

Contents

Architecture Patterns Catalog	2
Overview	2
Pattern Selection Guide	2
Pattern 1: Minimal Loop Agent	2
Applicabilità	2
Architecture	2
Components	3
Design Principles	3
Bounded Emergence	3
When to Use	3
Expected Performance	3
Pattern 2: Reflective Agent	4
Applicabilità	4
Architecture	4
Components	4
Design Principles	4
Bounded Emergence	5
When to Use	5
Expected Performance	5
Pattern 3: Verified Agent	5
Applicabilità	5
Architecture	5
Components	6
Design Principles	6
Bounded Emergence	6
When to Use	6
Expected Performance	6
Pattern 4: Reactive Agent	7
Applicabilità	7
Architecture	7
Components	7
Design Principles	8
Bounded Emergence	8
When to Use	8
Expected Performance	8
Pattern 5: Collaborative Multi-Agent	8
Applicabilità	8
Architecture	8
Components	9
Design Principles	9
Bounded Emergence	9
When to Use	9
Expected Performance	10
Comparative Analysis	10
Pattern Evolution	10
Hybrid Patterns	10

Architecture Patterns Catalog

Overview

Questo documento presenta **5 pattern architetturali** derivati dall'analisi di problem taxonomy e design dimensions.

Ogni pattern è ottimizzato per specifiche classi di problemi. **Non esiste pattern universale** - la scelta dipende da contesto.

Pattern Selection Guide

Problem Class → Recommended Pattern

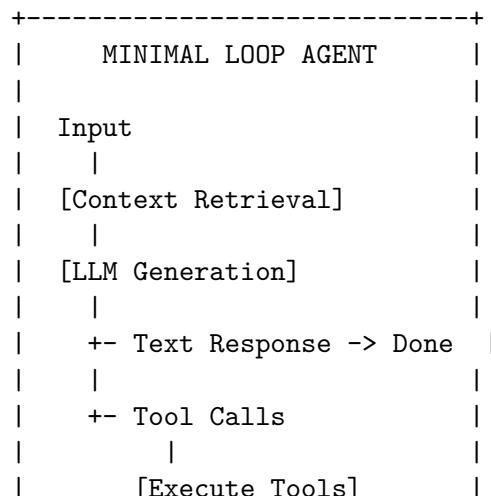
Classe A (Simple & Variable)	-> Pattern 1: Minimal Loop
Classe B (Complex Planning)	-> Pattern 2: Reflective Agent
Classe C (Safety-Critical)	-> Pattern 3: Verified Agent
Classe D (Real-Time)	-> Pattern 4: Reactive Agent
Classe E (Multi-Agent)	-> Pattern 5: Collaborative Multi-Agent

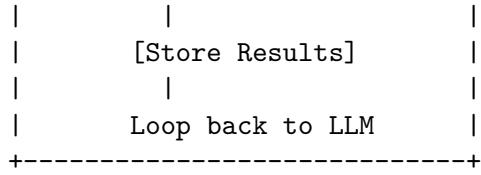
Pattern 1: Minimal Loop Agent

Applicabilità

- **Problem Class:** A (Simple & Variable)
 - **Complessità:** Level 0-1
 - **Latenza:** Interactive (<2s)
 - **Affidabilità:** Best-effort
 - **Safety:** Unconstrained

Architecture





Components

Core (3): 1. **Execution Loop:** Simple perceive-reason-act cycle 2. **Memory:** Context window only (no persistent storage initially) 3. **Tool Interface:** Basic registry + execution

Optional Extensions: - Long-term memory (vector DB) quando history matters - Structured logging quando debugging needed - Input validation quando security matters

Design Principles

- [OK] Minimalism: Fewest components possible
- [OK] Emergence: LLM handles complexity
- [OK] Rapid iteration: Fast to build and modify
- [NO] No explicit planning module
- [NO] No extensive error handling
- [NO] No formal verification

Bounded Emergence

Emergent: Planning, error recovery, adaptation **Bounded:** Tool whitelist, resource limits, output validation (optional)

When to Use

[OK] **Use when:** - Rapid prototyping/MVP - General-purpose assistant - High variability input - Failure tolerable - Budget constrained

[NO] **Don't use when:** - Safety-critical - Compliance required - Deterministic behavior needed - Complex multi-step critical

Expected Performance

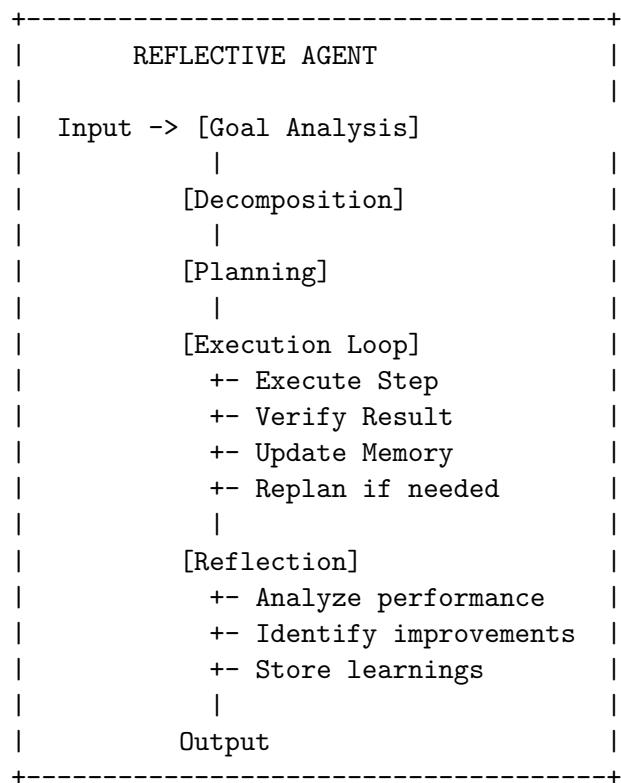
- Latency: <2s per simple query
- Success Rate: 70-85%
- Cost: Low (\$0.01-0.05 per task)
- Coverage: 80% of simple tasks

Pattern 2: Reflective Agent

Applicabilità

- **Problem Class:** B (Complex Planning)
- **Complessità:** Level 2-3
- **Latenza:** Responsive-Batch (2s-5min)
- **Affidabilità:** High
- **Safety:** Hard constraints

Architecture



Components

Core (6): 1. **Goal Decomposition:** Break complex task into subtasks 2. **Planning Module:** Generate execution plan 3. **Execution Loop:** Execute with monitoring 4. **Episodic Memory:** Store task executions 5. **Reflection Module:** Analyze and learn 6. **Tool Interface:** Extended with composition

Design Principles

- [OK] Explicit planning before execution
- [OK] Verification after each step
- [OK] Learning from episodes
- [OK] Decision logging

- [NO] Not real-time (planning overhead)

Bounded Emergence

Emergent: Plan generation, step adaptation, pattern learning **Bounded:** Plan verification, step validation, safety checks, resource budgets

When to Use

[OK] **Use when:** - Complex multi-step tasks - High success rate required - Learning from experience beneficial - Moderate latency acceptable

[NO] **Don't use when:** - Simple tasks (overkill) - Real-time required - Determinism absolute

Expected Performance

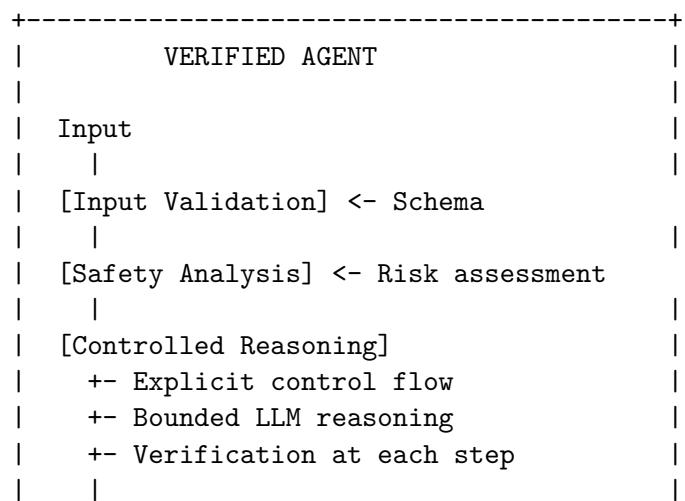
- Latency: 5s-2min per complex task
- Success Rate: 85-95%
- Cost: Moderate (\$0.10-0.50 per task)
- Coverage: 90% of complex tasks

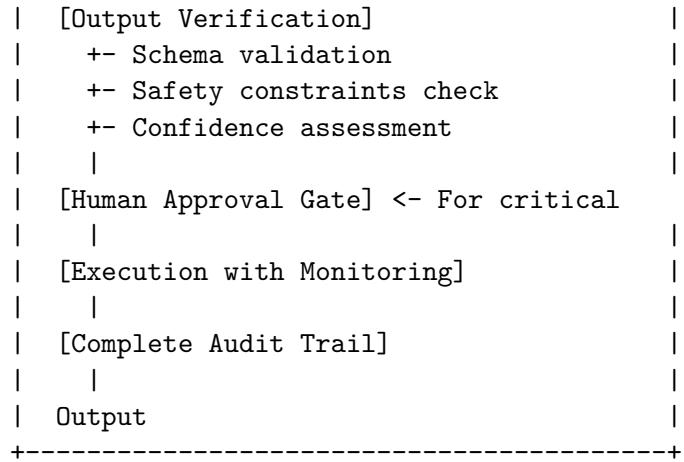
Pattern 3: Verified Agent

Applicabilità

- **Problem Class:** C (Safety-Critical)
- **Complessità:** Variable
- **Latency:** Variable
- **Affidabilità:** Mission/Safety-critical
- **Safety:** Rigid bounds

Architecture





Components

Core (8): 1. **Multi-Layer Validation:** Input, output, action 2. **Safety Verifier:** Check all constraints 3. **Controlled Execution:** Explicit flow + bounded emergence 4. **Audit Logger:** Complete traceability 5. **Human-in-Loop:** Approval gates 6. **Rollback System:** Undo capability 7. **Confidence Scorer:** Uncertainty quantification 8. **Fallback Mechanisms:** Safe defaults

Design Principles

- [OK] Safety first: Multiple verification layers
- [OK] Traceable: Every decision logged
- [OK] Human oversight: Approval for critical
- [OK] Fail-safe: Rollback + fallback
- [NO] No unconstrained emergence
- [NO] No black-box decisions

Bounded Emergence

Emergent: Analysis, suggestion generation, pattern recognition **Bounded: Rigid -** all bounds strictly enforced, violations -> human escalation

When to Use

[OK] **Use when:** - Failure catastrophic - Regulatory compliance - Medical/legal/financial critical - Audit trail mandatory

[NO] **Don't use when:** - Best-effort acceptable (overkill) - Rapid iteration needed (too rigid) - General-purpose tasks

Expected Performance

- Latency: Variable (human approval can add minutes)
- Success Rate: 95-99%+ (with conservative rejection)

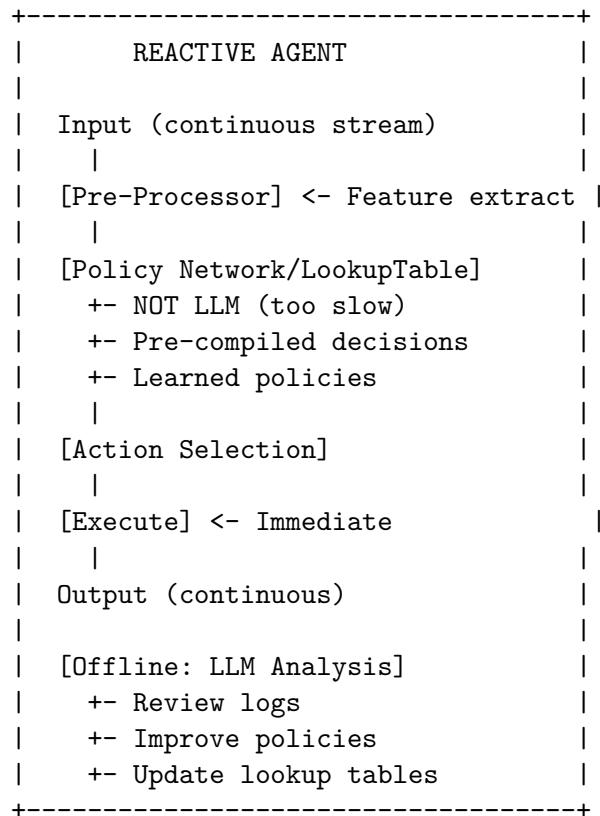
- Cost: High (\$0.50-5.00 per task, including human time)
- Coverage: 60-80% (conservative bounds reject edge cases)

Pattern 4: Reactive Agent

Applicabilità

- **Problem Class:** D (Real-Time)
- **Complessità:** Level 0-1 (forced by latency)
- **Latenza:** Real-time (<100ms)
- **Affidabilità:** Variable
- **Safety:** Domain-specific

Architecture



Components

Online (Real-Time): 1. **Fast Pre-Processor:** Feature extraction 2. **Policy Store:** Pre-compiled decisions 3. **Action Executor:** Immediate execution

Offline (Batch): 4. **LLM Analyzer:** Review and improve policies 5. **Policy Generator:** Create new rules 6. **Simulator:** Test before deployment

Design Principles

- [OK] Pre-computation: All heavy lifting offline
- [OK] Lookup/small models: Fast online decisions
- [OK] LLM in the loop: But offline only
- [NO] No online LLM calls (too slow)
- [NO] No complex reasoning (no time)

Bounded Emergence

Emergent (Offline): Policy generation, improvement strategies **Bounded**: Policies validated offline, deployed as deterministic lookup

When to Use

[OK] **Use when**: - <100ms latency mandatory - Robotics, real-time control - Gaming AI - High-frequency decisions

[NO] **Don't use when**: - Latency tolerance >1s (use richer architecture) - High variability input (pre-compilation insufficient)

Expected Performance

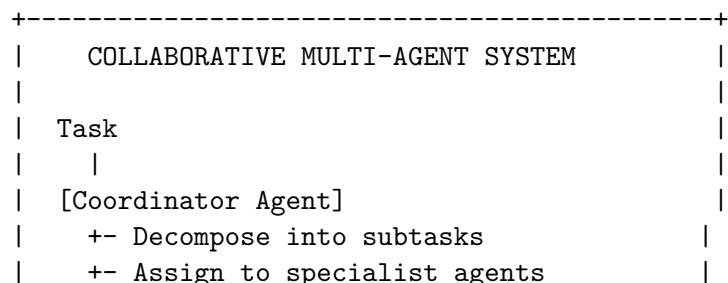
- Latency: <100ms
- Success Rate: Domain-dependent
- Cost: Low online (pre-compiled), moderate offline (LLM policy generation)
- Coverage: Limited to pre-compiled scenarios

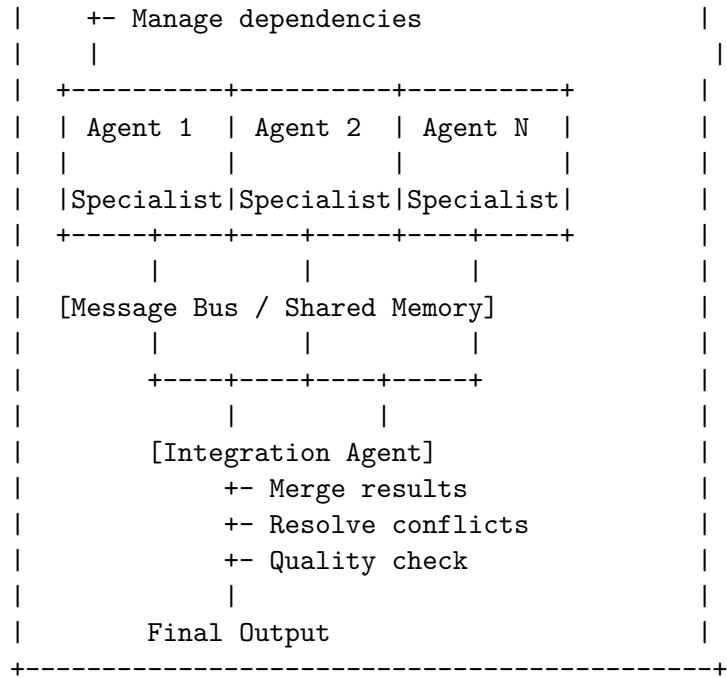
Pattern 5: Collaborative Multi-Agent

Applicabilità

- **Problem Class**: E (Multi-Agent)
- **Complessità**: Level 3-4
- **Latency**: Batch-Offline
- **Affidabilità**: High
- **Safety**: Coordinated constraints

Architecture





Components

Core: 1. **Coordinator:** Task decomposition & assignment 2. **Specialist Agents:** Domain-specific expertise 3. **Message Bus:** Communication infrastructure 4. **Shared Memory:** Common knowledge base 5. **Integration Agent:** Results merging 6. **Conflict Resolver:** Handle disagreements

Design Principles

- [OK] Specialization: Each agent expert in subdomain
- [OK] Parallelization: Agents work concurrently
- [OK] Coordination: Explicit communication protocol
- [OK] Integration: Results merged coherently
- [NO] Not for simple tasks (overhead)

Bounded Emergence

Emergent: Agent collaboration strategies, dynamic task distribution **Bounded:** Communication protocols, coordinator oversight, integration validation

When to Use

[OK] **Use when:** - Task naturally decomposes into parallel subtasks - Specialization beneficial - Scale requires parallelization - Latency tolerance high

[NO] **Don't use when:** - Task sequential/non-decomposable - Single agent sufficient - Coordination overhead > benefits

Expected Performance

- Latency: Minutes-hours (but parallel speedup)
- Success Rate: 80-95% (depends on integration)
- Cost: High (multiple agents) but parallelized
- Coverage: 85%+ for decomposable complex tasks

Comparative Analysis

Dimension	Minimal	Reflective	Verified	Reactive	Multi-Agent
Complessità Impl.	Low	Medium	High	Medium	Very High
Latency	Low	Medium	Variable	Very Low	High
Success Rate	70-85%	85-95%	95-99%	Variable	80-95%
Cost	Low	Medium	High	Low (online)	High
Flexibility	Very High	High	Low	Very Low	High
Safety	Low	Medium	Very High	Medium	Medium
Learning	Limited	Yes	Limited	Offline	Per-agent
Traceability	Low	High	Complete	Limited	Medium

Pattern Evolution

Architetture tipicamente evolvono:

MVP → Production → Mature

Minimal → Reflective → Verified
(add capabilities as requirements grow)

Reflective → Multi-Agent
(scale through parallelization)

Any → + Reactive Components
(add fast paths for critical loops)

Hybrid Patterns

In practice, mix patterns:

Example: E-Commerce Assistant - Minimal Loop: General customer queries (80% traffic) - Reactive: Product recommendations (<100ms) - Verified: Payment processing (safety-critical) - Reflective: Complex troubleshooting - Multi-Agent: Inventory management + logistics coordination

Next Steps

1. Classify problem using 01-problem-taxonomy.md
2. Select base pattern from this catalog

3. Customize using 02-design-dimensions.md
 4. Apply 05-bounded-emergence.md principles
 5. Validate with 04-evaluation-framework.md
-

Contribution: First systematic catalog of architecture patterns for LLM-based agents, mapped to problem classes.