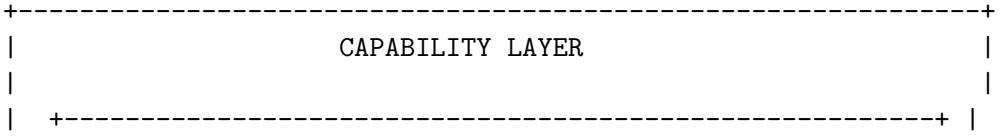
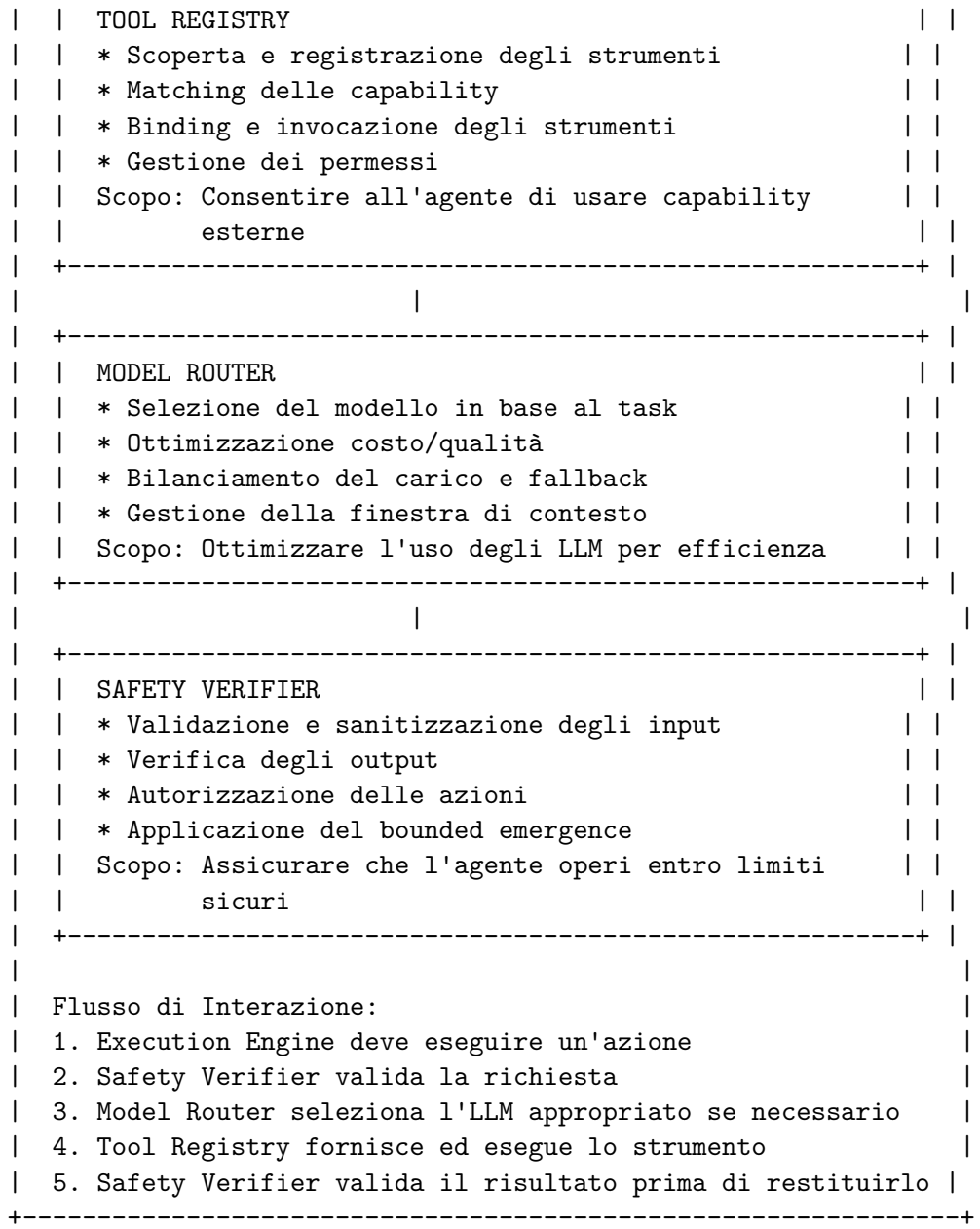
Contents

Capability Layer: Strumenti, Modelli e Sicurezza

Panoramica

Il Capability Layer fornisce le interfacce concrete attraverso cui l'agente interagisce col mondo: strumenti per azioni esterne, modelli per reasoning, e sistemi di sicurezza per bounded emergence. È il layer che traduce intenzioni high-level in azioni concrete sicure.





1. Tool Registry

1.1 Scopo e Responsabilità

Funzione Centrale: Gestire catalogo di capabilities esterne (tools) disponibili all'agente e mediare loro utilizzo.

Caratteristiche: - **Estensibile**: Nuovi tool facilmente aggiungibili - **Scopribile**: Agent può trovare tool per capability richiesta - **Sicuro**: Permission-based access control - **Versionato**: Supporto multiple versioni di stesso tool

Responsabilità: 1. **Registrazione Tool**: Registrare nuovi tool con metadata 2. **Scoperta**: Trovare tool che soddisfano requirements 3. **Capability Matching**: Match

task needs -> available tools 4. **Invocazione**: Eseguire tool calls in modo sicuro 5. **Gestione Risultati**: Parse e validate tool outputs 6. **Gestione Errori**: Gestire tool failures gracefully

1.2 Schema di Specifica Tool

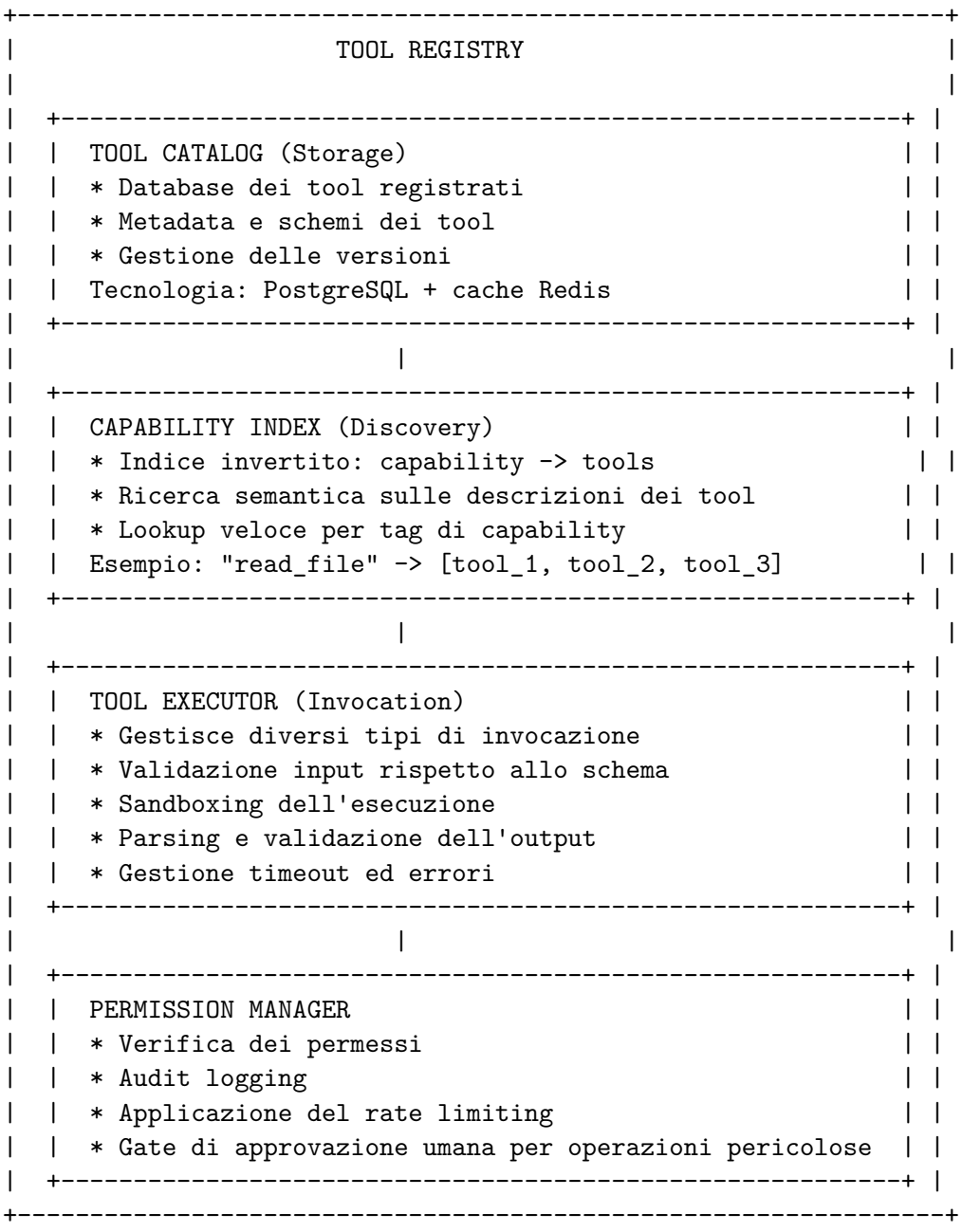
Definizione Completa di un Tool:

```
Tool {
  // Identificazione
  tool_id: string (UUID),
  name: string,  // Nome leggibile
  version: string,  // Versionamento semantico (es. "2.1.0")

  // Descrizione
  description: string,
  detailed_description: string,
  category: ToolCategory,  // "file_ops", "web", "compute", etc.

  // Capability
  capabilities: [string],  // Tag semantici: ["read_file", "text_file"]

  // Interfaccia
  input_schema: JSONSchema,  // Parametri accettati dallo strumento
  output_schema: JSONSchema,  // Cosa restituisce lo strumento

  // Esempi
  examples: [
    {
      description: string,
      input: {...},
      expected_output: {...}
    }
  ],

  // Esecuzione
  invocation: {
    type: "FUNCTION" | "HTTP_API" | "CLI" | "PLUGIN",

    // Per tipo FUNCTION
    function_ref: string | callable,

    // Per tipo HTTP_API
    endpoint: string,
    method: "GET" | "POST" | "PUT" | "DELETE",
    auth: AuthConfig,

    // Per tipo CLI
```

```
  command_template: string,

  // Per tipo PLUGIN
  plugin_loader: string
},

// Sicurezza & Permessi
safety: {
  permission_required: Permission,
  dangerous: boolean,  // Richiede validazione extra
  side_effects: [SideEffect],  // Cosa modifica
  reversible: boolean,  // L'azione può essere annullata
  idempotent: boolean,  // Sicuro riprovare
  rate_limit: RateLimit
},

// Prestazioni
performance: {
  typical_latency: float,  // Secondi
  timeout: float,  // Tempo massimo di esecuzione
  cost: float | null,  // Per invocazione, se applicabile
  resource_intensive: boolean
},

// Dipendenze
dependencies: {
  required_tools: [tool_id],
  required_permissions: [Permission],
  required_environment: {
    os: [string],
    libraries: [string]
  }
},

// Metadata
metadata: {
  author: string,
  documentation_url: string,
  source_code_url: string,
  license: string,
  tags: [string]
},

// Ciclo di vita
lifecycle: {
  status: "ACTIVE" | "DEPRECATED" | "DISABLED",
  deprecated_by: tool_id | null,
  created_at: datetime,
```

```
    updated_at: datetime
  }
}
```

1.3 Architettura del Tool Registry



1.4 Scoperta e Matching dei Tool

Algoritmo di Scoperta:

```
Funzione DISCOVER_TOOLS(requirement, context):
```

```
  # STEP 1: Parse del requirement
  # Il Requirement potrebbe essere linguaggio naturale o strutturato

  SE requirement.is_structured:
    # Strutturato: {capability: "read_file", constraints: {...}}
    capabilities_needed = requirement.capabilities
    constraints = requirement.constraints
  ALTRIMENTI:
    # Linguaggio naturale: "Ho bisogno di leggere un file JSON"
    # Usa LLM per estrarre le capability
    capabilities_needed = LLM.extract_capabilities(requirement.text)
    constraints = LLM.extract_constraints(requirement.text)

  # STEP 2: Query per capability
  candidate_tools = []

  PER OGNI capability IN capabilities_needed:
    # Query sull'indice delle capability
    tools_with_capability = INDEX.query(capability)
    candidate_tools.extend(tools_with_capability)

  # Rimuovi duplicati
  candidate_tools = UNIQUE(candidate_tools)

  # STEP 3: Filtra per vincoli
  filtered_tools = []

  PER OGNI tool IN candidate_tools:
    # Verifica se il tool soddisfa tutti i vincoli
    SE MATCHES_CONSTRAINTS(tool, constraints, context):
      filtered_tools.append(tool)

  # STEP 4: Ordina per idoneità
  ranked_tools = RANK_TOOLS(filtered_tools, requirement, context)

  # STEP 5: Restituisci i migliori match
  RESTITUISCI ranked_tools[:5]

Funzione MATCHES_CONSTRAINTS(tool, constraints, context):
  # Verifica permessi
  SE tool.safety.permission_required NON IN context.available_permissions:
    RESTITUISCI False

  # Verifica ambiente
  SE NON environment_compatible(tool, context.environment):
    RESTITUISCI False
```

```
  # Vincolo di prestazioni
  SE constraints.max_latency AND tool.performance.typical_latency > constraints.max_latency:
    RESTITUISCI False

  # Vincolo di costo
  SE constraints.max_cost AND tool.performance.cost > constraints.max_cost:
    RESTITUISCI False

  # Vincolo di sicurezza
  SE constraints.safe_only AND tool.safety.dangerous:
    RESTITUISCI False

  RESTITUISCI True

Funzione RANK_TOOLS(tools, requirement, context):
  scored = []

  PER OGNI tool IN tools:
    score = 0

    # Qualità del match delle capability (primario)
    match_quality = COMPUTE_CAPABILITY_MATCH(
      tool.capabilities,
      requirement.capabilities_needed
    )
    score += 0.4 * match_quality

    # Frequenza d'uso (tool popolari preferiti)
    usage_factor = LOG(1 + tool.usage_count) / 10
    score += 0.2 * MIN(usage_factor, 1.0)

    # Score di prestazioni (più veloce è meglio)
    latency_score = 1 / (1 + tool.performance.typical_latency)
    score += 0.15 * latency_score

    # Score di costo (più economico è meglio)
    SE tool.performance.cost:
      cost_score = 1 / (1 + tool.performance.cost * 10)
      score += 0.15 * cost_score
    ALTRIMENTI:
      score += 0.15  # Tool gratuiti ottengono score pieno

    # Score di sicurezza (più sicuro è meglio)
    safety_score = 1.0 se non tool.safety.dangerous altrimenti 0.5
    score += 0.1 * safety_score

    scored.append((tool, score))
```

```
  RESTITUISCI SORT_BY_SCORE(scored, descending=True)
```

1.5 Invocazione dei Tool

Flusso di Invocazione Sicuro:

```
+-------------------------------------------------------------+
|                  PIPELINE DI INVOCAZIONE TOOL                |
|                                                              |
|  Richiesta Utente: Eseguire tool X con parametri Y           |
|    |                                                         |
|  +-----------------------------------------------------+     |
|  |  1. VERIFICA PERMESSI                               |     |
|  |  * Verifica che il chiamante abbia il permesso richiesto |  |
|  |  * Verifica che i rate limit non siano superati     |     |
|  |  * Registra la richiesta per l'audit                |     |
|  |  SE FALLISCE: RIFIUTA con errore                    |     |
|  +-----------------------+-----------------------------+     |
|                          | PASSA                             |
|  +-----------------------------------------------------+     |
|  |  2. VALIDAZIONE INPUT                               |     |
|  |  * Valida rispetto a tool.input_schema              |     |
|  |  * Type checking                                    |     |
|  |  * Verifica range/vincoli                           |     |
|  |  * Rilevamento attacchi injection                   |     |
|  |  SE FALLISCE: RIFIUTA con errori di validazione     |     |
|  +-----------------------+-----------------------------+     |
|                          | PASSA                             |
|  +-----------------------------------------------------+     |
|  |  3. VERIFICA SICUREZZA (se tool pericoloso)         |     |
|  |  * Verifica azione rispetto ai bounds di sicurezza  |     |
|  |  * Simula/dry-run se possibile                      |     |
|  |  * Richiede approvazione umana se necessario        |     |
|  |  SE FALLISCE: RIFIUTA con violazione sicurezza      |     |
|  +-----------------------+-----------------------------+     |
|                          | PASSA                             |
|  +-----------------------------------------------------+     |
|  |  4. ESECUZIONE                                      |     |
|  |  * Prepara ambiente di esecuzione                   |     |
|  |  * Imposta timeout                                  |     |
|  |  * Invoca tool in base al tipo di invocazione:      |     |
|  |    - FUNCTION: Chiama direttamente                  |     |
|  |    - HTTP_API: Effettua richiesta HTTP              |     |
|  |    - CLI: Esegue comando shell                      |     |
|  |    - PLUGIN: Carica e invoca plugin                 |     |
|  |  * Cattura output ed errori                         |     |
|  |  SE TIMEOUT: Interrompi e restituisci errore timeout |    |
|  |  SE ERRORE: Cattura dettagli errore                 |     |
```

```
|  +-----------------------+---------------------------------+ |
|                          |                                   |
|  +-------------------------------------------------------+ |
|  |  5. ELABORAZIONE OUTPUT                                | |
|  |  * Parsa l'output del tool                             | |
|  |  * Valida rispetto a tool.output_schema                | |
|  |  * Estrae dati strutturati                             | |
|  |  * Gestisce gli errori con garbo                       | |
|  |  SE errore parsing: Incapsula in risposta errore       | |
|  +-----------------------+---------------------------------+ |
|                          |                                   |
|  +-------------------------------------------------------+ |
|  |  6. POST-INVOCAZIONE                                   | |
|  |  * Registra esecuzione (tool, params, result, duration)  | |
|  |  * Aggiorna statistiche d'uso del tool                 | |
|  |  * Aggiorna contatori rate limit                       | |
|  |  * Memorizza in working memory se necessario           | |
|  +-----------------------+---------------------------------+ |
|                          |                                   |
|  Restituisci ToolResult {                                    |
|    success: boolean,                                         |
|    output: {...} | null,                                    |
|    error: Error | null,                                      |
|    metadata: {duration, cost, ...}                           |
|  }                                                           |
+--------------------------------------------------------------+
```

1.6 Categorie di Tool Built-in

Strumenti Essenziali (sempre disponibili):

```
+-----------------------------------------------------+
|               CATEGORIE DI TOOL PRINCIPALI           |
|                                                      |
|  1. OPERAZIONI FILE                                  |
|     * read_file: Legge contenuto file               |
|     * write_file: Scrive/sovrascrive file           |
|     * append_file: Appende a file                   |
|     * delete_file: Elimina file                     |
|     * list_directory: Elenca file nella directory   |
|     * file_info: Ottiene metadata del file          |
|                                                      |
|  2. OPERAZIONI CODICE                                |
|     * parse_code: Parsa codice in AST               |
|     * format_code: Formatta automaticamente codice  |
|     * lint_code: Esegue linter                      |
|     * run_tests: Esegue suite di test               |
|     * static_analysis: Analizza qualità del codice  |
```

```
|                                                          |
|  3. OPERAZIONI WEB                                       |
|     * http_get: Richiesta HTTP GET                       |
|     * http_post: Richiesta HTTP POST                     |
|     * fetch_webpage: Recupera e parsa HTML               |
|     * web_search: Cerca sul web                          |
|                                                          |
|  4. OPERAZIONI DATI                                      |
|     * parse_json: Parsa stringa JSON                     |
|     * parse_yaml: Parsa YAML                             |
|     * parse_xml: Parsa XML                               |
|     * query_database: Query SQL                          |
|     * transform_data: Trasformazione dati                |
|                                                          |
|  5. COMPUTAZIONE                                         |
|     * evaluate_expression: Valuta espressione matematica|
|     * run_script: Esegue script                          |
|     * execute_command: Esegue comando shell              |
|                                                          |
|  6. COMUNICAZIONE                                        |
|     * send_notification: Notifica l'utente               |
|     * request_input: Chiede input all'utente             |
|     * request_approval: Richiede approvazione umana      |
|                                                          |
|  7. OPERAZIONI MEMORIA                                   |
|     * search_episodes: Query su memoria episodica        |
|     * retrieve_pattern: Ottiene pattern dalla cache      |
|     * store_note: Salva nota per dopo                    |
+----------------------------------------------------------+
```

1.7 Operazioni del Tool Registry

```
1. REGISTER_TOOL(tool_spec)
   * Valida specifica del tool
   * Verifica conflitti di nome
   * Assegna tool_id
   * Crea indici di capability
   * Memorizza nel catalogo
   * Restituisce tool_id

2. UNREGISTER_TOOL(tool_id)
   * Marca tool come disabilitato
   * Rimuove dagli indici attivi
   * Mantiene per record storico

3. UPDATE_TOOL(tool_id, updates)
   * Valida aggiornamenti
   * Incrementa versione
```

```
   * Aggiorna catalogo
   * Aggiorna indici
   * Notifica tool dipendenti se breaking change

4. GET_TOOL(tool_id)
   * Recupera specifica tool dal catalogo
   * Verifica se attivo
   * Restituisce tool o errore

5. DISCOVER_TOOLS(requirement, context)
   * Parsa requirement
   * Query sull'indice di capability
   * Filtra per vincoli
   * Ordina per idoneità
   * Restituisce i migliori match

6. INVOKE_TOOL(tool_id, parameters, context)
   * Verifica permessi
   * Validazione input
   * Verifica sicurezza (se pericoloso)
   * Esegue tool
   * Elaborazione output
   * Logging
   * Restituisce risultato

7. LIST_TOOLS(filters)
   * Query sul catalogo con filtri
   * Restituisce tool corrispondenti

8. GET_TOOL_USAGE_STATS(tool_id, time_range)
   * Query sui log d'uso
   * Calcola statistiche
   * Restituisce metriche

9. CHECK_PERMISSIONS(tool_id, context)
   * Verifica permessi richiesti disponibili
   * Restituisce boolean + permessi mancanti se presenti

10. REQUEST_APPROVAL(tool_id, parameters, reason)
    * Per operazioni pericolose
    * Presenta richiesta all'umano
    * Attende approvazione/rifiuto
    * Restituisce decisione
```

2. Model Router

2.1 Scopo e Responsabilità

Funzione Centrale: Selezionare optimal LLM per ogni reasoning task, bilanciando quality, cost, e latency.

Key Insight: Not all tasks require most capable (expensive) model. Strategic routing can reduce cost 5-10x without quality loss.

Responsabilità: 1. **Selezione Modello**: Scegli modello appropriato per il task 2. **Bilanciamento del Carico**: Distribuisci richieste tra istanze 3. **Failover**: Gestisci indisponibilità del modello 4. **Gestione Contesto**: Assicura che il task rientri nella finestra di contesto del modello 5. **Ottimizzazione Costi**: Minimizza la spesa mantenendo gli obiettivi di qualità 6. **Tracking Prestazioni**: Monitora prestazioni del modello per tipo di task

2.2 Sistema a Livelli di Modello

Architettura a Tre Livelli:

```
+-----------------------------------------------------------+
|                    LIVELLI DI MODELLO                     |
|                                                           |
|  LIVELLO 1: MODELLI PICCOLI/VELOCI                        |
|  +- Esempi: GPT-3.5, Claude Haiku, Llama 2 7B           |
|  +- Costo: $0.001-0.002 per 1K token                      |
|  +- Latenza: 1-3s tipica                                  |
|  +- Contesto: 4K-8K token                                 |
|  +- Punti di forza: Velocità, costo, task semplici        |
|  +- Usare per: Query semplici, formattazione,             |
|                classificazione                            |
|                                                           |
|  LIVELLO 2: MODELLI MEDI                                  |
|  +- Esempi: GPT-4o-mini, Claude Sonnet                  |
|  +- Costo: $0.01-0.03 per 1K token                        |
|  +- Latenza: 3-8s tipica                                  |
|  +- Contesto: 16K-128K token                              |
|  +- Punti di forza: Equilibrio tra capability e costo     |
|  +- Usare per: Maggior parte coding, task complessità     |
|                moderata                                   |
|                                                           |
|  LIVELLO 3: MODELLI GRANDI/CAPACI                         |
|  +- Esempi: GPT-4, Claude Opus, GPT-4-turbo             |
|  +- Costo: $0.03-0.10 per 1K token                        |
|  +- Latenza: 5-15s tipica                                 |
|  +- Contesto: 128K-200K token                             |
|  +- Punti di forza: Massima capability, reasoning         |
|  |                   complesso                            |
|  +- Usare per: Architettura complessa, problemi nuovi     |
```

```
+-----------------------------------------------------------+
```

2.3 Albero Decisionale di Routing

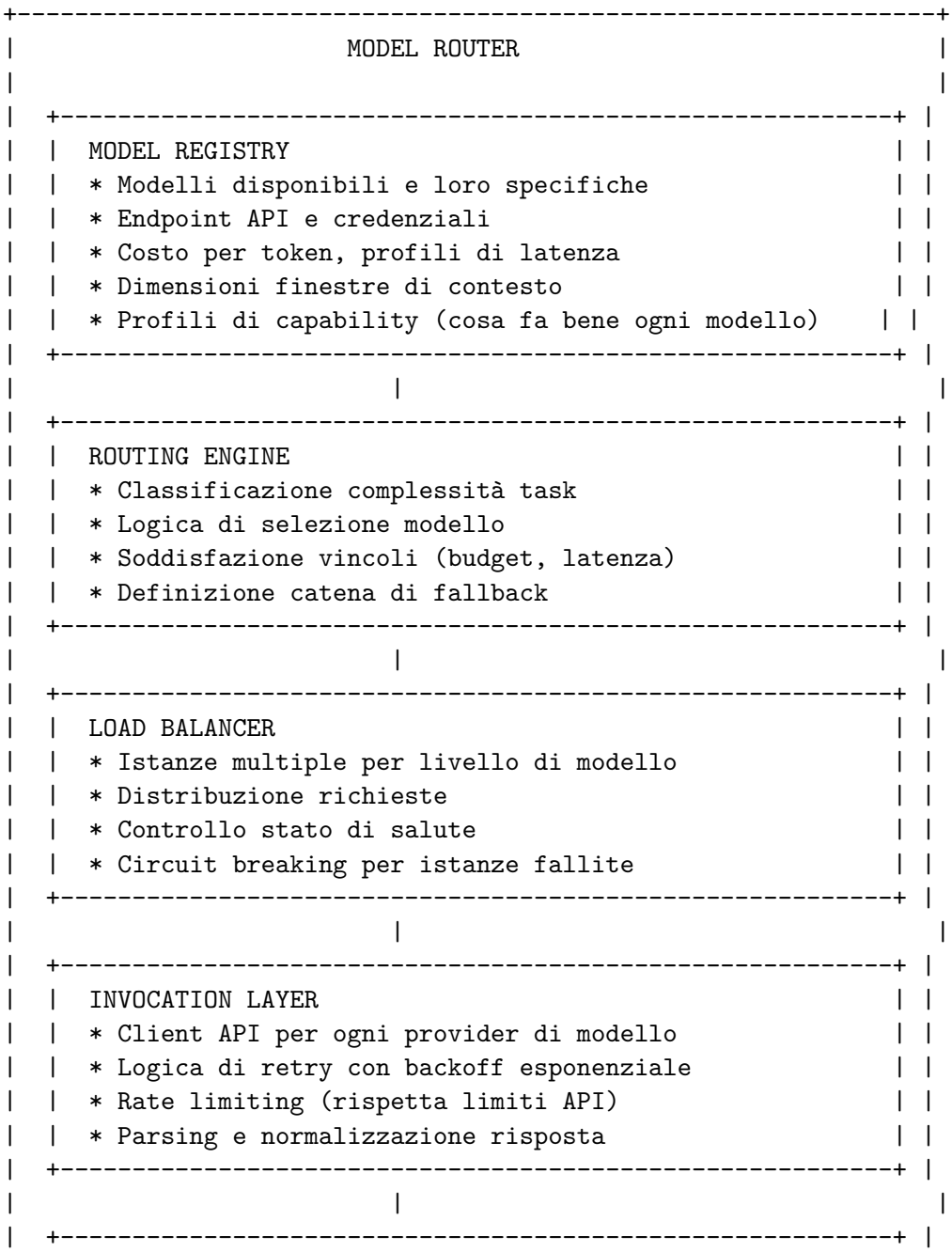
Algoritmo di Decisione:

```
Funzione SELECT_MODEL(task, context):

  # STEP 1: Classifica complessità del task
  complexity = CLASSIFY_COMPLEXITY(task)
  # Restituisce: SIMPLE | MODERATE | COMPLEX

  # STEP 2: Verifica suggerimenti di routing
  SE context.model_hint:
    # Suggerimento esplicito dalla pianificazione (es. "use_best_model")
    SE context.model_hint == "best":
      RESTITUISCI TIER_3_MODEL
    ALTRIMENTI SE context.model_hint == "fast":
      RESTITUISCI TIER_1_MODEL

  # STEP 3: Routing per complessità e altri fattori

  SE complexity == SIMPLE:
    # Task semplici -> Modelli veloci

    SE task.requires_creativity:
      # Task creativi necessitano modelli migliori
      RESTITUISCI TIER_2_MODEL
    ALTRIMENTI SE task.is_classification:
      # Classificazione va bene con modello piccolo
      RESTITUISCI TIER_1_MODEL
    ALTRIMENTI SE task.is_formatting:
      # Formattazione banale
      RESTITUISCI TIER_1_MODEL
    ALTRIMENTI:
      # Default semplice
      RESTITUISCI TIER_1_MODEL

  ALTRIMENTI SE complexity == MODERATE:
    # Task moderati -> Modelli medi per lo più

    SE task.is_coding AND task.language IN WELL_KNOWN_LANGUAGES:
      # Task di coding comune
      RESTITUISCI TIER_2_MODEL
    ALTRIMENTI SE task.requires_extensive_reasoning:
      # Richiede thinking -> Modello migliore
      RESTITUISCI TIER_3_MODEL
    ALTRIMENTI SE task.context_size > TIER_2_CONTEXT_LIMIT:
```

```
      # Troppo contesto per tier 2
      RESTITUISCI TIER_3_MODEL
    ALTRIMENTI:
      # Default moderato
      RESTITUISCI TIER_2_MODEL

  ALTRIMENTI:  # complexity == COMPLEX
    # Task complessi -> Modelli grandi

    SE task.is_novel AND NOT task.has_similar_patterns:
      # Problema nuovo richiede modello migliore
      RESTITUISCI TIER_3_MODEL
    ALTRIMENTI SE task.is_safety_critical:
      # Critico per sicurezza -> Usa il migliore
      RESTITUISCI TIER_3_MODEL
    ALTRIMENTI SE budget_constrained(context):
      # Budget stretto -> Prova tier 2 prima
      RESTITUISCI TIER_2_MODEL_WITH_FALLBACK_TO_TIER_3
    ALTRIMENTI:
      # Default complesso
      RESTITUISCI TIER_3_MODEL

Funzione CLASSIFY_COMPLEXITY(task):
  score = 0

  # Fattore 1: Tipo di task
  SE task.type IN ["classification", "formatting", "simple_query"]:
    score += 1
  ALTRIMENTI SE task.type IN ["coding", "analysis", "planning"]:
    score += 2
  ALTRIMENTI SE task.type IN ["architecture", "novel_problem", "research"]:
    score += 3

  # Fattore 2: Profondità del reasoning
  SE task.requires_multi_step_reasoning:
    score += 1
  SE task.requires_abstraction:
    score += 1
  SE task.requires_creativity:
    score += 1

  # Fattore 3: Dimensione del contesto
  SE task.context_size > 20K:
    score += 1

  # Fattore 4: Incertezza
  SE task.has_high_uncertainty:
    score += 1
```

```
  # Classificazione
  SE score <= 2:
    RESTITUISCI SIMPLE
  ALTRIMENTI SE score <= 5:
    RESTITUISCI MODERATE
  ALTRIMENTI:
    RESTITUISCI COMPLEX
```

2.4 Architettura del Model Router



```
|  |  PERFORMANCE TRACKER                                        | |
|  |  * Registra ogni invocazione del modello                    | |
|  |  * Traccia: latenza, costo, qualità (quando disponibile)    | |
|  |  * Rileva degrado delle prestazioni                         | |
|  |  * Genera raccomandazioni per ottimizzazione routing        | |
|  +-------------------------------------------------------------+ |
+-------------------------------------------------------------------+
```

2.5 Gestione della Finestra di Contesto

Sfida: I task possono avere contesto che supera la finestra del modello.

Strategie:

```
+-----------------------------------------------------------+
|       GESTIONE DELLA FINESTRA DI CONTESTO                 |
|                                                           |
|  STRATEGIA 1: UPGRADE DEL MODELLO                         |
|  * Se context > current_model.context_limit               |
|  * Instrada a modello con finestra di contesto più grande|
|  * Esempio: GPT-3.5 (4K) -> GPT-4-turbo (128K)            |
|  * Pro: Semplice, nessuna perdita di informazioni         |
|  * Contro: Più costoso                                    |
|                                                           |
|  STRATEGIA 2: COMPRESSIONE DEL CONTESTO                   |
|  * Riassumi parti non essenziali                          |
|  * Mantieni intatto il contesto caldo (recente,           |
|    rilevante)                                             |
|  * Usa LLM per generare riassunto del contesto freddo     |
|  * Pro: Si adatta a finestra di contesto più piccola      |
|  * Contro: Potenziale perdita di informazioni             |
|                                                           |
|  STRATEGIA 3: DIVISIONE DEL CONTESTO                      |
|  * Dividi task in sub-task con contesto più piccolo       |
|  * Elabora ciascuno indipendentemente                     |
|  * Unisci risultati                                       |
|  * Pro: Può usare modelli più veloci                      |
|  * Contro: Può perdere reasoning cross-context            |
|                                                           |
|  STRATEGIA 4: RETRIEVAL-AUGMENTED                         |
|  * Memorizza contesto completo esternamente               |
|  * Recupera pezzi rilevanti on-demand                     |
|  * Include solo contesto recuperato nel prompt            |
|  * Pro: Gestisce dimensione contesto illimitata           |
|  * Contro: Qualità del retrieval critica                  |
+-----------------------------------------------------------+
```

Algoritmo di Gestione del Contesto:

```
Funzione MANAGE_CONTEXT(task, model):
```

```
  context_size = ESTIMATE_TOKENS(task.context)

  SE context_size <= model.context_window * 0.8:
    # Si adatta comodamente (soglia 80% per margine di sicurezza)
    RESTITUISCI task.context

  ALTRIMENTI SE context_size <= TIER_3_MODEL.context_window * 0.8:
    # Non si adatta al modello corrente ma si adatta a uno più grande
    # Upgrade del modello
    upgraded_model = SELECT_MODEL_WITH_CONTEXT(context_size)
    RESTITUISCI (task.context, upgraded_model)

  ALTRIMENTI:
    # Non si adatta nemmeno al modello più grande
    # Deve comprimere o dividere

    SE task.allows_summarization:
      # Strategia 2: Comprimi
      compressed_context = COMPRESS_CONTEXT(
        task.context,
        target_size=model.context_window * 0.7
      )
      RESTITUISCI compressed_context

    ALTRIMENTI SE task.is_splittable:
      # Strategia 3: Dividi
      subtasks = SPLIT_TASK(task, max_context_size=model.context_window * 0.7)
      RESTITUISCI subtasks  # Saranno elaborati separatamente

    ALTRIMENTI:
      # Strategia 4: Retrieval-augmented
      relevant_context = RETRIEVE_RELEVANT_CONTEXT(
        task.query,
        full_context=task.context,
        max_size=model.context_window * 0.7
      )
      RESTITUISCI relevant_context

Funzione COMPRESS_CONTEXT(context, target_size):
  # Identifica contenuto caldo vs freddo
  hot = IDENTIFY_HOT_CONTENT(context)  # Recente, referenziato
  cold = context - hot

  current_size = SIZE(hot)

  SE current_size >= target_size:
    # Anche il contenuto caldo è troppo grande, deve riassumere in modo aggressivo
```

```
    RESTITUISCI AGGRESSIVE_SUMMARIZE(hot, target_size)

  # Riassumi contenuto freddo per adattarsi
  remaining_budget = target_size - current_size
  cold_summary = SUMMARIZE(cold, max_size=remaining_budget)

  RESTITUISCI hot + cold_summary
```

2.6 Ottimizzazione dei Costi

Tracking e Ottimizzazione dei Costi:

```
CostOptimizer {
  // Gestione Budget
  budget: {
    daily_limit: float,
    per_task_limit: float,
    current_spend_today: float
  },

  // Preferenze di Routing
  routing_policy: {
    mode: "MINIMIZE_COST" | "BALANCE" | "MAXIMIZE_QUALITY",

    // Per modalità BALANCE
    cost_weight: float,  // 0-1
    quality_weight: float,  // 0-1
    latency_weight: float  // 0-1
  },

  // Profili di Costo dei Modelli
  model_costs: {
    model_id: {
      input_cost_per_1k: float,
      output_cost_per_1k: float,
      minimum_charge: float
    }
  }
}

Funzione OPTIMIZE_ROUTING(task, budget_remaining):

  SE budget_remaining < MINIMUM_VIABLE_BUDGET:
    # Emergenza: Usa solo il modello più economico
    RESTITUISCI TIER_1_MODEL

  # Stima costo per ogni modello praticabile
  viable_models = []
```

```
  PER OGNI tier IN [TIER_1, TIER_2, TIER_3]:
    model = GET_MODEL_FOR_TIER(tier)
    estimated_cost = ESTIMATE_COST(task, model)

    SE estimated_cost <= budget_remaining * 0.1:
      # Usa al massimo il 10% del budget rimanente per task
      quality_score = ESTIMATE_QUALITY(task, model)
      viable_models.append({
        model: model,
        cost: estimated_cost,
        quality: quality_score
      })

  # Seleziona in base alla policy
  SE routing_policy.mode == "MINIMIZE_COST":
    RESTITUISCI MIN(viable_models, key=lambda x: x.cost).model

  ALTRIMENTI SE routing_policy.mode == "MAXIMIZE_QUALITY":
    RESTITUISCI MAX(viable_models, key=lambda x: x.quality).model

  ALTRIMENTI:  # Modalità BALANCE
    # Calcola score ponderato
    best = None
    best_score = -INF

    PER OGNI candidate IN viable_models:
      # Normalizza costo e qualità a [0, 1]
      cost_normalized = 1 - (candidate.cost / MAX_COST)
      quality_normalized = candidate.quality

      score = (
        cost_weight * cost_normalized +
        quality_weight * quality_normalized
      )

      SE score > best_score:
        best_score = score
        best = candidate.model

    RESTITUISCI best

Funzione ESTIMATE_COST(task, model):
  input_tokens = ESTIMATE_TOKENS(task.context + task.prompt)
  output_tokens = ESTIMATE_OUTPUT_TOKENS(task)

  input_cost = (input_tokens / 1000) * model.input_cost_per_1k
  output_cost = (output_tokens / 1000) * model.output_cost_per_1k
```

```
  RESTITUISCI input_cost + output_cost + model.minimum_charge
```

2.7 Logica di Fallback e Retry

Gestione dei Fallimenti del Modello:

```
Funzione INVOKE_WITH_FALLBACK(task, primary_model):

  # Definisci catena di fallback
  fallback_chain = BUILD_FALLBACK_CHAIN(primary_model)
  # Esempio: [GPT-4, Claude-Opus, GPT-4-turbo]

  PER OGNI model IN fallback_chain:
    PROVA:
      result = INVOKE_MODEL(model, task, max_retries=3)
      RESTITUISCI result

    ECCETTO RateLimitError:
      # Rate limited, prova modello successivo
      LOG_WARNING(f"Rate limited su {model}, provo fallback")
      CONTINUA

    ECCETTO TimeoutError:
      # Timeout, prova modello successivo
      LOG_WARNING(f"Timeout su {model}, provo fallback")
      CONTINUA

    ECCETTO ModelUnavailable:
      # Modello non disponibile, prova modello successivo
      LOG_WARNING(f"{model} non disponibile, provo fallback")
      CONTINUA

    ECCETTO InvalidRequestError:
      # Problema con la nostra richiesta, improbabile che funzioni su altro modello
      RESTITUISCI ERROR(InvalidRequestError)

  # Tutti i modelli nella catena sono falliti
  RESTITUISCI ERROR("Tutti i modelli sono falliti")

Funzione INVOKE_MODEL(model, task, max_retries=3):
  PER OGNI attempt IN RANGE(1, max_retries + 1):
    PROVA:
      response = model.api_client.complete(
        prompt=task.prompt,
        context=task.context,
        max_tokens=task.max_output_tokens,
        timeout=model.typical_latency * 3
```

```
    )
    RESTITUISCI response

  ECCETTO (NetworkError, TransientError):
    SE attempt < max_retries:
      # Riprova con backoff esponenziale
      wait_time = 2 ** attempt  # 2s, 4s, 8s
      SLEEP(wait_time)
      CONTINUA
    ALTRIMENTI:
      RILANCIA

  ECCETTO NonRetriableError:
    # Non riprovare
    RILANCIA
```

2.8 Operazioni del Model Router

```
1. SELECT_MODEL(task, context, constraints)
   * Classifica complessità del task
   * Valuta vincoli (budget, latenza)
   * Applica policy di routing
   * Restituisce modello selezionato

2. INVOKE_MODEL(model_id, prompt, context, parameters)
   * Bilancia carico tra istanze
   * Gestisce finestra di contesto
   * Invoca API
   * Gestisce retry
   * Restituisce risposta

3. ESTIMATE_COST(task, model_id)
   * Stima uso dei token
   * Calcola costo
   * Restituisce stima

4. GET_MODEL_INFO(model_id)
   * Restituisce specifiche del modello
   * Finestra di contesto, costo, latenza, etc.

5. UPDATE_MODEL_STATS(model_id, invocation_result)
   * Registra latenza
   * Registra costo
   * Aggiorna tasso di successo
   * Rileva problemi di prestazioni

6. GET_ROUTING_RECOMMENDATIONS()
   * Analizza pattern di routing
```

```
   * Identifica opportunità di ottimizzazione
   * Restituisce raccomandazioni

7. SET_ROUTING_POLICY(policy)
   * Aggiorna preferenze di routing
   * Applica nuova policy alle richieste future

8. CHECK_BUDGET_STATUS()
   * Restituisce budget rimanente
   * Tasso di spesa
   * Tempo previsto di esaurimento
```

3. Safety Verifier

3.1 Scopo e Responsabilità

Funzione Centrale: Enforce safety bounds su tutte le operazioni dell'agente, implementando bounded emergence framework.

Ruolo Critico: Questo è il layer che previene l'agente da: - Eseguire azioni non autorizzate - Generare output dannosi - Violare constraint di sistema - Superare limiti di risorse - Operare fuori dai bounds definiti

Responsabilità: 1. **Validazione Input**: Verify input safety e sanity 2. **Autorizzazione Azioni**: Approva/rifiuta richieste di azione 3. **Verifica Output**: Valida che l'output soddisfi i requisiti 4. **Applicazione Bounds**: Assicura che l'agente operi entro i bound definiti 5. **Prevenzione Injection**: Rileva e blocca attacchi injection 6. **Audit Logging**: Registra tutte le decisioni di sicurezza

3.2 Tassonomia dei Safety Bounds

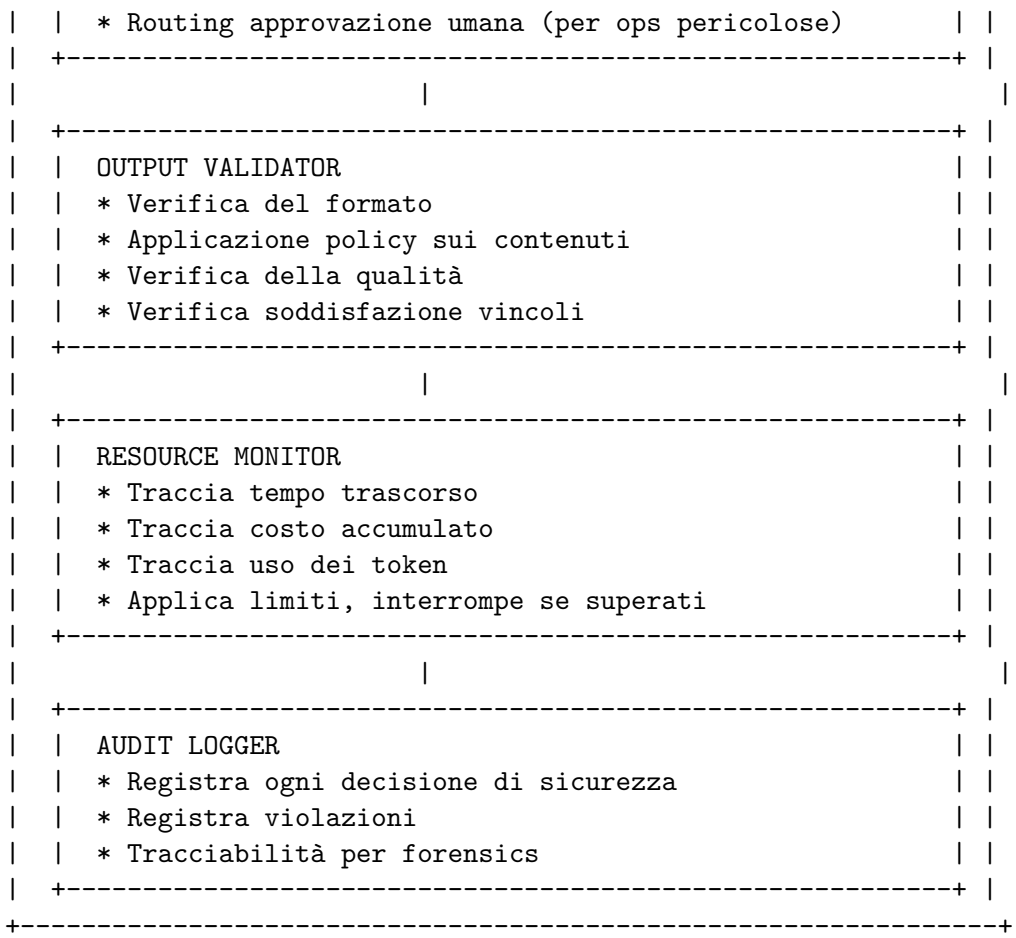
Cinque Tipi di Bounds (da Bounded Emergence Theory):

```
+-----------------------------------------------------------+
|                     SAFETY BOUNDS                         |
|                                                           |
|  1. INPUT BOUNDS                                          |
|     * Quale input l'agente può processare                 |
|     * Validazione dello schema                            |
|     * Limiti di dimensione                                |
|     * Filtraggio dei contenuti                            |
|     * Rilevamento injection                               |
|     Esempio: "L'input deve essere JSON valido < 1MB"      |
|                                                           |
|  2. OUTPUT BOUNDS                                         |
|     * Quale output l'agente può generare                  |
|     * Requisiti di formato                                |
|     * Policy sui contenuti                                |
|     * Soglie di qualità                                   |
```

```
|     Esempio: "L'output deve essere codice Python valido" |
|                                                          |
|  3. ACTION BOUNDS (Più Critici)                          |
|     * Quali azioni l'agente può eseguire                 |
|     * Whitelist degli strumenti                          |
|     * Sistema di permessi                                |
|     * Operazioni proibite                                |
|     Esempio: "Può leggere file ma non eliminarli"        |
|                                                          |
|  4. SAFETY BOUNDS                                        |
|     * Limiti rigidi su operazioni pericolose             |
|     * Gate di approvazione umana                         |
|     * Protezione azioni irreversibili                    |
|     Esempio: "Non accedere mai al database di produzione"|
|                                                          |
|  5. RESOURCE BOUNDS                                      |
|     * Limiti sul consumo di risorse                      |
|     * Limiti di tempo                                    |
|     * Budget token/costo                                 |
|     * Limiti di rate                                     |
|     Esempio: "Max 5 minuti, $1 per task"                 |
+----------------------------------------------------------+
```

3.3 Architettura del Safety Verifier

```
+-------------------------------------------------------------------+
|                      SAFETY VERIFIER                              |
|                                                                   |
|  +-------------------------------------------------------------+  |
|  |  BOUNDS REGISTRY                                            |  |
|  |  * Memorizza tutti i safety bounds definiti                 |  |
|  |  * Struttura gerarchica (globale -> specifico per task)     |  |
|  |  * Versionato                                               |  |
|  +-------------------------------------------------------------+  |
|                          |                                        |
|  +-------------------------------------------------------------+  |
|  |  INPUT VALIDATOR                                            |  |
|  |  * Validazione dello schema                                 |  |
|  |  * Rilevamento injection (SQL, command, prompt)             |  |
|  |  * Filtraggio dei contenuti                                 |  |
|  |  * Verifica dimensione/formato                              |  |
|  +-------------------------------------------------------------+  |
|                          |                                        |
|  +-------------------------------------------------------------+  |
|  |  ACTION AUTHORIZER                                          |  |
|  |  * Verifica permessi                                        |  |
|  |  * Applicazione whitelist degli strumenti                   |  |
|  |  * Rilevamento azioni proibite                              |  |
```



3.4 Validazione Input

Validazione Input Multi-Layer:

```
Funzione VALIDATE_INPUT(input, bounds, context):

  # LAYER 1: Validazione dello Schema
  SE bounds.input_schema:
    validation_result = VALIDATE_SCHEMA(input, bounds.input_schema)
    SE NON validation_result.valid:
      RESTITUISCI REJECT("Validazione schema fallita", validation_result.errors)

  # LAYER 2: Limiti di Dimensione
  input_size = COMPUTE_SIZE(input)
  SE input_size > bounds.max_input_size:
    RESTITUISCI REJECT(f"Input troppo grande: {input_size} > {bounds.max_input_size}")

  # LAYER 3: Rilevamento Injection
  injection_checks = [
    CHECK_SQL_INJECTION(input),
    CHECK_COMMAND_INJECTION(input),
```

```
    CHECK_PROMPT_INJECTION(input),
    CHECK_PATH_TRAVERSAL(input)
  ]

  PER OGNI check IN injection_checks:
    SE check.detected:
      RESTITUISCI REJECT(f"Violazione sicurezza: {check.type}", check.details)

  # LAYER 4: Filtraggio Contenuti
  SE bounds.content_policy:
    content_check = CHECK_CONTENT_POLICY(input, bounds.content_policy)
    SE NON content_check.passes:
      RESTITUISCI REJECT("Violazione policy contenuti", content_check.violations)

  # LAYER 5: Validazione Specifica per Contesto
  SE bounds.custom_validators:
    PER OGNI validator IN bounds.custom_validators:
      result = validator(input, context)
      SE NON result.valid:
        RESTITUISCI REJECT(f"Validazione custom fallita: {validator.name}", result.reason)

  # Tutti i controlli passati
  RESTITUISCI ACCEPT()

# Funzioni di Rilevamento Injection

Funzione CHECK_SQL_INJECTION(input):
  # Rileva pattern comuni di SQL injection
  dangerous_patterns = [
    r"'\s*OR\s+'1'\s*=\s*'1",
    r";\s*DROP\s+TABLE",
    r"UNION\s+SELECT",
    r"--\s*$",  # Commento SQL
    r"/\*.*\*/"  # Commento a blocchi SQL
  ]

  PER OGNI pattern IN dangerous_patterns:
    SE REGEX_MATCH(pattern, input, case_insensitive=True):
      RESTITUISCI {detected: True, type: "SQL_INJECTION", pattern: pattern}

  RESTITUISCI {detected: False}

Funzione CHECK_COMMAND_INJECTION(input):
  # Rileva injection di comandi shell
  dangerous_chars = [";", "|", "&", "$", "`", "\n", "$(", "${"]
  dangerous_commands = ["rm", "dd", "mkfs", "wget", "curl"]

  PER OGNI char IN dangerous_chars:
```

```
    SE char IN input:
      RESTITUISCI {detected: True, type: "COMMAND_INJECTION", char: char}

  PER OGNI cmd IN dangerous_commands:
    SE cmd IN input.split():
      RESTITUISCI {detected: True, type: "DANGEROUS_COMMAND", command: cmd}

  RESTITUISCI {detected: False}

Funzione CHECK_PROMPT_INJECTION(input):
  # Rileva tentativi di manipolare l'agente via prompt injection
  injection_indicators = [
    "ignora le istruzioni precedenti",
    "ignora tutto",
    "nuove istruzioni:",
    "system:",
    "modalità admin",
    "sovrascrivi sicurezza"
  ]

  input_lower = input.lower()

  PER OGNI indicator IN injection_indicators:
    SE indicator IN input_lower:
      RESTITUISCI {detected: True, type: "PROMPT_INJECTION", indicator: indicator}

  RESTITUISCI {detected: False}
```

3.5 Autorizzazione Azioni

Autorizzazione Basata su Permessi:

```
Permission {
  permission_id: string,
  name: string,  // es. "file:read", "file:write", "web:fetch"
  description: string,
  category: "READ" | "WRITE" | "DELETE" | "EXECUTE" | "ADMIN",
  risk_level: "LOW" | "MEDIUM" | "HIGH" | "CRITICAL",
  requires_approval: boolean
}

Funzione AUTHORIZE_ACTION(action, context):

  # STEP 1: Verifica se l'azione è esplicitamente proibita
  SE action IN context.bounds.prohibited_actions:
    RESTITUISCI DENY("Azione esplicitamente proibita", reason="security_policy")

  # STEP 2: Verifica permesso richiesto
```

```
  required_permission = action.tool.safety.permission_required

  SE required_permission NON IN context.available_permissions:
    RESTITUISCI DENY(
      "Permessi insufficienti",
      required=required_permission,
      available=context.available_permissions
    )

  # STEP 3: Verifica se l'azione è pericolosa e richiede approvazione
  SE action.tool.safety.dangerous:
    SE action.tool.safety.permission_required.requires_approval:
      SE NON context.has_human_approval:
        # Richiede approvazione umana
        approval = REQUEST_HUMAN_APPROVAL(action, context)
        SE approval.denied:
          RESTITUISCI DENY("Approvazione umana negata", reason=approval.reason)
        # Se approvato, continua

  # STEP 4: Verifica vincoli specifici dell'azione
  SE action.tool.safety.constraints:
    PER OGNI constraint IN action.tool.safety.constraints:
      SE NON EVALUATE_CONSTRAINT(constraint, action, context):
        RESTITUISCI DENY(f"Violazione vincolo: {constraint.name}")

  # STEP 5: Verifica se l'azione supererebbe i resource bounds
  estimated_resource_usage = ESTIMATE_RESOURCE_USAGE(action)
  SE context.resources_used + estimated_resource_usage > context.resource_limits:
    RESTITUISCI DENY("Supererebbe i limiti di risorse")

  # STEP 6: Registra decisione di autorizzazione
  LOG_AUTHORIZATION(action, context, decision="APPROVED")

  RESTITUISCI APPROVE()

Funzione REQUEST_HUMAN_APPROVAL(action, context):
  # Presenta azione all'umano per approvazione
  approval_request = {
    action_description: action.description,
    tool: action.tool.name,
    parameters: action.parameters,
    reason: context.reasoning_for_action,
    risks: action.tool.safety.side_effects,
    reversible: action.tool.safety.reversible
  }

  # Blocca finché l'umano risponde
  response = PRESENT_TO_HUMAN(approval_request)
```

```
  RESTITUISCI response  # {approved: boolean, reason: string}
```

3.6 Validazione Output

Verifica Output Multi-Criteri:

```
Funzione VALIDATE_OUTPUT(output, expected_schema, success_criteria, bounds):

  # LAYER 1: Validazione dello Schema
  SE expected_schema:
    schema_result = VALIDATE_SCHEMA(output, expected_schema)
    SE NON schema_result.valid:
      RESTITUISCI REJECT("Schema output non valido", schema_result.errors)

  # LAYER 2: Verifica Criteri di Successo
  PER OGNI criterion IN success_criteria:
    result = VERIFY_CRITERION(output, criterion)
    SE criterion.required AND NON result.passed:
      RESTITUISCI REJECT(f"Criterio richiesto non soddisfatto: {criterion.description}")

  # LAYER 3: Policy sui Contenuti
  SE bounds.output_content_policy:
    policy_check = CHECK_CONTENT_POLICY(output, bounds.output_content_policy)
    SE NON policy_check.passes:
      RESTITUISCI REJECT("Violazione policy contenuti output", policy_check.violations)

  # LAYER 4: Verifiche di Sicurezza
  safety_checks = [
    CHECK_NO_SECRETS_LEAKED(output),
    CHECK_NO_MALICIOUS_CODE(output),
    CHECK_NO_PII_EXPOSED(output)
  ]

  PER OGNI check IN safety_checks:
    SE check.failed:
      RESTITUISCI REJECT(f"Verifica sicurezza fallita: {check.name}", check.details)

  # LAYER 5: Soglie di Qualità (se specificato)
  SE bounds.min_quality_score:
    quality = ASSESS_OUTPUT_QUALITY(output)
    SE quality < bounds.min_quality_score:
      RESTITUISCI REJECT(f"Qualità output troppo bassa: {quality} < {bounds.min_quality_score}"

  RESTITUISCI ACCEPT()

Funzione CHECK_NO_SECRETS_LEAKED(output):
  # Rileva pattern comuni di segreti
```

```
  secret_patterns = [
    r"(api[_-]?key|apikey)[\s:=]+['\"]\w+['\"]",  # Chiavi API
    r"(password|passwd|pwd)[\s:=]+['\"]\w+['\"]",  # Password
    r"-----BEGIN (RSA |)PRIVATE KEY-----",  # Chiavi private
    r"sk-[A-Za-z0-9]{32,}",  # Chiavi segrete stile OpenAI
    r"ghp_[A-Za-z0-9]{36}",  # Token GitHub
  ]

  PER OGNI pattern IN secret_patterns:
    matches = REGEX_FINDALL(pattern, output, case_insensitive=True)
    SE matches:
      RESTITUISCI {failed: True, name: "SecretDetection", details: "Potenziale segreto trovato"

  RESTITUISCI {failed: False}

Funzione CHECK_NO_MALICIOUS_CODE(output):
  # Se l'output è codice, scansiona per pattern pericolosi
  SE NON IS_CODE(output):
    RESTITUISCI {failed: False}

  dangerous_code_patterns = [
    r"eval\s*\(",  # Eval è pericoloso
    r"exec\s*\(",  # Exec è pericoloso
    r"__import__\s*\(",  # Import dinamici
    r"subprocess\.",  # Esecuzione shell
    r"os\.system",  # Esecuzione shell
    r"rm\s+-rf\s+/",  # Comandi distruttivi
  ]

  PER OGNI pattern IN dangerous_code_patterns:
    SE REGEX_MATCH(pattern, output):
      RESTITUISCI {
        failed: True,
        name: "MaliciousCode",
        details: f"Pattern pericoloso rilevato: {pattern}"
      }

  RESTITUISCI {failed: False}
```

3.7 Monitoraggio Risorse

Tracciamento Risorse in Tempo Reale:

```
ResourceMonitor {
  limits: {
    max_time: float,  // secondi
    max_cost: float,  // dollari
    max_tokens: int,
```

```
    max_llm_calls: int,
    max_tool_calls: int
  },

  current_usage: {
    time_elapsed: float,
    cost_accumulated: float,
    tokens_used: int,
    llm_calls_made: int,
    tool_calls_made: int
  },

  start_time: datetime
}

Funzione MONITOR_RESOURCES(monitor, operation, estimated_cost):

  # Verifica se l'operazione supererebbe qualsiasi limite
  violations = []

  # Verifica tempo
  SE monitor.current_usage.time_elapsed >= monitor.limits.max_time:
    violations.append("Limite tempo superato")

  # Verifica costo
  SE monitor.current_usage.cost_accumulated + estimated_cost.cost > monitor.limits.max_cost:
    violations.append("Limite costo verrebbe superato")

  # Verifica token
  SE monitor.current_usage.tokens_used + estimated_cost.tokens > monitor.limits.max_tokens:
    violations.append("Limite token verrebbe superato")

  # Verifica conteggio chiamate
  SE operation.type == "LLM_CALL":
    SE monitor.current_usage.llm_calls_made >= monitor.limits.max_llm_calls:
      violations.append("Limite chiamate LLM superato")

  SE operation.type == "TOOL_CALL":
    SE monitor.current_usage.tool_calls_made >= monitor.limits.max_tool_calls:
      violations.append("Limite chiamate tool superato")

  # Se ci sono violazioni, rifiuta operazione
  SE violations:
    RESTITUISCI REJECT("Violazione limite risorse", violations)

  # Altrimenti, approva e aggiorna uso
  monitor.current_usage.time_elapsed = (now - monitor.start_time).seconds
  monitor.current_usage.cost_accumulated += estimated_cost.cost
```

```
  monitor.current_usage.tokens_used += estimated_cost.tokens

  SE operation.type == "LLM_CALL":
    monitor.current_usage.llm_calls_made += 1
  SE operation.type == "TOOL_CALL":
    monitor.current_usage.tool_calls_made += 1

  RESTITUISCI APPROVE()

Funzione GET_RESOURCE_STATUS(monitor):
  RESTITUISCI {
    time: {
      used: monitor.current_usage.time_elapsed,
      limit: monitor.limits.max_time,
      remaining: monitor.limits.max_time - monitor.current_usage.time_elapsed,
      percentage: (monitor.current_usage.time_elapsed / monitor.limits.max_time) * 100
    },
    cost: {
      used: monitor.current_usage.cost_accumulated,
      limit: monitor.limits.max_cost,
      remaining: monitor.limits.max_cost - monitor.current_usage.cost_accumulated,
      percentage: (monitor.current_usage.cost_accumulated / monitor.limits.max_cost) * 100
    },
    // Simile per tokens, llm_calls, tool_calls
  }
```

3.8 Audit Logging

Audit Trail Completo:

```
AuditLog Entry {
  timestamp: datetime,
  event_type: "INPUT_VALIDATION" | "ACTION_AUTHORIZATION" | "OUTPUT_VALIDATION" | "RESOURCE_CHE
  decision: "APPROVED" | "REJECTED" | "REQUIRES_APPROVAL",

  // Contesto
  task_id: string,
  execution_phase: string,

  // Cosa è stato verificato
  subject: {...},  // La cosa che viene validata
  bounds_applied: [BoundReference],

  // Dettagli della decisione
  reason: string,
  violations: [Violation] | null,
  warnings: [Warning] | null,
```

```
  // Se coinvolge approvazione umana
  approval_request_id: string | null,
  human_decision: "APPROVED" | "DENIED" | null,
  human_reason: string | null,

  // Metadata
  verifier_version: string,
  rule_version: string
}

Funzione LOG_SAFETY_DECISION(event_type, decision, details):
  entry = AuditLogEntry {
    timestamp: NOW(),
    event_type: event_type,
    decision: decision,
    ...details
  }

  # Scrivi nell'audit log (append-only, immutabile)
  AUDIT_LOG.append(entry)

  # Se rifiuto o violazione, genera anche alert
  SE decision == "REJECTED":
    ALERT_SECURITY_TEAM(entry)

  # Se richiesta approvazione umana, traccia
  SE decision == "REQUIRES_APPROVAL":
    TRACK_APPROVAL_REQUEST(entry)
```

3.9 Operazioni del Safety Verifier

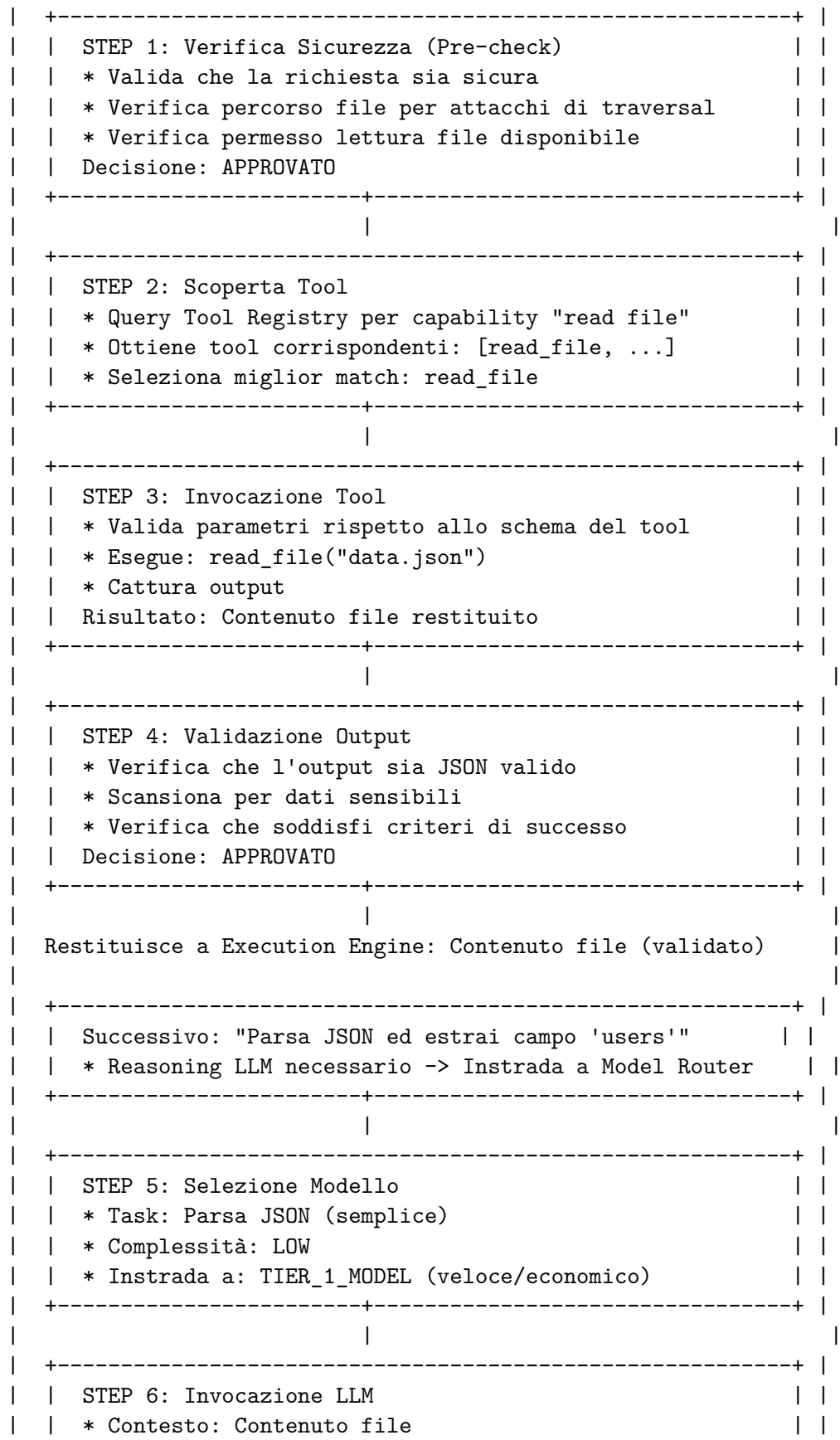
```
1. VALIDATE_INPUT(input, bounds, context)
   * Validazione multi-layer
   * Schema, dimensione, injection, contenuto
   * Restituisce risultato validazione

2. AUTHORIZE_ACTION(action, context)
   * Verifica permessi
   * Valutazione pericolo
   * Approvazione umana se necessaria
   * Restituisce decisione autorizzazione

3. VALIDATE_OUTPUT(output, schema, criteria, bounds)
   * Validazione schema
   * Verifica criteri di successo
   * Verifiche sicurezza (segreti, codice malevolo)
   * Restituisce risultato validazione
```

```
4. MONITOR_RESOURCES(monitor, operation, estimated_cost)
   * Verifica limiti risorse
   * Aggiorna tracciamento uso
   * Interrompe se limite superato
   * Restituisce approvazione/rifiuto

5. REQUEST_HUMAN_APPROVAL(action, context)
   * Presenta all'umano
   * Attende decisione
   * Registra decisione
   * Restituisce risultato approvazione

6. GET_RESOURCE_STATUS(monitor)
   * Restituisce uso corrente risorse
   * Budget rimanenti
   * Percentuale utilizzata

7. LOG_SAFETY_DECISION(event_type, decision, details)
   * Crea voce audit log
   * Allerta se violazione
   * Traccia per forensics

8. UPDATE_BOUNDS(bound_updates)
   * Aggiorna safety bounds
   * Incremento versione
   * Valida nuovi bounds
   * Propaga modifiche

9. GET_VIOLATIONS(time_range, filters)
   * Query sull'audit log
   * Restituisce violazioni corrispondenti ai criteri
   * Per analisi sicurezza

10. GENERATE_SAFETY_REPORT(time_range)
    * Analizza audit log
    * Calcola statistiche
    * Identifica pattern
    * Restituisce report stato sicurezza
```

4. Integrazione del Capability Layer

4.1 Flusso di Interazione Tipico

```
+-------------------------------------------------------------+
|      CAPABILITY LAYER: FLUSSO DI INTERAZIONE COMPLETO       |
|                                                             |
|  Execution Engine: "Ho bisogno di leggere file 'data.json'" |
|    |                                                        |
```


STEP 1: Verifica Sicurezza (Pre-check)
* Valida che la richiesta sia sicura
* Verifica percorso file per attacchi di traversal
* Verifica permesso lettura file disponibile
Decisione: APPROVATO
STEP 2: Scoperta Tool
* Query Tool Registry per capability "read file"
* Ottiene tool corrispondenti: [read_file, ...]
* Seleziona miglior match: read_file
STEP 3: Invocazione Tool
* Valida parametri rispetto allo schema del tool
* Esegue: read_file("data.json")
* Cattura output
Risultato: Contenuto file restituito
STEP 4: Validazione Output
* Verifica che l'output sia JSON valido
* Scansiona per dati sensibili
* Verifica che soddisfi criteri di successo
Decisione: APPROVATO
Restituisce a Execution Engine: Contenuto file (validato)
Successivo: "Parsa JSON ed estrai campo 'users'"
* Reasoning LLM necessario -> Instrada a Model Router
STEP 5: Selezione Modello
* Task: Parsa JSON (semplice)
* Complessità: LOW
* Instrada a: TIER_1_MODEL (veloce/economico)
STEP 6: Invocazione LLM
* Contesto: Contenuto file

```
|  |  * Prompt: "Estrai campo 'users'"                          | |
|  |  * Invoca modello con retry/fallback                       | |
|  |  Risultato: Lista utenti estratta                          | |
|  +------------------------+-----------------------------------+ |
|                           |                                     |
|  +--------------------------------------------------------+ |
|  |  STEP 7: Monitoraggio Risorse                          | |
|  |  * Aggiorna tempo trascorso                            | |
|  |  * Aggiorna costo ($0.001 per chiamata LLM)            | |
|  |  * Aggiorna conteggio token                            | |
|  |  * Verifica tutti entro limiti [v]                       | |
|  +------------------------+-----------------------------------+ |
|                           |                                     |
|  Restituisce a Execution Engine: Lista utenti (validata,       |
|                                  entro budget)                 |
+----------------------------------------------------------------+
```

4.2 Caratteristiche di Prestazioni

Metriche del Capability Layer:

```
TOOL REGISTRY
  * Lookup tool: <10ms (cached)
  * Overhead invocazione tool: <50ms
  * Verifica permessi: <5ms
  * Tool supportati totali: 50-200 (tipico)

MODEL ROUTER
  * Selezione modello: <20ms
  * Invocazione modello: 1-15s (dipende dal tier del modello)
  * Ottimizzazione costi: riduzione 5-10x vs sempre-miglior-modello
  * Gestione fallback: <2s latenza aggiuntiva

SAFETY VERIFIER
  * Validazione input: <50ms
  * Autorizzazione azioni: <30ms
  * Validazione output: <100ms
  * Audit logging: <10ms (async)
  * Attesa approvazione umana: 30s - 5min (dipendente dall'umano)
```

Prossimo: 05-infrastructure.md -> Specifiche di Observability, Resource Management, Error Handling