guida preferenziali
- Violazioni non critiche
- Esempio: Content generation con stile

HARD CONSTRAINTS
- Bounds non violabili
- Validation richiesta
- Esempio: Generazione codice con security requirements

SAFETY-CRITICAL CONSTRAINTS
- Violazione = fallimento catastrofico
- Formal verification necessaria
- Esempio: Medical dosage calculation

Classi di Problemi Principali

Combinando le 5 dimensioni, emergono classi distinte:

Classe A: Task Semplici e Variabili

Profilo: - Complessità: Level 0-1 - Affidabilità: Best-effort - Latenza: Interactive - Tracciabilità: Nessuna-Basic - Safety: Unconstrained-Soft

Caratteristiche: - Alto volume, bassa criticità - Variabilità alta di input - Fallimento tollerabile - Focus su flessibilità e generalizzazione

Esempi: - Chatbot generale - Content generation - Summarization - Simple Q&A

Requisiti Architetturali: - [OK] Minimalismo (overhead basso) - [OK] Flessibilità (generalizzazione) - [NO] Non serve planning esplicito - [NO] Non serve extensive logging - [NO] Non servono safety bounds rigidi

Classe B: Task Complessi con Planning

Profilo: - Complessità: Level 2-3 - Affidabilità: Alta - Latenza: Responsive-Batch - Tracciabilità: Decision tracking - Safety: Soft-Hard constraints

Caratteristiche: - Decomposizione necessaria - Multi-step reasoning - Success rate importante - Debugging capability richiesta

Esempi: - Software development assistant - Data analysis automation - Complex research tasks - Multi-step workflows

Requisiti Architetturali: - [OK] Goal decomposition esplicita - [OK] Memory per subtasks - [OK] Decision logging - [OK] Error recovery mechanisms - [NO] Non serve formal verification (ma validation)

Classe C: Task Safety-Critical

Profilo: - Complessità: Varia (anche semplice può essere critical) - Affidabilità: Mission/Safety-critical - Latenza: Varia - Tracciabilità: Full auditability - Safety: Hard-Safety-critical constraints

Caratteristiche: - Fallimento non accettabile - Ogni decisione deve essere tracciata - Bounds rigorosi - Human-in-the-loop spesso richiesto

Esempi: - Medical diagnosis assistant - Financial advisory systems - Legal document analysis - Autonomous vehicle decision-making

Requisiti Architetturali: - [OK] Controllo esplicito (non emergenza pura) - [OK] Validation layers multiple - [OK] Complete audit trail - [OK] Safety bounds enforcement - [OK] Fallback to human - [NO] Emergenza non controllata inaccettabile

Classe D: Task Real-Time

Profilo: - Complessità: Level 0-1 (forced by latency) - Affidabilità: Varia - Latenza: Real-time (<100ms) - Tracciabilità: Varia - Safety: Varia

Caratteristiche: - Vincolo temporale dominante - Decisioni devono essere rapide - Complessità limitata da latenza - Spesso reactive più che deliberative

Esempi: - Robotics control - Real-time gaming AI - High-frequency trading decisions - Voice assistant wake-word detection

Requisiti Architetturali: - [OK] Architettura reattiva - [OK] Pre-computation quando possibile - [OK] Cached patterns/policies - [NO] Non possibile extensive reasoning online - [NO] LLM generation troppo lenta (spesso)

Classe E: Task Multi-Agent Collaborativi

Profilo: - Complessità: Level 3-4 - Affidabilità: Alta - Latenza: Batch-Offline - Tracciabilità: Decision tracking - Safety: Soft-Hard constraints

Caratteristiche: - Decomponibile in sub-task paralleli - Coordinazione richiesta - Specializzazione benefica - Comunicazione inter-agent necessaria

Esempi: - Software development team (design, code, test, review) - Research collaboration (literature review, experiments, analysis) - Complex simulations - Distributed problem solving

Requisiti Architetturali: - [OK] Agent coordination mechanism - [OK] Message passing infrastructure - [OK] Task distribution strategy - [OK] Conflict resolution - [NO] Single-agent pattern insufficiente

Decision Tree: Da Problema a Classe

```
START
|
+- Latenza <100ms richiesta?
|  +- YES -> Classe D (Real-Time)
|  +- NO -> Continua
|
+- Fallimento catastrofico?
|  +- YES -> Classe C (Safety-Critical)
|  +- NO -> Continua
|
+- Decomponibile in sub-task paralleli da agents specializzati?
|  +- YES, benefico -> Classe E (Multi-Agent)
|  +- NO o non necessario -> Continua
|
+- Complessità decisionale Level 0-1?
|  +- YES -> Classe A (Simple & Variable)
|  +- NO (Level 2+) -> Classe B (Complex Planning)
```

Matrice di Mapping: Classe -> Requisiti

Requisito	Classe A	Classe B	Classe C	Classe D	Classe E
Execution Loop	Semplice	Planning-aware	Verified	Reactive	Coordinated
Memory System	Context-only	Episodic	Full audit trail	Minimal	Shared
Tool Interface	Basic	Extended	Validated	Pre-compiled	Distributed
Observability	Minimal	Moderate	Complete	Performance-focused	Multi-agent tracing
Error Handling	Retry	Recovery	Prevention	Graceful degradation	Coordination-aware
Safety Bounds	Soft	Moderate	Rigid	Domain-specific	Coordinated

Esempi Concreti per Classe

Classe A Examples

Email Reply Suggestion - Input: Email ricevuto - Output: Bozza risposta - Profilo: Simple, best-effort, interactive, no trace, unconstrained

Social Media Post Generation - Input: Topic + style - Output: Post content - Profilo: Simple, best-effort, responsive, basic log, soft constraints

Classe B Examples

Code Refactoring Assistant - Input: Codebase + refactoring goal - Output: Refactored code + explanation - Profilo: Complex, high reliability, batch, decision tracking, hard constraints (must compile)

Research Literature Analysis - Input: Research question + papers - Output: Synthesis + gaps - Profilo: Very complex, high reliability, offline, decision tracking, soft constraints

Classe C Examples

Medical Diagnosis Support - Input: Symptoms + patient history - Output: Diagnosis suggestions + confidence + reasoning - Profilo: Intermediate, safety-critical, responsive, full audit, safety-critical constraints

Financial Portfolio Advisor - Input: Portfolio + market + goals - Output: Recommendations + risk analysis - Profilo: Complex, mission-critical, batch, full audit, hard constraints

Classe D Examples

Robotic Grasping Decision - Input: Visual + tactile sensors - Output: Grasp parameters - Profilo: Reactive, high reliability, real-time, performance log, hard constraints

Voice Command Recognition - Input: Audio stream - Output: Command classification - Profilo: Reactive, best-effort, real-time, minimal trace, unconstrained

Classe E Examples

Software Development Team - Agents: Architect, Developer, Tester, Reviewer - Input: Feature requirements - Output: Implemented & tested feature - Profilo: Very complex, high reliability, offline, decision tracking, hard constraints

Scientific Research Collaboration - Agents: Literature Reviewer, Experiment Designer, Data Analyst, Paper Writer - Input: Research question - Output: Research paper - Profilo: Very complex, high reliability, offline, decision tracking, soft constraints

Anti-Patterns: Misclassificazione Comune

Anti-Pattern 1: Over-Engineering Classe A

Errore: Usare architettura Classe C per problema Classe A

Esempio: - Task: Chatbot generale - Architettura scelta: Full planning, extensive logging, safety verification - Problema: Overhead » beneficio, rigidità dannosa

Correzione: Riconoscere che best-effort è sufficiente

Anti-Pattern 2: Under-Engineering Classe C

Errore: Usare architettura Classe A per problema Classe C

Esempio: - Task: Medical diagnosis - Architettura scelta: Simple LLM call, no verification - Problema: Safety risk inaccettabile

Correzione: Aggiungere controlli richiesti dalla criticità

Anti-Pattern 3: Forcing Real-Time su Task Complesso

Errore: Requisito latenza incompatibile con complessità

Esempio: - Task: Decisione investimento complessa - Requisito: <100ms - Problema: Impossibile ragionamento adeguato in tempo

Correzione: Rilassare vincolo temporale O pre-computare O semplificare decisione

Uso della Taxonomy

Per Architetti

1. **Classificare** problema usando 5 dimensioni

2. **Identificare** classe primaria
3. **Consultare** pattern raccomandato (Documento 03)
4. **Adattare** per specificità contesto
5. **Validare** con metriche (Documento 04)

Per Implementatori

1. Ricevere classificazione da architetto
2. Selezionare template per classe
3. Implementare pattern base
4. Customizzare per requisiti specifici
5. Testare contro metriche di classe

Per Ricercatori

1. Usare taxonomy per strutturare evaluation
2. Identificare quale classe benchmark targeta
3. Comparare solo architetture appropriate per classe
4. Riconoscere trade-off specifici di classe

Limiti di Questa Taxonomy

Limitazioni Riconosciute

1. **Boundaries Sfumati**: Alcuni problemi sono al confine tra classi
2. **Multi-Dimensionalità**: Problema può richiedere mix di pattern
3. **Contesto-Dipendenza**: Stessa task può essere Classe diversa in contesti diversi
4. **Evoluzione**: Nuove classi possono emergere con nuovi LLM

Quando Taxonomy Non Basta

- Problemi ultra-specifici con requisiti unici
- Contesti regolatori particolari
- Vincoli risorse estreme
- Combinazioni inusuali di requisiti

In questi casi: usare taxonomy come starting point, customizzare estensivamente.

Next Steps

Dopo aver classificato il problema:

-> **02-design-dimensions.md**: Comprendere trade-off specifici

-> **03-architecture-patterns.md**: Selezionare pattern per classe

-> **04-evaluation-framework.md**: Metriche per validazione

Contribution: Questa taxonomy è la prima classificazione sistematica di problemi agentici basata su requisiti architetturali multi-dimensionali, non solo complessità task.