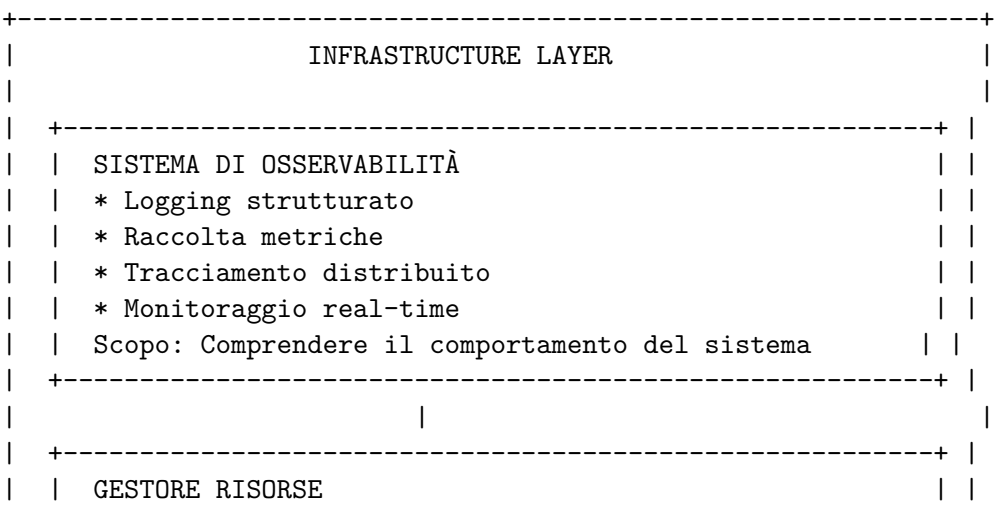
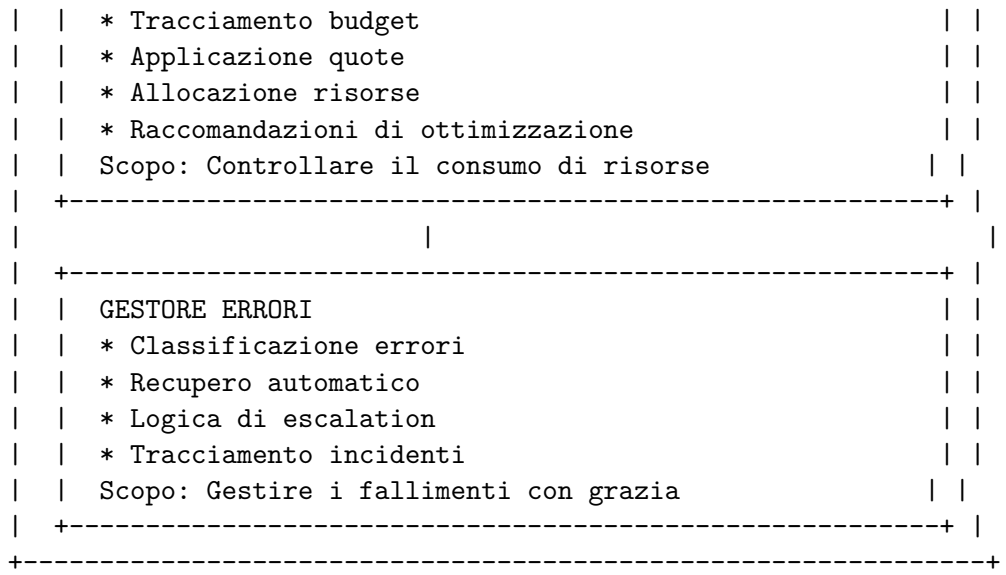
Contents

Infrastructure Layer: Osservabilità, Risorse e Gestione degli Errori

Panoramica

L'Infrastructure Layer fornisce i servizi fondazionali che supportano tutti gli altri layer: osservabilità per capire cosa sta accadendo, gestione delle risorse per controllare i consumi, e gestione degli errori per trattare i fallimenti in modo robusto.





1. Sistema di Osservabilità

1.1 Scopo e Responsabilità

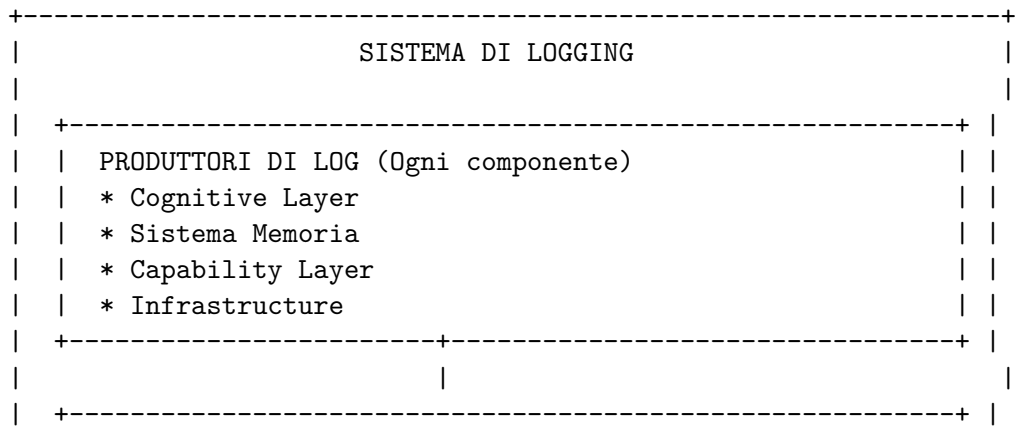
Funzione Principale: Rendere il sistema trasparente - capire cosa sta accadendo, perché, e come sta performando.

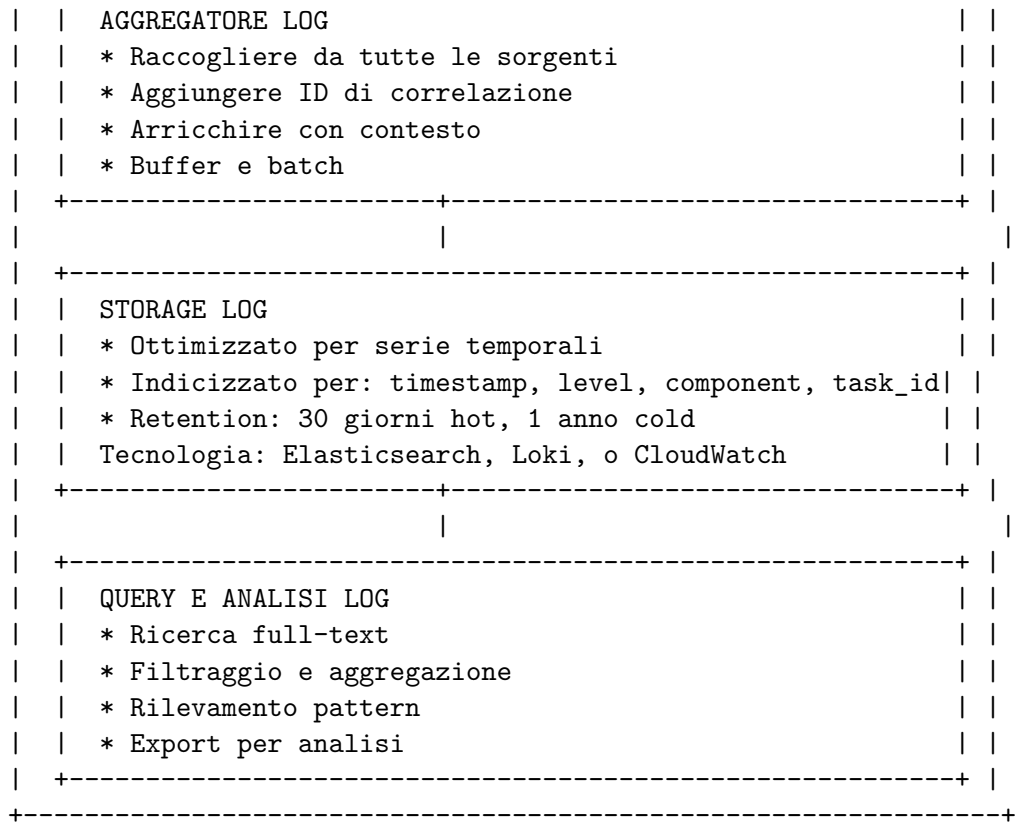
Tre Pilastri dell'Osservabilità: 1. **Log**: Eventi discreti con contesto 2. **Metriche**: Misurazioni quantitative aggregate 3. **Tracce**: Flusso di esecuzione end-to-end

Responsabilità: 1. **Logging Strutturato**: Registrare eventi con metadata strutturati 2. **Raccolta Metriche**: Raccogliere metriche di performance/business 3. **Tracciamento Distribuito**: Tracciare richieste attraverso componenti 4. **Alerting**: Notificare anomalie e problemi 5. **Dashboarding**: Visualizzare lo stato del sistema 6. **Supporto Debugging**: Fornire informazioni per troubleshooting

1.2 Sistema di Logging

Architettura Logging Strutturato:





Schema Entry Log:

```
LogEntry {
  // Identificazione
  timestamp: datetime (ISO 8601, UTC),
  log_id: string (UUID),

  // Categorizzazione
  level: "DEBUG" | "INFO" | "WARNING" | "ERROR" | "CRITICAL",
  component: string,  // es. "PlanningEngine", "ModelRouter"
  event_type: string,  // es. "task_started", "tool_invoked"

  // Correlazione
  task_id: string,
  session_id: string,
  user_id: string,
  trace_id: string,  // Per tracciamento distribuito
  span_id: string,

  // Contenuto
  message: string,
  structured_data: {
    // Dati specifici dell'evento
    // Esempi:
    // - tool_name, parameters per invocazioni tool
```

```
    // - model_id, tokens_used per chiamate LLM
    // - error_code, stack_trace per errori
  },

  // Contesto
  context: {
    execution_phase: string,
    parent_task: string,
    resource_usage: {...}
  },

  // Metadata
  host: string,
  process_id: string,
  thread_id: string,
  version: string
}
```

Livelli di Log e Utilizzo:

```
+-----------------------------------------------------------+
|                    LIVELLI DI LOG                         |
|                                                           |
|  DEBUG (Solo sviluppo)                                    |
|  * Info diagnostiche dettagliate                         |
|  * Valori variabili, stato interno                       |
|  * Esempio: "Dimensione working memory: 15234 token"     |
|  -> Non in produzione (troppo verbose)                    |
|                                                           |
|  INFO (Operazioni normali)                                |
|  * Eventi significativi                                  |
|  * Transizioni di stato                                  |
|  * Esempio: "Task T-123 completato con successo"         |
|  -> Livello default in produzione                         |
|                                                           |
|  WARNING (Potenziali problemi)                            |
|  * Performance degradata                                 |
|  * Errori recuperabili                                   |
|  * Esempio: "Model Router fallback a tier 2"             |
|  -> Investigare se frequenti                              |
|                                                           |
|  ERROR (Fallimenti)                                       |
|  * Fallimenti operazioni                                 |
|  * Errori inaspettati                                    |
|  * Esempio: "Invocazione tool fallita: timeout"          |
|  -> Richiede investigazione                               |
|                                                           |
|  CRITICAL (Fallimenti a livello di sistema)               |
|  * Servizio non disponibile                              |
```

```
|  * Corruzione dati                                          |
|  * Esempio: "Sistema memoria non raggiungibile"             |
|  -> Azione immediata richiesta                               |
+-------------------------------------------------------------+
```

1.3 Sistema Metriche

Architettura Raccolta Metriche:

```
+-------------------------------------------------------------------+
|                     SISTEMA METRICHE                              |
|                                                                   |
|  +-------------------------------------------------------------+ |
|  |  STRUMENTAZIONE METRICHE                                    | |
|  |  * Contatori (solo incremento)                              | |
|  |  * Gauge (valore corrente)                                  | |
|  |  * Istogrammi (distribuzioni)                               | |
|  |  * Timer (durate)                                           | |
|  +------------------------+------------------------------------+ |
|                           |                                       |
|  +-------------------------------------------------------------+ |
|  |  AGGREGATORE METRICHE                                       | |
|  |  * Raccogliere da tutti i componenti                        | |
|  |  * Aggregare a intervalli 1min, 5min, 1h                  | | |
|  |  * Calcolare percentili (p50, p95, p99)                     | |
|  |  * Downsampling per storage long-term                       | |
|  +------------------------+------------------------------------+ |
|                           |                                       |
|  +-------------------------------------------------------------+ |
|  |  DATABASE TIME-SERIES                                       | |
|  |  * Memorizzare metriche con timestamp                       | |
|  |  * Indicizzato per nome metrica + etichette                 | |
|  |  * Retention: 1 settimana raw, 1 mese aggregato, 1 anno | | |
|  |  Tecnologia: Prometheus, InfluxDB, o CloudWatch             | |
|  +------------------------+------------------------------------+ |
|                           |                                       |
|  +-------------------------------------------------------------+ |
|  |  DASHBOARD E ALERT                                          | |
|  |  * Visualizzazione real-time                                | |
|  |  * Rilevamento anomalie                                     | |
|  |  * Alert basati su soglie                                   | |
|  |  Tecnologia: Grafana, Datadog, o custom                     | |
|  +-------------------------------------------------------------+ |
+-------------------------------------------------------------------+
```

Categorie Metriche Chiave:

```
+-----------------------------------------------------+
|                  CATEGORIE METRICHE                  |
```

```
|                                                          |
|  1. METRICHE THROUGHPUT                                  |
|     * tasks_started (contatore)                         |
|     * tasks_completed (contatore)                        |
|     * tasks_failed (contatore)                           |
|     * tasks_per_minute (gauge)                           |
|     Etichette: [user_id, task_type, complexity]          |
|                                                          |
|  2. METRICHE LATENZA                                     |
|     * task_duration_seconds (istogramma)                 |
|     * goal_analysis_duration (istogramma)                |
|     * planning_duration (istogramma)                     |
|     * execution_duration (istogramma)                    |
|     * reflection_duration (istogramma)                   |
|     Percentili: p50, p90, p95, p99                       |
|                                                          |
|  3. METRICHE COSTO                                       |
|     * llm_cost_dollars (contatore)                       |
|     * tokens_consumed (contatore)                        |
|     * tool_invocation_cost (contatore)                   |
|     * cost_per_task (gauge)                              |
|     Etichette: [model_id, tool_name]                     |
|                                                          |
|  4. METRICHE QUALITÀ                                     |
|     * success_rate (gauge)                               |
|     * verification_pass_rate (gauge)                     |
|     * human_satisfaction_score (gauge)                   |
|     * retry_rate (gauge)                                 |
|                                                          |
|  5. METRICHE RISORSE                                     |
|     * memory_usage_mb (gauge)                            |
|     * cpu_utilization_percent (gauge)                    |
|     * active_tasks (gauge)                               |
|     * queue_depth (gauge)                                |
|                                                          |
|  6. METRICHE COMPONENTI                                  |
|     * model_router_calls (contatore)                     |
|     * tool_registry_lookups (contatore)                  |
|     * episodic_memory_queries (contatore)                |
|     * pattern_cache_hits (contatore)                     |
|     * safety_violations (contatore)                      |
+----------------------------------------------------------+
```

Convenzione Naming Metriche:

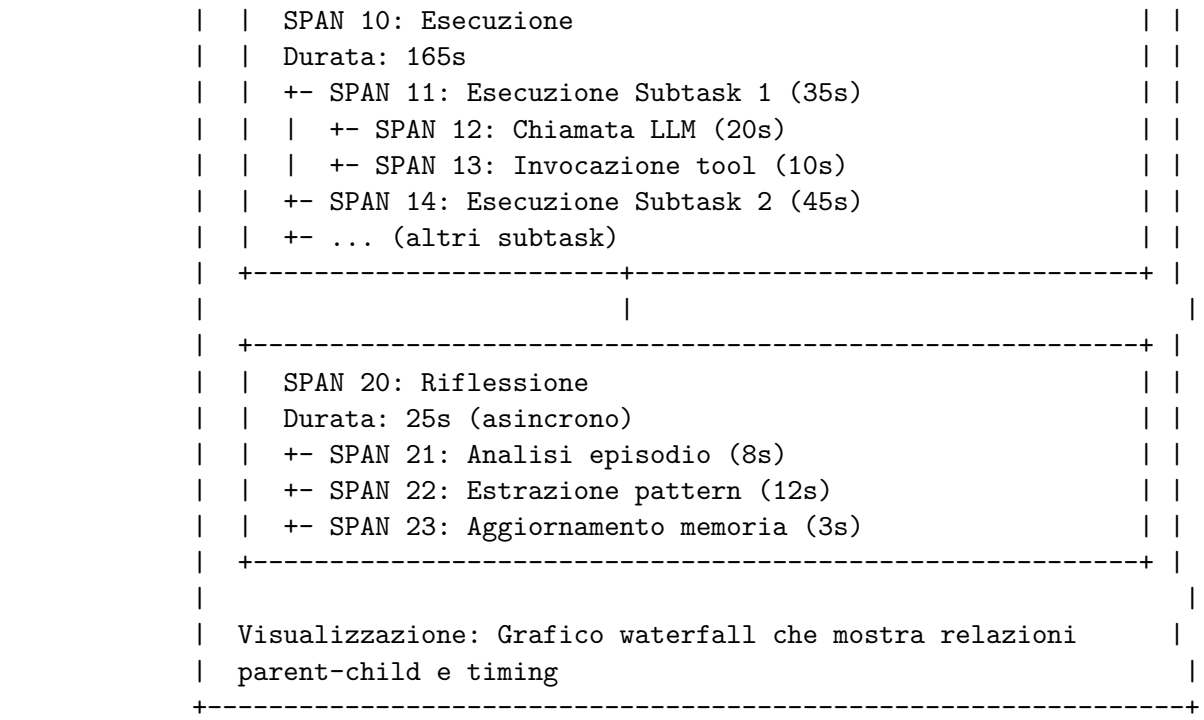
```
<componente>_<metrica>_<unità>

Esempi:
- planning_engine_duration_seconds
```

```
- model_router_cost_dollars
- safety_verifier_rejections_total
- memory_system_cache_hit_ratio

Etichette per dimensionalità:
{
  component="PlanningEngine",
  strategy="HTN",
  complexity="moderate",
  user_id="user_123"
}
```

1.4 Tracciamento Distribuito

Architettura Tracciamento:

```
+-------------------------------------------------------------+
|                   TRACCIAMENTO DISTRIBUITO                  |
|                                                             |
|  Richiesta: "Utente chiede: Refactoring autenticazione"     |
|    |                                                        |
|  +-----------------------------------------------------+    |
|  |  TRACE ROOT (Span 1)                                |    |
|  |  Operazione: handle_task                            |    |
|  |  trace_id: abc123                                   |    |
|  |  span_id: span-1                                    |    |
|  |  Inizio: 2024-01-15 10:00:00                        |    |
|  |  Durata: 245s                                       |    |
|  +------------------------+----------------------------+    |
|                           |                                 |
|  +-----------------------------------------------------+    |
|  |  SPAN 2: Analisi Obiettivi                          |    |
|  |  parent_span: span-1                                |    |
|  |  Durata: 18s                                        |    |
|  |  +- SPAN 3: Parsing semantico (5s)                  |    |
|  |  +- SPAN 4: Estrazione goal (8s)                    |    |
|  |  +- SPAN 5: Classificazione complessità (3s)        |    |
|  +------------------------+----------------------------+    |
|                           |                                 |
|  +-----------------------------------------------------+    |
|  |  SPAN 6: Pianificazione                             |    |
|  |  Durata: 42s                                        |    |
|  |  +- SPAN 7: Query pattern cache (2s)                |    |
|  |  +- SPAN 8: Decomposizione task (30s)               |    |
|  |  +- SPAN 9: Analisi dipendenze (8s)                 |    |
|  +------------------------+----------------------------+    |
|                           |                                 |
|  +-----------------------------------------------------+    |
```



Schema Span:

```
Span {
  // Identificazione
  trace_id: string,  // Stesso per l'intera richiesta
  span_id: string,   // Unico per span
  parent_span_id: string | null,

  // Operazione
  operation_name: string,  // es. "PlanningEngine.generate_plan"
  component: string,

  // Timing
  start_time: datetime,
  end_time: datetime,
  duration_ms: int,

  // Contesto
  tags: {
    // Coppie chiave-valore per filtraggio
    task_type: string,
    model_id: string,
    user_id: string,
    ...
  },

  // Eventi all'interno dello span
  events: [
```

```
    {
      timestamp: datetime,
      name: string,
      attributes: {...}
    }
  ],

  // Esito
  status: "OK" | "ERROR",
  error: Error | null
}
```

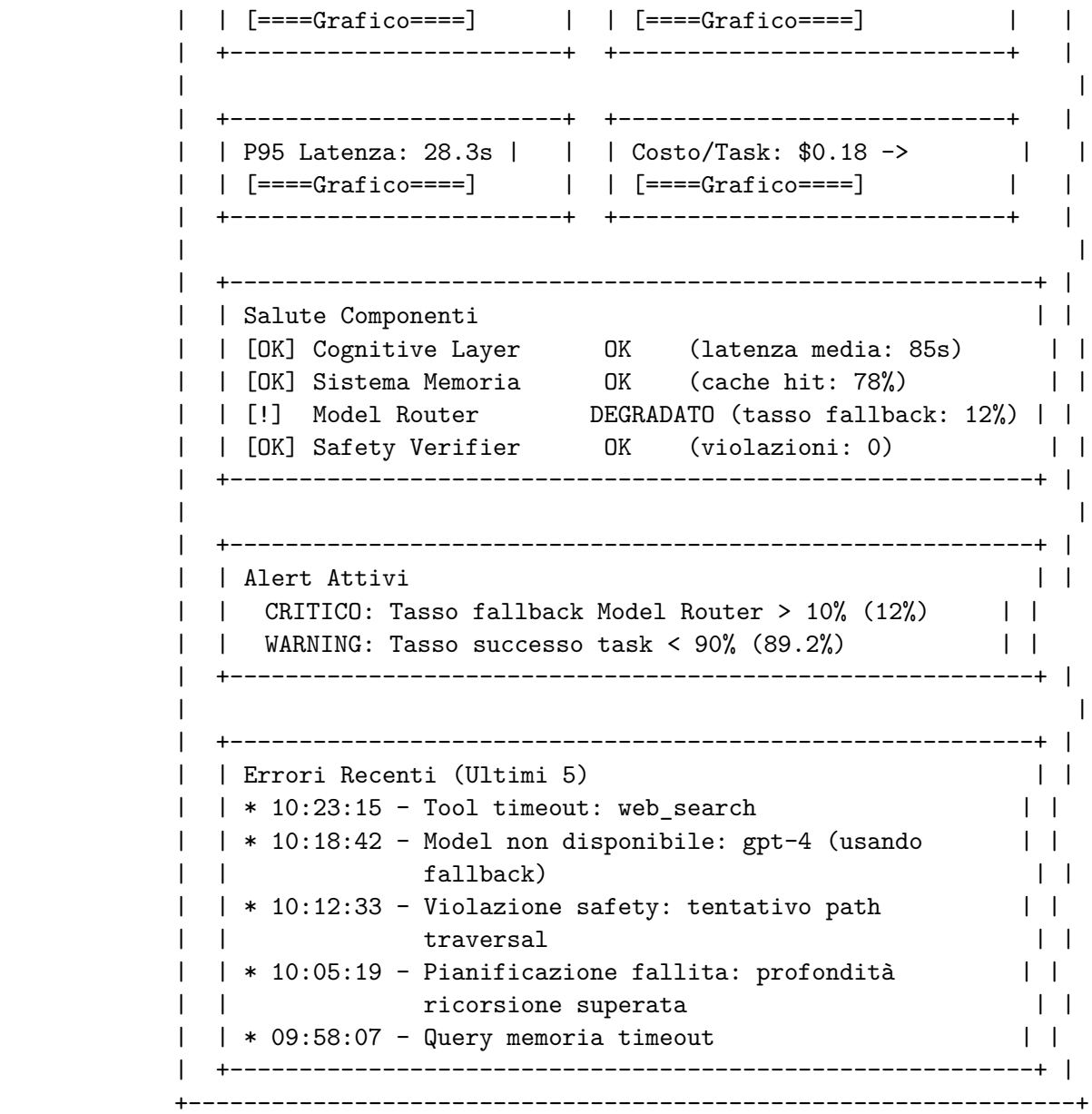
Strategia Campionamento Tracce:

```
Traffico Alto -> Non si può tracciare tutto -> Campionamento

STRATEGIE DI CAMPIONAMENTO:

1. BASATO SU PROBABILITÀ
   * Campionare X% di tutte le tracce casualmente
   * Esempio: 10% campionamento
   * Pro: Statisticamente rappresentativo
   * Contro: Potrebbe perdere problemi rari

2. LIMITATO PER RATE
   * Campionare max N tracce al secondo
   * Esempio: 100 tracce/sec
   * Pro: Controllo costo storage
   * Contro: Potrebbe perdere dettagli durante picchi

3. TAIL-BASED (Campionamento Intelligente)
   * Mantenere tutti gli errori
   * Mantenere tracce lente (>p95 latenza)
   * Campionare altre a basso rate
   * Esempio: 100% errori, 100% >p95, 1% altre
   * Pro: Catturare tracce interessanti
   * Contro: Logica più complessa

RACCOMANDATO: Campionamento tail-based
```

1.5 Dashboard di Monitoraggio

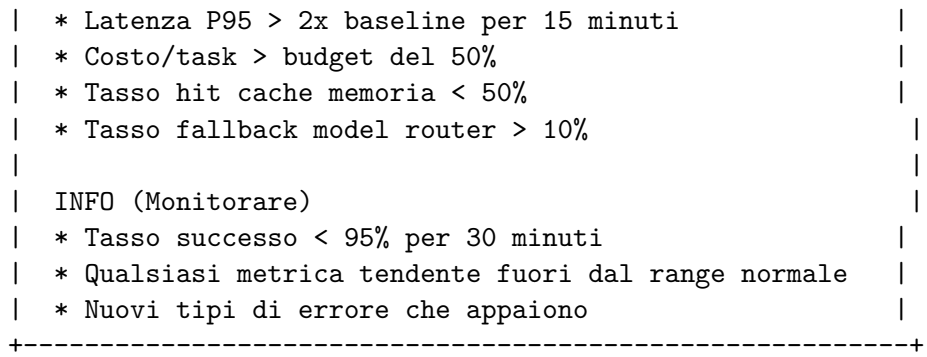
Dashboard Salute Sistema:

```
+-------------------------------------------------------------+
|                  DASHBOARD SALUTE SISTEMA                   |
|                                                             |
|  +-----------------------+  +----------------------------+  |
|  | Task/Min: 12.5 |      |  | Tasso Successo: 89.2% |    |  |
```



Regole Alert:

```
+-----------------------------------------------------------+
|                    REGOLE ALERT                           |
|                                                           |
|  CRITICO (Avvisare immediatamente)                        |
|  * Tasso successo < 70% per 5 minuti                     |
|  * Latenza P95 > 5x baseline per 10 minuti               |
|  * Tasso errori > 50% per 5 minuti                       |
|  * Qualsiasi componente completamente non disponibile     |
|  * Violazioni safety > 10 al minuto                       |
|                                                           |
|  WARNING (Investigare entro 1 ora)                        |
|  * Tasso successo < 90% per 15 minuti                    |
```



2. Gestore Risorse

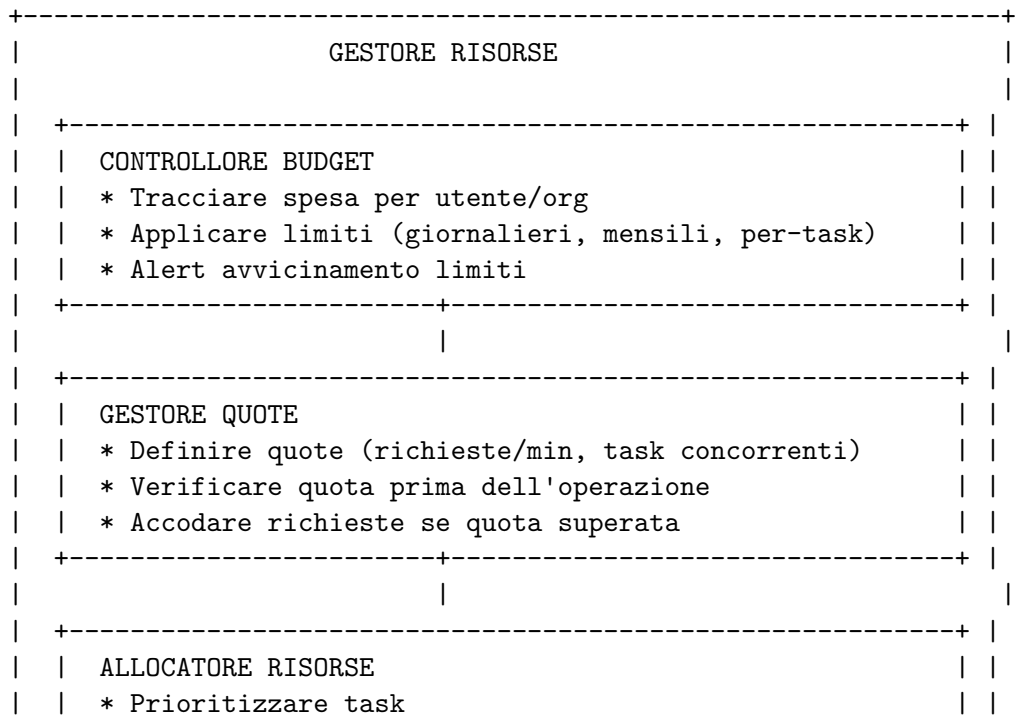
2.1 Scopo e Responsabilità

Funzione Principale: Controllare e ottimizzare il consumo di risorse (tempo, costo, memoria, compute).

Insight Chiave: Senza gestione delle risorse, l'agente può: - Spendere l'intero budget su un singolo task - Eseguire indefinitamente (denial of service) - Esaurire la memoria - Causare rate limiting su API esterne

Responsabilità: 1. **Tracciamento Budget**: Monitorare spesa rispetto ai limiti 2. **Applicazione Quote**: Applicare quote per-utente, per-task 3. **Allocazione Risorse**: Distribuire risorse ottimalmente 4. **Throttling**: Limitare il rate quando necessario 5. **Ottimizzazione**: Suggerire miglioramenti per efficienza

2.2 Architettura Gestore Risorse



```
|  |  * Allocare risorse compute                                  | |
|  |  * Load balancing                                            | |
|  +-----------------------+--------------------------------+ |
|                          |                                    |
|  +-----------------------------------------------------------+ |
|  |  MOTORE DI OTTIMIZZAZIONE                                  | |
|  |  * Analizzare pattern di utilizzo risorse                  | |
|  |  * Identificare sprechi                                    | |
|  |  * Raccomandare miglioramenti                              | |
|  +-----------------------------------------------------------+ |
+-----------------------------------------------------------------+
```

2.3 Sistema Budget

Gerarchia Budget:

```
+-------------------------------------------------------------+
|                   GERARCHIA BUDGET                          |
|                                                             |
|  Budget Organizzazione (Livello Top)                        |
|  +- $10.000 / mese                                          |
|     |                                                        |
|     +- Budget Utente (Per Utente)                           |
|     |  +- $500 / mese                                       |
|     |     |                                                 |
|     |     +- Budget Task (Per Task)                         |
|     |     |  +- $1.00 / task (limite soft)                  |
|     |     |     +- $5.00 / task (limite hard)               |
|     |     |                                                 |
|     |     +- Budget Giornaliero                             |
|     |        +- $20 / giorno                                |
|     |                                                       |
|     +- Budget Servizio (Per Tipo Servizio)                  |
|        +- LLM: $7.000 / mese                                |
|        +- Tool: $2.000 / mese                               |
|        +- Infrastructure: $1.000 / mese                     |
+-------------------------------------------------------------+
```

Logica Applicazione Budget:

```
Funzione CHECK_BUDGET(operazione, costo_stimato, contesto):

  # LIVELLO 1: Verificare budget organizzazione
  org_rimanente = org_budget.limite_mensile - org_budget.speso_questo_mese
  SE costo_stimato > org_rimanente:
    SE org_budget.consenti_eccedenza:
      LOG_WARNING("Budget organizzazione superato, consentendo eccedenza")
    ALTRIMENTI:
      RITORNA RIGETTA("Budget organizzazione esaurito")
```

```
  # LIVELLO 2: Verificare budget utente
  utente_rimanente = utente_budget.limite_mensile - utente_budget.speso_questo_mese
  SE costo_stimato > utente_rimanente:
    RITORNA RIGETTA("Budget mensile utente esaurito")

  # LIVELLO 3: Verificare budget giornaliero
  giornaliero_rimanente = utente_budget.limite_giornaliero - utente_budget.speso_oggi
  SE costo_stimato > giornaliero_rimanente:
    RITORNA RIGETTA("Budget giornaliero esaurito, riprova domani")

  # LIVELLO 4: Verificare limite soft per-task
  SE costo_stimato > task_budget.limite_soft:
    SE costo_stimato < task_budget.limite_hard:
      # Richiedere approvazione per superare limite soft
      approvazione = RICHIEDI_APPROVAZIONE("Costo stimato ${costo_stimato} supera limite soft $
      SE NON approvazione:
        RITORNA RIGETTA("Limite soft superato, approvazione negata")
    ALTRIMENTI:
      RITORNA RIGETTA("Limite hard verrebbe superato")

  # Tutti i controlli passati
  RITORNA APPROVA()

Funzione REGISTRA_COSTO_EFFETTIVO(task_id, costo_effettivo):
  # Aggiornare tutti i livelli di budget
  org_budget.speso_questo_mese += costo_effettivo
  utente_budget.speso_questo_mese += costo_effettivo
  utente_budget.speso_oggi += costo_effettivo

  # Se speso più rispetto alla stima, analizzare
  stima = task_budget.stime[task_id]
  SE costo_effettivo > stima * 1.5:
    ANALIZZA_ECCESSO_SPESA(task_id, stima, costo_effettivo)
```

2.4 Sistema Quote

Tipi di Quote:

```
+-----------------------------------------------------------+
|                    TIPI DI QUOTE                          |
|                                                           |
|  QUOTE RATE (Richieste per periodo di tempo)              |
|  * tasks_per_minute: 10                                  |
|  * llm_calls_per_minute: 100                             |
|  * tool_invocations_per_minute: 50                       |
|  Scopo: Prevenire rate limiting API, DoS                 |
|                                                           |
```

```
|  QUOTE CONCORRENZA (Operazioni parallele)                |
|  * max_concurrent_tasks: 5                               |
|  * max_concurrent_llm_calls: 10                          |
|  Scopo: Prevenire esaurimento risorse                    |
|                                                           |
|  QUOTE VOLUME (Quantità totale)                          |
|  * max_tasks_per_day: 1000                               |
|  * max_tokens_per_month: 10M                             |
|  Scopo: Prevenire abuso, controllare costi               |
|                                                           |
|  QUOTE DIMENSIONE (Limiti per-item)                      |
|  * max_task_duration: 600s (10 min)                      |
|  * max_context_size: 200K token                          |
|  * max_output_size: 100KB                                |
|  Scopo: Prevenire operazioni fuori controllo             |
+----------------------------------------------------------+
```

Applicazione Quote con Accodamento:

```
Funzione APPLICA_QUOTA(tipo_operazione, user_id):

  quota = OTTIENI_QUOTA(tipo_operazione, user_id)
  utilizzo_corrente = OTTIENI_UTILIZZO_CORRENTE(tipo_operazione, user_id)

  SE utilizzo_corrente < quota.limite:
    # Sotto quota, consentire immediatamente
    INCREMENTA_UTILIZZO(tipo_operazione, user_id)
    RITORNA CONSENTI()

  ALTRIMENTI:
    # Quota superata
    SE quota.consenti_accodamento:
      # Aggiungere a coda, sarà processata quando quota disponibile
      posizione_coda = ACCODA(operazione, user_id)
      RITORNA ACCODATO(posizione=posizione_coda, attesa_stimata=...)
    ALTRIMENTI:
      # Rigettare immediatamente
      RITORNA RIGETTA("Quota superata", riprova_dopo=...)
```

2.5 Ottimizzazione Risorse

Analizzatore Ottimizzazione:

```
Funzione ANALIZZA_UTILIZZO_RISORSE(periodo_tempo):

  task = OTTIENI_TASK_IN_PERIODO(periodo_tempo)

  # ANALISI 1: Efficienza costo
  analisi_costo = {
```

```
    costo_totale: SOMMA(task.costo per task in task),
    costo_medio_per_task: MEDIA(task.costo per task in task),
    costo_per_componente: RAGGRUPPA_PER(task, 'componente', SOMMA('costo')),

    # Identificare outlier costosi
    task_costosi: task DOVE costo > PERCENTILE(task.costo, 95),

    # Efficienza routing modello
    risparmio_routing_modello: STIMA_RISPARMIO_DA_ROUTING(task)
  }

  # ANALISI 2: Efficienza tempo
  analisi_tempo = {
    tempo_totale: SOMMA(task.durata per task in task),
    tempo_medio_per_task: MEDIA(task.durata per task in task),

    # Colli di bottiglia
    colli_bottiglia: IDENTIFICA_COLLI_BOTTIGLIA(task),

    # Opportunità parallelizzazione mancate
    potenziale_parallelizzazione: TROVA_OPPORTUNITA_PARALLELIZZAZIONE(task)
  }

  # ANALISI 3: Utilizzo risorse
  analisi_utilizzo = {
    cpu_media: MEDIA(campione.cpu per campione in metriche),
    memoria_media: MEDIA(campione.memoria per campione in metriche),

    # Sotto/sovra provisionamento
    tasso_utilizzo_cpu: cpu_media / cpu_allocata,
    tasso_utilizzo_memoria: memoria_media / memoria_allocata
  }

  # RACCOMANDAZIONI
  raccomandazioni = GENERA_RACCOMANDAZIONI(
    analisi_costo,
    analisi_tempo,
    analisi_utilizzo
  )

  RITORNA ReportOttimizzazione {
    analisi: {...},
    raccomandazioni: raccomandazioni,
    risparmio_potenziale: STIMA_RISPARMIO_POTENZIALE(raccomandazioni)
  }

Funzione GENERA_RACCOMANDAZIONI(analisi_costo, analisi_tempo, analisi_utilizzo):
```

```
  raccomandazioni = []

  # Ottimizzazione costo
  SE risparmio_routing_modello.potenziale > 0.2:  # 20%+ risparmio possibile
    raccomandazioni.aggiungi({
      tipo: "OTTIMIZZAZIONE_COSTO",
      titolo: "Migliorare routing modello",
      descrizione: f"Il routing corrente potrebbe risparmiare {risparmio_routing_modello.potenz
      risparmio_potenziale: "$X/mese"
    })

  # Ottimizzazione tempo
  PER collo_bottiglia IN analisi_tempo.colli_bottiglia:
    raccomandazioni.aggiungi({
      tipo: "OTTIMIZZAZIONE_TEMPO",
      titolo: f"Ottimizzare {collo_bottiglia.componente}",
      descrizione: f"{collo_bottiglia.componente} richiede {collo_bottiglia.tempo_medio}s in me
      azioni: collo_bottiglia.azioni_suggerite
    })

  # Utilizzo risorse
  SE analisi_utilizzo.tasso_utilizzo_cpu < 0.3:
    raccomandazioni.aggiungi({
      tipo: "OTTIMIZZAZIONE_RISORSE",
      titolo: "Ridurre allocazione CPU",
      descrizione: "Utilizzo CPU è solo 30%, è possibile ridurre allocazione per risparmiare co
    })

  RITORNA raccomandazioni
```

3. Gestore Errori

3.1 Scopo e Responsabilità

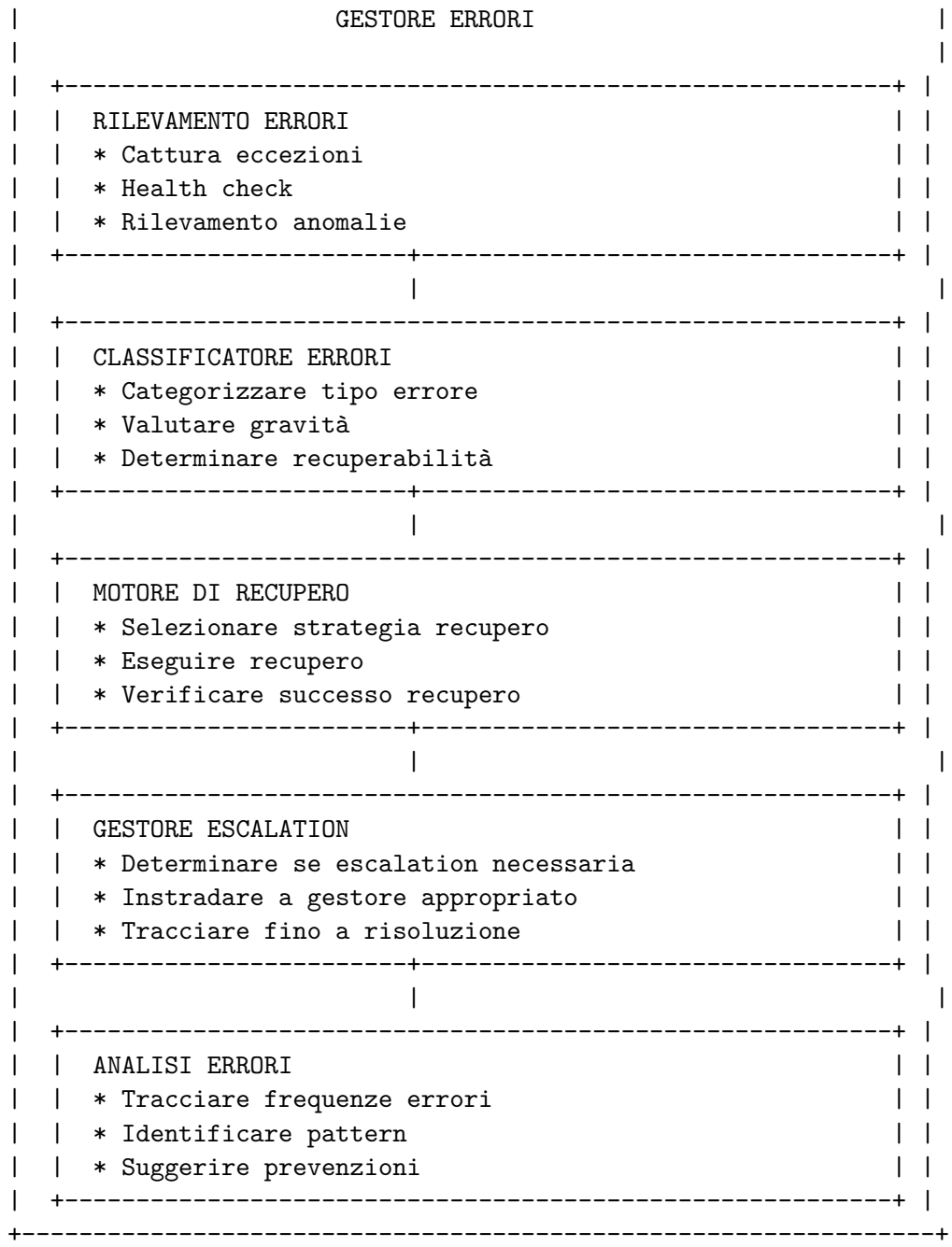
Funzione Principale: Gestire i fallimenti in modo robusto - rilevare, classificare, recuperare quando possibile, escalare quando necessario.

Filosofia: Gli errori sono inevitabili. L'obiettivo è la degradazione graduale, non l'affidabilità perfetta.

Responsabilità: 1. **Rilevamento Errori**: Catturare errori da tutti i componenti 2. **Classificazione Errori**: Categorizzare per tipo e gravità 3. **Recupero Automatico**: Applicare strategie di recupero 4. **Escalation**: Instradare a umano quando impossibile auto-recuperare 5. **Apprendimento**: Tracciare pattern di errori per migliorare nel tempo

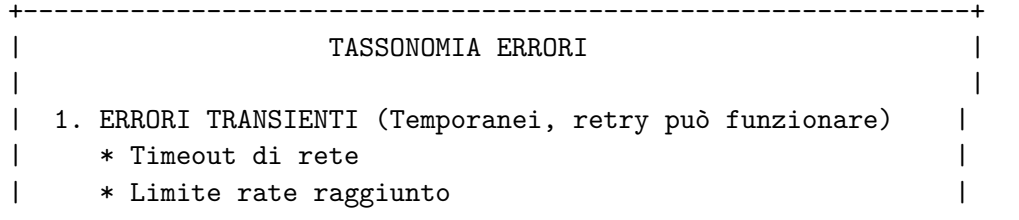
3.2 Architettura Gestore Errori

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



3.3 Tassonomia Errori

Categorie Errori:



```
|     * Servizio temporaneamente non disponibile              |
|     * Timeout connessione database                          |
|     Recupero: Retry con backoff esponenziale                |
|                                                              |
|  2. ERRORI RISORSE (Risorse insufficienti)                  |
|     * Budget esaurito                                       |
|     * Limite memoria superato                               |
|     * Timeout (task troppo lungo)                           |
|     * Context window superata                               |
|     Recupero: Interrompere, notificare utente, suggerire    |
|               alternative                                    |
|                                                              |
|  3. ERRORI LOGICA (Bug interni o assunzioni)                |
|     * Fallimento asserzione                                 |
|     * Puntatore nullo / variabile non definita              |
|     * Indice fuori limiti                                   |
|     * Mismatch tipo                                         |
|     Recupero: Fallback a default sicuro, escalare           |
|                                                              |
|  4. ERRORI INPUT (Input utente errato)                      |
|     * Sintassi invalida                                     |
|     * Fallimento validazione schema                         |
|     * Operazione non supportata                             |
|     * Vincoli in conflitto                                  |
|     Recupero: Chiedere chiarimenti all'utente               |
|                                                              |
|  5. ERRORI ESTERNI (Fallimenti terze parti)                 |
|     * API non disponibile                                   |
|     * Cambio breaking API                                   |
|     * Fallimento esecuzione tool                            |
|     * Modello non disponibile                               |
|     Recupero: Usare fallback, approccio alternativo         |
|                                                              |
|  6. ERRORI SAFETY (Violazioni sicurezza)                    |
|     * Permesso negato                                       |
|     * Violazione limite safety                              |
|     * Tentativo injection rilevato                          |
|     * Azione proibita richiesta                             |
|     Recupero: Rifiutare, loggare, allertare team security   |
+-------------------------------------------------------------+
```

Livelli di Gravità Errore:

```
+-----------------------------------------------------------+
|                  GRAVITÀ ERRORE                           |
|                                                           |
|  BASSA (Degradato ma operativo)                           |
|  * Funzionalità non critica non disponibile              |
|  * Performance degradata ma accettabile                   |
```

```
|  * Esempio: Miss cache pattern (funziona, solo più lento)|
|  Azione: Loggare, continuare operazione                  |
|                                                           |
|  MEDIA (Fallimento parziale)                              |
|  * Subtask fallito ma task può continuare                |
|  * Alternativa non preferita ma accettabile usata        |
|  * Esempio: Tool timeout, usando tool alternativo        |
|  Azione: Loggare, applicare recupero, notificare se      |
|          frequente                                        |
|                                                           |
|  ALTA (Fallimento maggiore)                               |
|  * Task non può completare con successo                  |
|  * Azione utente bloccata                                |
|  * Esempio: Tutte le API modello non disponibili         |
|  Azione: Interrompere task, notificare utente, escalare  |
|                                                           |
|  CRITICA (Fallimento a livello di sistema)                |
|  * Task multipli affetti                                 |
|  * Servizio core down                                    |
|  * Rischio integrità dati                                |
|  * Esempio: Sistema memoria non raggiungibile            |
|  Azione: Escalation emergenza, può mettere in pausa      |
|          nuovi task                                       |
+-----------------------------------------------------------+
```

3.4 Strategie di Recupero

Selezione Strategia Recupero:

```
Funzione GESTISCI_ERRORE(errore, contesto):

  # PASSO 1: Classificare errore
  classificazione = CLASSIFICA_ERRORE(errore)
  # Ritorna: {categoria, gravità, recuperabilità}

  # PASSO 2: Selezionare strategia recupero
  strategia = SELEZIONA_STRATEGIA_RECUPERO(classificazione, contesto)

  # PASSO 3: Eseguire recupero
  PROVA:
    risultato_recupero = ESEGUI_RECUPERO(strategia, errore, contesto)

    SE risultato_recupero.successo:
      LOG_INFO(f"Errore recuperato usando {strategia}")
      RITORNA CONTINUA(risultato_recupero.output)
    ALTRIMENTI:
      # Recupero fallito, provare escalation
      RITORNA ESCALA(errore, strategia, "Recupero fallito")
```

```
  ECCETTO ErroreRecupero come errore_recupero:
    # Recupero stesso fallito
    RITORNA ESCALA(errore, strategia, f"Errore recupero: {errore_recupero}")

Funzione SELEZIONA_STRATEGIA_RECUPERO(classificazione, contesto):

  categoria = classificazione.categoria
  gravità = classificazione.gravità

  # ERRORI TRANSIENTI -> Retry
  SE categoria == "TRANSIENTE":
    SE contesto.contatore_retry < MAX_RETRY:
      RITORNA StrategiaRetry(
        max_tentativi=MAX_RETRY - contesto.contatore_retry,
        backoff=ESPONENZIALE
      )
    ALTRIMENTI:
      RITORNA StrategiaEscalation("Max retry superati")

  # ERRORI RISORSE -> Interrompere o ottimizzare
  ALTRIMENTI SE categoria == "RISORSA":
    SE errore.tipo == "BUDGET_ESAURITO":
      RITORNA StrategiaInterrompi("Budget esaurito, impossibile continuare")
    ALTRIMENTI SE errore.tipo == "CONTESTO_TROPPO_GRANDE":
      RITORNA StrategiaComprimiContesto()
    ALTRIMENTI SE errore.tipo == "TIMEOUT":
      RITORNA StrategiaInterrompi("Task richiede troppo tempo")

  # ERRORI ESTERNI -> Fallback
  ALTRIMENTI SE categoria == "ESTERNO":
    SE FALLBACK_DISPONIBILE(errore.componente):
      RITORNA StrategiaFallback(
        fallback=OTTIENI_FALLBACK(errore.componente)
      )
    ALTRIMENTI:
      RITORNA StrategiaEscalation("Nessun fallback disponibile")

  # ERRORI INPUT -> Chiedere utente
  ALTRIMENTI SE categoria == "INPUT":
    RITORNA StrategiaChiediUtente(
      domanda=GENERA_DOMANDA_CHIARIMENTO(errore)
    )

  # ERRORI LOGICA -> Default sicuro o escalare
  ALTRIMENTI SE categoria == "LOGICA":
    SE HA_DEFAULT_SICURO(errore.operazione):
      RITORNA StrategiaDefaultSicuro()
```

```
    ALTRIMENTI:
      RITORNA StrategiaEscalation("Errore interno, nessun recupero sicuro")

  # ERRORI SAFETY -> Rifiutare ed escalare
  ALTRIMENTI SE categoria == "SAFETY":
    RITORNA StrategiaRifiutaEEscala(
      ragione="Violazione sicurezza",
      allerta_security=Vero
    )

  # Categoria sconosciuta
  ALTRIMENTI:
    RITORNA StrategiaEscalation("Tipo errore sconosciuto")
```

Strategia Retry:

```
StrategiaRetry {
  max_tentativi: int,
  tipo_backoff: "ESPONENZIALE" | "LINEARE" | "COSTANTE",
  ritardo_base: float,  // secondi
  ritardo_max: float,
  jitter: boolean  // Aggiungere casualità per prevenire thundering herd
}

Funzione ESEGUI_RETRY(strategia, operazione, contesto):

  PER tentativo IN INTERVALLO(1, strategia.max_tentativi + 1):

    PROVA:
      risultato = ESEGUI(operazione, contesto)
      RITORNA SUCCESSO(risultato)

    ECCETTO Errore come errore:
      SE tentativo == strategia.max_tentativi:
        # Ultimo tentativo fallito
        RITORNA FALLIMENTO("Tutti i tentativi retry esauriti")

      # Calcolare tempo di attesa
      SE strategia.tipo_backoff == "ESPONENZIALE":
        attesa = MIN(strategia.ritardo_base * (2 ** tentativo), strategia.ritardo_max)
      ALTRIMENTI SE strategia.tipo_backoff == "LINEARE":
        attesa = MIN(strategia.ritardo_base * tentativo, strategia.ritardo_max)
      ALTRIMENTI:
        attesa = strategia.ritardo_base

      # Aggiungere jitter
      SE strategia.jitter:
        attesa = attesa * (0.5 + CASUALE() * 0.5)
```

```
      LOG_INFO(f"Tentativo retry {tentativo}, attesa {attesa}s")
      DORMI(attesa)
      # Il ciclo continua al tentativo successivo
```

3.5 Gestione Escalation

Albero Decisionale Escalation:

```
Errore Verificato
    |
Può recuperare automaticamente?
+- SÌ -> Applicare Recupero
|        |
|        Successo?
|        +- SÌ -> Continuare (Loggare per apprendimento)
|        +- NO -> Escalare
|
+- NO -> Valutare Gravità
          |
          Gravità?
          +- BASSA -> Loggare, Continuare con funzionalità degradata
          +- MEDIA -> Loggare, Notificare utente del problema
          +- ALTA -> Interrompere task, Notificare utente, Loggare
          |         incidente
          +- CRITICA -> Escalation emergenza, Allertare team

Percorsi Escalation:
* Notifica utente (per gravità ALTA che affetta il loro task)
* Alert team engineering (per problemi di sistema CRITICI)
* Alert team security (per violazioni safety)
* Creazione incidente (per fallimenti ripetuti)
```

Azioni Escalation:

```
Funzione ESCALA(errore, contesto, ragione):

  gravità = errore.classificazione.gravità

  # Creare record incidente
  incidente = Incidente {
    id: GENERA_ID(),
    timestamp: ORA(),
    errore: errore,
    contesto: contesto,
    ragione: ragione,
    stato: "APERTO"
  }

  MEMORIZZA_INCIDENTE(incidente)
```

```
  # Escalation basata su gravità
  SE gravità == "CRITICA":
    # Azione immediata richiesta
    INVIA_ALERT(
      canale="pager",
      destinatari=INGEGNERI_REPERIBILITA,
      messaggio=f"CRITICO: {errore.sommario}",
      id_incidente=incidente.id
    )

    # Potrebbe essere necessario fermare accettazione nuovi task
    SE errore.affetta_sistema_core:
      IMPOSTA_STATO_SISTEMA("DEGRADATO")

  ALTRIMENTI SE gravità == "ALTA":
    # Task fallito, notificare utente
    NOTIFICA_UTENTE(
      user_id=contesto.user_id,
      messaggio=f"Task fallito: {errore.messaggio_user_friendly}",
      id_incidente=incidente.id,
      retry_possibile=errore.retry_possibile
    )

    # Allertare team engineering (non urgente)
    INVIA_ALERT(
      canale="slack",
      destinatari=TEAM_ENGINEERING,
      messaggio=f"Errore gravità ALTA: {errore.sommario}",
      id_incidente=incidente.id
    )

  ALTRIMENTI SE gravità == "MEDIA":
    # Loggare e notificare utente se appropriato
    NOTIFICA_UTENTE(
      user_id=contesto.user_id,
      messaggio=f"Task completato con problemi: {errore.messaggio_user_friendly}",
      livello="WARNING"
    )

  RITORNA incidente.id
```

3.6 Analisi Errori

Rilevamento Pattern Errori:

```
Funzione ANALIZZA_PATTERN_ERRORI(finestra_tempo):
```

```
errori = OTTIENI_ERRORI_IN_FINESTRA(finestra_tempo)

# PATTERN 1: Picchi di frequenza
tasso_errori = len(errori) / finestra_tempo.durata
tasso_baseline = OTTIENI_TASSO_ERRORI_BASELINE()

SE tasso_errori > tasso_baseline * 2:
  ALERT("Picco tasso errori rilevato", {
    corrente: tasso_errori,
    baseline: tasso_baseline
  })

# PATTERN 2: Nuovi tipi di errore
tipi_errore = INSIEME(errore.tipo per errore in errori)
tipi_conosciuti = OTTIENI_TIPI_ERRORE_CONOSCIUTI()
nuovi_tipi = tipi_errore - tipi_conosciuti

SE nuovi_tipi:
  ALERT("Nuovi tipi errore rilevati", {
    nuovi_tipi: lista(nuovi_tipi),
    frequenza: {tipo: CONTA(errori dove errore.tipo == tipo) per tipo in nuovi_tipi}
  })

# PATTERN 3: Errori correlati
# Trovare errori che tendono a verificarsi insieme
correlazioni = TROVA_CORRELAZIONI_ERRORI(errori)

PER correlazione IN correlazioni DOVE correlazione.significatività > SOGLIA:
  ALERT("Correlazione errori rilevata", {
    errore_A: correlazione.tipo_A,
    errore_B: correlazione.tipo_B,
    correlazione: correlazione.coefficiente,
    # Può indicare causa radice comune
  })

# PATTERN 4: Fallimenti a cascata
# Errori che innescano altri errori
cascate = TROVA_CASCATE_ERRORI(errori)

PER cascata IN cascate:
  ALERT("Fallimento a cascata rilevato", {
    trigger: cascata.errore_iniziale,
    causati: cascata.errori_successivi,
    # Necessario risolvere causa radice
  })

# PATTERN 5: Pattern specifici utente
distribuzione_errori_utente = RAGGRUPPA_PER(errori, 'user_id')
```

```
PER user_id, errori_utente IN distribuzione_errori_utente:
  SE len(errori_utente) > PERCENTILE(conteggi_errori, 95):
    INVESTIGA("Utente sta sperimentando alto tasso errori", {
      user_id: user_id,
      conteggio_errori: len(errori_utente),
      tipi_errore_comuni: PIÙ_COMUNI(errori_utente, chiave='tipo')
      # Può essere problema specifico utente o abuso
    })
```

Prossimo: 06-data-flows.md -> Pattern di interazione dettagliati e trasformazioni dati