TUTORIAL #1

BARCELONA
2025

The Anatomy of Agentic Frameworks and Practical Use Cases

Speaker

Prof. Dr. Petre Dini, IARIA, USA/EU

Statement: We were all using an Agentic framework, with one small caveat: not based on a natural language narrative for agent communications, but based on formal rules and well-defined protocols, instead.

Stories: I remember when I was presenting the project requirements and goals, while some people were already starting to write the code. I recall that with partial accuracy only, we achieved 99.999 service availability across the USA.

Déjà vu

BARCELONA
2025

Basis: requirements, players & duties (hardware and software), message flowcharts, hardware selection, software framework, APIs, Interfaces, user interfaces, etc. [then: verification, validation, assurance)

MODEL
Abstract classes, objects, aggregation, inheritance, object contracts, agents (smart objects), MAS, agentic framework

Keywords of change: automation, self-reasoning, and self-healing

'Old' OO (or so) models

> English narrative
> Structured requirements
> Nouns, verbs -> objects, actions, goals
> OO-framework (IBM: Java, Eclipse}
Framework objects (cca 10%)
Specific objects, operational request
Middleware for object communication (traders, brokers, hierarchy, etc.)
BUS architecture (event subscription)
Object storage (specific databases: ObjectStore, etc.)

Formalisms
Verification
Validation
Maintenance
Modeling
Simulation
Monitoring
Management
Reflective architectures
vs
Digital Twins

'New' Agentic frameworks

> English narrative
> Structured requirements
> LLCs identify main requirements
> LLMs identify specific constraints and goals
> Agentic-framework
Framework agents (cca 90%)
Specific agents
Middleware as an agent [Orchestrator]
Communicating Agents (hallucinations, bias)
Agents library

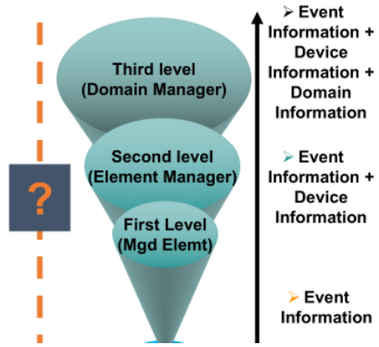
Industrial Challenges in Working with Events

Prof. Dr. Petre DINI
Senior Technical Leader, NMTG Office of the CTO
Cisco Systems, Inc.
pdini@cisco.com

DEBS 2004
Edinburg

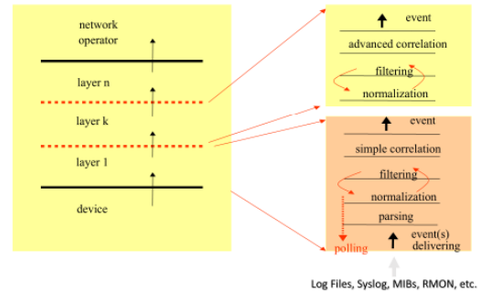
Bottom-up vs. Top-down

- Domain Manager enriches with domain information
- EMS enriches with multi-device information
- Notification Engine collects OS notifications

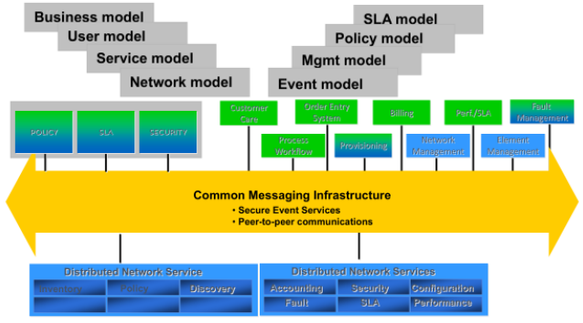


- Event Information + Device Information + Domain Information
- Event Information + Device Information
- Event Information

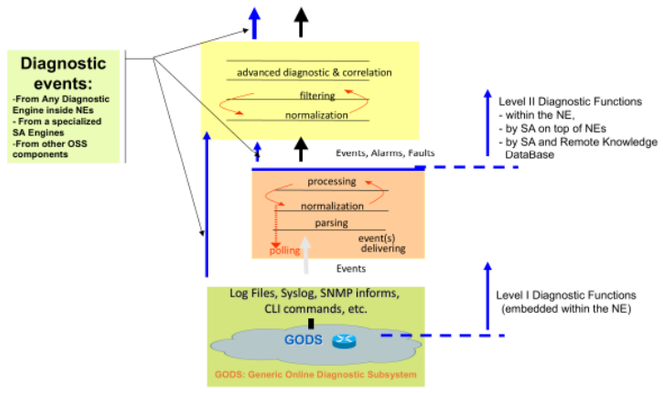
A Layered Processing View



Communication Bus

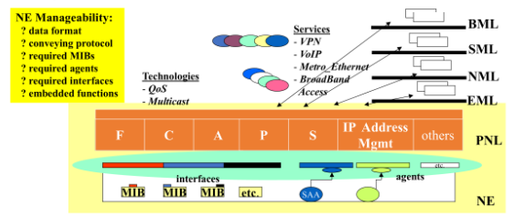


Multi-level diagnostic

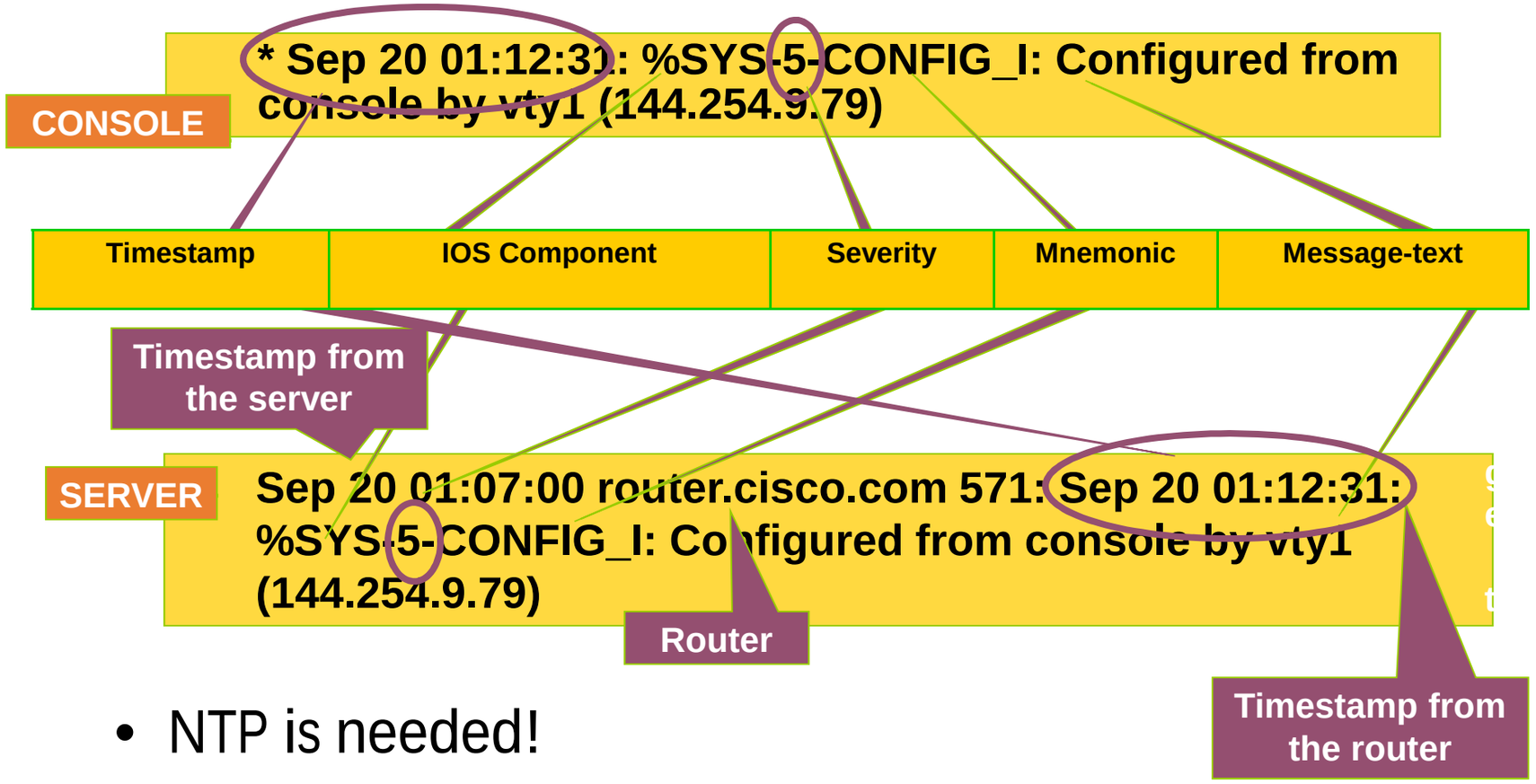


Syntax Issues

- Various formats
- Myriad of conversions needed
- Lack of syntax control



Syslog Message “Body” Format in the IOS



- NTP is needed!
- **Header:level can be different than Body:severity**

Use Case #1: Sport Tickets

BARCELONA 2025

(Example) Assume

a. We have a paragraphs of ten sentences describing a potential system (irrelevant, but as an example: selling football tickets, coming with requirements for distribution, security, authenticity of the tickets, etc.)
b. We opt for using an agentic framework
c. Q1: What are the sequence of steps?
d. Q2: Are there dedicated agents we should choose from (having definite roles, limitations, etc.)?
e. Q3: Can we build personalized agents and insert them into the whole framework?
f. Q4: How are the constraints of the original system (requirements) translated into the goals of the agentic framework and how are the goals assigned to the agents, namely, are they split (par., seq.) in sub-goals, or joint, or mediated if they are conflicting?

Note: goals conflicts can be static (easy verifiable) or dynamic (changing, status, volatile, ...)

Sport Tickets - Pipeline

BARCELONA 2025

Q1 — Sequence of steps (agentic design pipeline)

- **Requirements → Constraints**
  Normalize the paragraph into atomic constraints (e.g., “only 6 tickets/user,” “cryptographic authenticity,” “latency < 200 ms,” “fair queueing,” “GDPR compliance”).
- **Constraints → Goals** (WHAT)
  Translate each constraint into one or more goals with acceptance criteria (e.g., VerifyTicketAuth with “<5 ms verify; FIPS-approved crypto”).
- **Goals → Capabilities** (HOW)
  For each goal, list capabilities needed: verify signature, allocate inventory, anti-bot scoring, payment, ID verification, audit logging, anomaly detection.
- Organization model
  Choose agent types and interaction styles: hierarchical (coordinator), market/contract-net, or peer mesh. Define authority, priorities, SLAs.
- Allocation
  Map goals ↔ agents via: required capability match, trust level, performance budget, and data locality. (Greedy first fit → refine with constraints solver).
- Coordination protocols
  Pick protocols per interaction: request–response, publish/subscribe, contract-net (bids), two-phase commit, saga (compensations).
- Conflict handling
  Predefine policies for scarce inventory, double-spend, identity disputes, fairness vs. revenue, security vs. latency. Attach tie-breakers.
- Assurance hooks
  Add runtime monitors (temporal rules), guard rails (Simplex/shields), provenance logs, and canary scenarios.
- Simulation & dry-runs
  Load/chaos tests with adversaries (scalpers/bots), failure injection, latency budgets.
- Deployment with continuous governance
  SLAs, rate limits, ABAC/RBAC, rotation of keys/models, drift control, post-mortems.

Sport Tickets - Agents

Q2 — Dedicated agents (typical roles & limits)

- Orchestrator/Goal Manager: decomposes goals, assigns tasks; limits: no direct data custody.
- Inventory Agent: seat allocation, holds, releases; limits: cannot bypass fairness policy.
- AuthN/Z Agent: KYC/ID checks, RBAC/ABAC decisions; limits: no pricing authority.
- Crypto/Attestation Agent: signing, verification, key rotation, HSM access; limits: read-only to PII.
- Payment & Risk Agent: PSP integration, fraud scoring, SCA, chargeback handling; limits: cannot allocate seats.
- Anti-Bot/Trust Agent: device fingerprinting, rate-limit advice, CAPTCHA orchestration; limits: advisory → Orchestrator enforces.
- Queueing/Fairness Agent: virtual lobby, lottery/queue discipline, per-user caps; limits: cannot edit ticket metadata.
- Compliance & Privacy Agent: data minimization, consent, retention, audit trails; veto power on unlawful flows.
- Observability Agent: SLO monitors, tracing, anomaly alerts; limits: no business decisions.
- Settlement & Ledger Agent: immutable log (append-only), refunds, compensations; limits: no user policy changes.

Q3 — Personalized agents

Yes. Define a capability contract (inputs/outputs, pre/post-conditions, latency & trust class), implement your agent, and register it with the Orchestrator. It can then be selected during allocation if it satisfies:

```
capabilities ⊇ goal.reqs && SLA_met && policy_compliant && trust_level_ok
```

Sport Tickets - Goals

BARCELONA 2025

Translating Requirements → Goals → Agents (mini example)

1. Requirements (excerpt)

R1 Authentic tickets only; cryptographic validation.
R2 Max 6 tickets/user; prevent bots.
R3 Fair access at drop time; no cart hoarding.
R4 End-to-end latency < 200 ms.
R5 GDPR compliance; immutable audit.

2. Goals

G1 VerifyTicketAuth (verify ≤5 ms, FIPS algos).
G2 EnforceUserCaps (≤6/user, per-event).
G3 EnsureFairAccess (virtual lobby + lottery/queue).
G4 MeetLatencyBudget (<200 ms, back-pressure).
G5 Provenance&Audit (append-only, replayable).
G6 AntiBotMitigation (risk score; action ladder).
G7 PrivacyCompliance (min data, DSR support).

3. Allocation (sample):

G1 → Crypto/Attestation Agent
G2 → Queueing/Fairness Agent + AuthZ Agent
G3 → Queueing/Fairness Agent (lottery/queue policy)
G4 → Orchestrator + Observability (shed/back-pressure)
G5 → Settlement & Ledger Agent
G6 → Anti-Bot/Trust Agent (+ Orchestrator enforcer)
G7 → Compliance & Privacy Agent

4. Conflict patterns & policies

Fairness vs. Revenue (R2 vs dynamic pricing): declare lexicographic priority: safety/security → compliance → fairness → revenue.
Latency vs. Security (R4 vs strong checks): apply progressive trust: light check on hot path; deep check async or on anomalies.
User Cap vs. Group Orders: introduce goal refinement: EnforceUserCaps → per-identity + per-payment-instrument + per-device.
Anti-Bot false positives vs. Fairness: dual-channel appeal (human-in-the-loop) with bounded SLA.

Use Case #2: Healthcare- Agents

Telehealth triage & e-prescription scenario; agentic coordination under strict safety, privacy, and auditability constraints

The increasing pressure on healthcare systems to deliver rapid, accurate, and equitable medical responses has intensified interest in telehealth triage augmented by intelligent agents. In such settings, patients interact remotely with AI-assisted services capable of preliminary symptom evaluation, prioritization, and e-prescription. The incentive lies in creating an agentic ecosystem where distributed digital actors — diagnostic agents, privacy guardians, audit agents, and prescribing modules — collaborate autonomously yet transparently to ensure that each clinical and administrative action respects medical ethics, data protection laws, and accountability standards.

Building this environment demands traceable goal alignment among agents: safety goals constrain diagnostic decisions, privacy goals regulate data access and transfer, and audit goals enforce non-repudiation and explainability. The overarching motivation is to design an intelligent, self-monitoring telehealth process that maintains human oversight while improving timeliness and consistency. From these drivers emerge the system requirements — for secure reasoning pipelines, consent-aware data flow, verifiable e-prescription issuance, and adaptive escalation to human clinicians — forming the structural and behavioral blueprint of an agentic telehealth framework.

Healthcare - Goals

Translating Requirements → Goals → Agents (mini example)

B1. Requirements (excerpt)

R1 Patient safety first; critical symptoms must escalate to human within 2 min.
R2 HIPAA privacy; minimum necessary data, consent tracking, audit logs.
R3 e-Rx authenticity; cryptographic signing + pharmacy verification.
R4 Latency: triage advice ≤ 5 s p95; video intake stable at 30 fps.
R5 Bias control in triage recommendations; explanations required.
R6 Data provenance for training/QA; no raw PHI leaves region.
R7 Fallback: degraded service during outages (phone/SMS, cached rules).

B2. Goals (what?)

G1 SafeTriage (critical→human ≤2 min; advice ≤5 s; explainable).
G2 PrivacyCompliance (HIPAA, consent, minimization, DSR).
G3 AuthEPrescription (sign e-Rx; verify at pharmacy).
G4 MeetLatencyBudget (p95 ≤5 s; video QoS maintained).
G5 Provenance&Audit (append-only, region-bound).
G6 BiasMonitoring (drift/inequity alarms; periodic fairness reports).
G7 Resilience (graceful degradation path active on SLO breach).

B3. Agents (roles)

Orchestrator/Goal Manager — decomposes & routes requests, enforces priorities.
Triage Agent — symptom NLP + rules/ML; advisory on severity.
Clinical Safety Agent (Runtime Assurance) — safety authority; escalates to human; Privacy & Compliance Agent — consent checks, field minimization, policy gate.
e-Rx Crypto Agent — HSM-backed signing/verification, key rotation.
QoS Agent — monitors latency/video metrics; triggers adaptive bitrate/throttling.
Bias & Drift Agent — monitors cohort outcomes; gates model updates.
Provenance/Ledger Agent — immutable logs, regional storage policy.
Fallback/Resilience Agent — activates phone/SMS workflows; caches top rules.

B4. Allocation (sample):

G1 → Triage Agent (advice) + Clinical Safety Agent (escalation decision).
G2 → Privacy & Compliance Agent (gate on every data flow).
G3 → e-Rx Crypto Agent (sign/verify).
G4 → Orchestrator + QoS Agent (shed load, degrade bitrate).
G5 → Provenance/Ledger Agent.
G6 → Bias & Drift Agent (observe-only → periodic reports; veto
G7 → Fallback/Resilience Agent (policy-driven activation).

B5. Conflict patterns & policies

• Latency vs. Safety: If p95 > 5 s, short-path rules activate; Safety Agent keeps escalation budget ≤2 min.
• Privacy vs. Explainability: Explanations must not reveal PHI beyond consent; Compliance Agent redacts.
• Throughput vs. Video QoS: QoS Agent reduces bitrate/resolution before dropping sessions.
• Model vs. Policy: If Triage suggests “home care” but red flags present, Safety Agent overrides → human.

B6. Example temporal guards

IF red_flag_detected THEN escalate_to_human WITHIN 120s
IF queue_length > Qmax THEN enable_short_path_triage AND alert Ops
IF consent.revoked(user) THEN purge non-essential caches WITHIN 24h

B7. Assurance snapshots

Pre-deployment: spec-based tests (STL/MTL), failure injection (network, HSM), PHI leak scanner.
Runtime: monitors + signed decisions to Ledger; fairness drift alarms weekly; Simplex-style safety override always on.
Continuous: canary updates; post-incident reviews tied to provenance.

A1) Example personalized agent [Health]

Agent name: ProgressiveTrustScorer
Purpose: Score each purchase/session for bot/abuse risk without hurting latency; supports progressive checks (cheap→costly).
Capability contract (WHAT + SLA):
Inputs: { user_id, device_fingerprint, payment_hash, ip, behavior_signals[], cart_value }
Outputs: { risk_score∈[0,1], risk_band∈{low,med,high}, actions: {throttle?, challenge?, block?}, rationale[] }
Non-functionals: p95 ≤ 5ms, p99 ≤ 12ms, availability ≥ 99.95%, determinism on same inputs (seeded), no PII persistence.
Trust class: Advisory (cannot directly block; Orchestrator enforces).
Policies: No use of prohibited features (e.g., demographics), explainability required (rationale with top 3 features), signed responses.

```
{
  "name": "ProgressiveTrustScorer",
  "version": "1.2.0",
  "inputs": {
    "user_id": "uuid",
    "device_fingerprint": "string",
    "payment_hash": "sha256",
    "ip": "ipv4|ipv6",
    "behavior_signals": "array<float>",
    "cart_value": "float"
  },
  "outputs": {
    "risk_score": "float[0,1]",
    "risk_band": "enum(low,med,high)",
    "actions": {"throttle": "bool", "challenge": "bool", "block": "bool"},
    "rationale": "array<string>"
  },
  "slas": {"p95_ms": 5, "p99_ms": 12, "availability": "99.95%"},
  "trust_class": "Advisory",
  "prohibited_features": ["age","race","religion","gender"]
}
```

A2) Avoiding conflicts with the “agent library”

Design-time controls

1. Capability ontology & registry
   1. Each agent declares capabilities, pre/post-conditions, trust class, resource budget.
   2. Namespacing: `risk.trust.progressive.v1`. Prevents accidental role overlap.
2. Conflict declarations
   1. Agents state mutual-exclusion or precedence:
      `1.conflicts_with: ["risk.trust.static"], precedence_under: ["queue.fairness"].`
3. Policy contracts (ABAC/RBAC)
   1. Who can decide vs. advise. Your new agent stays “advisory” unless governance promotes it.
4. Static conformance suite
   1. Schema validation, SLA proof (bench harness), negative tests (reject PII, fail closed).
5. Formal invariants (lightweight)
   1. Temporal specs like “never both `block` and `low` band”; Orchestrator checks at runtime.

Run-time controls

Arbitration layer in Orchestrator

Priority graph: Safety > Compliance > Fairness > Revenue.
Rule combiner: (deny-overrides, permit-overrides, first-applicable).

Resource quotas & circuit breakers

Cap CPU/latency; isolate misbehaving agents (bulkhead pattern).

Shadow/canary & auto-revert

New agent runs shadow; if SLOs or policy drift, auto-disable and alert.

Provenance & explainability bus

All decisions with signatures → Ledger; conflicting outputs are flagged for adjudication.

Treat your custom agent as a plugin with a contract.
Declare where it sits in the hierarchy, what it may never do, and how conflicts are arbitrated.

Healthcare- Synchronization

Agents' behavior is driven by feedback from the real system (disregard simulation). The events from the system reach the agents after going up from the physical elements (e.g, CPU, routers) through many levels, and each attaches to the event its own timestamp (assume they all have a clock); Now, the events reach different agents. What is the mechanism for agent synchronization?

1. Time Sources and Synchronization Layers

Clock sync protocols:

- NTP (Network Time Protocol) — ms-level sync, good for distributed IT.
- PTP (Precision Time Protocol / IEEE 1588) — µs or ns-level sync, used in telecom, finance, avionics.
- GPS-disciplined clocks — absolute reference, often for high-assurance systems.

Local monotonic clocks: Each node uses a monotonic counter (not wall clock) to avoid regressions when NTP shifts.

2. Event Timestamp Strategies

Source timestamping: Events stamped at origin (e.g., NIC, router ASIC, kernel driver). Highest fidelity, but requires hardware support.

Ingress timestamping: Middleware adds a timestamp on receipt. Easier, but mixes propagation delays.

Hybrid: Keep both (source + ingress) so later agents can reason about uncertainty.

3. Event Correlation Mechanisms

Lamport clocks: Logical clocks — give a partial order (“happened before”) without relying on wall time.

Vector clocks: Capture causality in multi-agent settings; more overhead but resolve concurrent vs. causal.

Hybrid logical clocks (HLC): Combine physical time with logical counters — practical for distributed systems like CockroachDB.

Causal message tagging: Propagate context (e.g., trace IDs in OpenTelemetry) so events can be reassembled in order.

4. Dealing with Clock Drift and Skew

Bounded uncertainty windows: Every timestamp is expressed as `[t ± δ]` where δ is drift + jitter. Agents reason over intervals, not points.

Temporal logic with slack: Instead of “event A before event B,” you specify `A → B within [0, 50 ms]` with tolerance.

Resequencers / buffers: Small delay queues reorder events by timestamp before passing to reasoning agents.

5. Practical Tools / Patterns

Syslog limitation: As you noted, fields vary — solution is a normalization gateway that canonicalizes all logs/events into a shared schema (`event_time`, `ingest_time`, `source_id`, `uncertainty`).
Observability frameworks: OpenTelemetry / Jaeger / Zipkin use trace & span IDs + NTP/PTP to unify timing across layers.
Event bus guarantees: Kafka, Pulsar, etc., ensure ordering per partition; you still need clocks for cross-partition causality.
Complex Event Processing (CEP) engines: They implement "temporal windows" with late-event handling and watermarking.

6. In Agentic Systems (your setting)

Each agent sees "causal envelopes" not raw timestamps: e.g., `Event(A) @ [12:00:01.002 ± 3ms]`.
Orchestrator/Time Service agent normalizes events:

- Aligns clocks (NTP/PTP),
- Canonicalizes timestamp fields,
- Adds provenance (which layer's clock),
- Issues watermarks ("safe up to T").

Agents' reasoning is then expressed in temporal logics with slack (e.g., "if login-failed followed by account-locked within 10s ±1s, trigger escalation")

Synchronization is achieved by global clock sync (NTP/PTP) + event normalization (canonical timestamps) + logical/causal clocks for ordering + uncertainty windows for safety. In practice, a dedicated "Time/Provenance Agent" often sits in the architecture to buffer, reorder, and annotate events before downstream agents consume them.

1. 📌 Starting Point: Original Requirements

In agent-based systems, requirements are usually given as high-level objectives (sometimes vague, sometimes constrained).
Example: “Ensure safe delivery of supplies to location X.”
These are not yet actionable for an agent — they need to be processed into goals.

2. 🎯 Goals Composition

Definition: Combining multiple requirements or subgoals into a coherent higher-level goal.
How it works:

- If two requirements overlap or are interdependent, they may be merged.
- Example: *“Deliver supplies” + “Minimize exposure” → Composite goal: Deliver safely while minimizing risk*.

Agentic role: Ensures that agents do not treat requirements in isolation but as part of a system of intent.

3. ✂ Goals Splitting (Decomposition)

Definition: Breaking a complex or abstract goal into manageable subgoals that can be executed by an agent or a group of agents.
How it works:

- Goal “Deliver supplies safely” →
  - Plan route selection.
  - Avoid hazardous zones.
  - Monitor vehicle health.
  - Confirm package receipt.

Framework function: Provides hierarchical task networks (HTN) or similar structures where high-level goals → lower-level operational goals.

4. Goals Derivation

Definition: Deriving specific goals from general requirements using constraints, context, and reasoning.

How it works:

- Original requirement: "Maintain system stability."
- Derived goals:
  - Keep CPU temperature < 80°C.
  - Ensure error rate < 1%.
  - Balance resource allocation across processes.

Agentic role: Converts vague mission statements into measurable, actionable, verifiable goals.

5. Conflict Handling

Goal conflicts often arise in multi-goal or multi-agent setups.

- E.g., "Minimize delivery time" vs. "Minimize exposure risk."

Frameworks resolve this via:

- Priority hierarchies.
- Utility-based reasoning.
- Constraint satisfaction.

6. Integration in Agentic Frameworks

In practice:

1. Requirement ingestion → agent interprets mission.
2. Goals composition → build composite objectives.
3. Goals splitting → generate subgoals for action layers.
4. Goals derivation → ground subgoals in specific metrics/constraints.
5. Execution + monitoring → track progress and revise if context changes.

✅ In summary:

Composition = merging goals into a unified intent.
Splitting = decomposing into smaller subgoals.
Derivation = translating abstract requirements into concrete, operational goals.

Pipeline sequence

Original Requirements → captured as mission objectives.
Goals Composition → merge and align multiple intents.
Goals Splitting → decompose into manageable subgoals.
Goals Derivation → refine into contextual, measurable targets.
Execution & Monitoring → action, feedback, and adaptation.

Q: How do agents decide
- when compose/split/derivate
- who takes the duty
- what about conflicts handling

1. ⚙ When to Compose, Split, or Derive Goals

Agents usually rely on triggers and context checks:

Compose goals

- Trigger: multiple incoming requirements overlap, share resources, or are mutually dependent.
- Example: “Collect sensor data” and “Preserve battery life” → compose into “Collect data with minimal energy cost.”
- Mechanism: utility fusion (maximize combined utility subject to constraints).

Split goals

- Trigger: a goal exceeds the agent’s capability or requires sequential/parallel substeps.
- Example: “Deliver package to destination” → navigation + obstacle avoidance + delivery confirmation.
- Mechanism: hierarchical task networks (HTN), recursive decomposition.

Derive goals

- Trigger: vague or abstract requirements need operational grounding.
- Example: “Maintain safety” → concrete thresholds like max temperature, max error rate.
- Mechanism: rules, domain ontologies, or constraint satisfaction.

2. Who Takes the Duty

In multi-agent frameworks, delegation is crucial:

Capability-based allocation

- Each agent maintains a profile (skills, resources, trust level).
- Goals/subgoals are mapped to the agent best able to execute them.

Market- or contract-based allocation

- Agents "bid" for goals based on utility/cost (Contract Net Protocol).
- Efficient for distributed coordination.

Role-based allocation

- Predefined roles (e.g., leader, monitor, executor).
- Subgoals flow to agents with the designated role.

3. Conflict Handling

Conflicts arise in two dimensions:

Between goals (internal to one agent)

- Example: "Minimize time" vs. "Minimize energy use."
- Resolution methods:
  - Priority ordering (hard-coded or context-sensitive).
  - Multi-objective optimization (Pareto fronts, weighted sums).
  - Meta-reasoning (agent reflects on which goal matters most now).

Between agents (distributed)

- Example: two UAVs both plan to occupy same air corridor.
- Resolution methods:
  - Negotiation protocols (bargaining, mediation).
  - Coordination strategies (swarm consensus, auction allocation).
  - Arbitration (leader or central authority resolves).

4. Integration in Agentic Frameworks

Most agent frameworks (e.g., BDI: Belief–Desire–Intention) embed this into deliberation cycles:

1.Perceive → environment and requirements.
2.Deliberate → decide composition/splitting/derivation.
3.Allocate → self or other agents.
4.Act → execute subgoals.
5.Revise → resolve conflicts, update priorities.

Formally, this is often modeled as goal-lifecycle automata with transitions triggered by context and feasibility checks.

Formally, this is often modeled as goal-lifecycle automata with transitions triggered by context and feasibility checks.

☑ In summary:
When: composition, splitting, derivation are triggered by complexity, overlap, or vagueness.
Who: allocation follows capability, contracts, or roles.
Conflicts: handled by priority, optimization, or negotiation.

(Goal → {Compose, Split, Derive} → Assign Duty → Resolve Conflicts → Execute)

→ Object-Z / Vienna Method (just to say!)

Past Revisited

BARCELONA
2025

History revisited (i)

- Requirements
- Requirements tracability
- Pre-post conditions
- Control policies (Definition/Access Points)
- Agent contract agreements
- SLA/SLO agreement
- Formal specification of interactions (V&V)
- (Formal Robust Protocols)
- Unique standard framework (s) (Eclipse, as an example)
- Patterns, Artefacts, Software reuse
- Customized (embedded) agents
- Formal agent communication (trusted exchanges)

History revisited (II)

SLA/SLO specifications
UML (semi-formal) specification (another tens, or hundreds)
SDL, LOTOS – protocol formal specifications
Patterns Catalogues
Policy Formal Definitions/Frameworks (type, actions, guarantees)
Activities: actions, plans (par. & seq., actions, temporal aspects, conflicts, mitigation, etc.)
Versioning control (configuration mgmt)
Support for Legacy systems
! 99,999 service availability

Formal Methodologies

Rebecca Wirfs-Brock - Responsibility-Driven Design (RDD) (OOPSA 1989+)
Bertrand Meyer - Design by Contract (DbC) - Eiffel programming language (~ 1986 +)
Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides,
with a foreword by Grady Booch: - Design Patterns (reusable elements) (OOPSLA, 1994)

Agents

AT&T - Monitoring and Management system had ~ 600 specialized agents (~2000)
Cisco Systems - inside each router (~50 agents, fault, performance, etc.)
ODP (1990 - Trader - formal definition), CORBA (1990 Broker - SDL specifications), TINA, etc. (Manager) --- > Agentic (Orchestrator).

Q; Status quo

Design time consuming
Limited knowledge (of some, all)
Human-depending productivity
High skilled experts (cost)
Long learning curve
Poor code documentation / manuals

BACK TO AGENTIC FRAMEWORK (again LLMs/LCMs and some standard agents)

Q: Status quo

Design time consuming
Limited knowledge
Human-depending productivity
High skilled experts (cost)
Long learning curve
Poor (code) documentation / product manuals

A+

Quick design (more than Agile approach)
Prompt information at large scale [caveat-pre-knowledge is needed]
Automation-based productivity (less human workforce0
Min high skilled exerts (prompt experts and tools knowledgeable)
Long learning curve (almost instant; see prerequisites)
Instant generation of documentation / manuals

A- (to be improved)

Deskilling
Highly depending on a few individuals
Lack of or not at a required level of Explainability, Ethics(Opaqueness)
Uncontrolled bias (European Act, USA ….)
Great ROI (for some)
Unreliable information (hallucinations, unintended (or not) consequences,
Biased, unreliable and not trustable communications between agent

A-/+ (to be improved)

Decision of NLP is not accurate (see Syslog payload field)
Difficult cu catch errors/mistakes
Bias in data sets (V&V)
AI literacy, Data literacy

Ready to be Shipped

AWS Agents for Bedrock / AgentCore – Build & run enterprise agents with tool/action execution, multi-agent setups, and production monitoring; AgentCore focuses on getting agents to production at scale. Amazon Web Services, Inc.+3Amazon Web Services, Inc.+3Amazon Web Services, Inc.+3

Google Vertex AI – Agent Builder / Agent Engine – Managed runtime for agents with sessions, memory bank, built-in tools (e.g., grounding via Google Search), and evaluation; positioned for multi-agent "experiences." Google Cloud+1

Microsoft Azure – AI Foundry Agent Service & Microsoft Agent Framework – A unified runtime (and open-source framework) to orchestrate models, tools, safety, identity, observability; converges earlier work like AutoGen/Semantic Kernel. Microsoft Learn+2Microsoft Azure+2

Anthropic – Claude "Skills", Agent SDK, and Multi-Agent Research – Organization-scoped skills (instructions/scripts/resources) loaded on demand; SDK for building agents; published engineering notes on multi-agent research flows. Anthropic+2Anthropic+2

OpenAI – AgentKit (DevDay 2025) – Toolkit and SDK for designing, evaluating, and deploying agents with tool calling, chaining, and multi-agent orchestration; visual builder surfaced at DevDay. Composio+4OpenAI+4OpenAI+4

LangGraph (by LangChain) – Open-source agentic state-machine framework (persistence, streaming, debugging, deploy) with a clear "workflow vs. agent" distinction; widely used in production demos. LangChain AI+1

Salesforce – Agentforce 360 – Enterprise "agentic stack" tying agents to CRM/Data Cloud/Slack with low-code builder, voice, and multi-agent orchestration—very much an agentic-enterprise platform. Salesforce Ben+4Salesforce+4Salesforce Investor Relations+4

Mapping Tools and Skills

Platform	Security controls (PHI/PII)	Auditability	Multi-agent patterns	Evals / guardrails	Cost levers	Healthcare fit (quick take)
AWS Bedrock – AgentCore	IAM, VPC, KMS at-rest/in-transit; session isolation; integrates with AWS compliance stack (BAA eligible via AWS services). Amazon Web Services, Inc.+2Amazon Web Services, Inc.+2	CloudTrail traces; AWS Audit Manager alignment for controls. AWS Documentation	Tool/action execution, long-running tasks; partners show multi-agent via LangGraph+Bedrock. Amazon Web Services, Inc.	Native policy controls; combine with Bedrock guardrails; healthcare blog examples. Amazon Web Services, Inc.	Pay per model/runtime; reuse existing AWS networking & security investments. Amazon Web Services, Inc.	**Strong**: mature identity/networking, clear audit paths; good for PHI with BAA on underlying services. Amazon Web Services, Inc.
Google Vertex AI – Agent Builder	Runs on Google Cloud security baseline; HIPAA program available on GCP. Google Cloud+1	Cloud logging; enterprise policies across Vertex stack. Google Cloud	"Agent" runtime with sessions, grounding; multi-agent experiences emerging. latenode.com	Google evals/grounding; policy & data controls documented. Google Cloud	Vertex per-use + GCP infra (net/storage) pricing. Google Cloud	**Strong**: good compliance posture on GCP; check BAA scope for specific services you enable. Google Cloud
Microsoft Azure – AI Foundry Agent Service	Azure security/identity; HIPAA/BAA available across eligible services; documented data-handling for "Azure Direct Models." Microsoft Learn	Centralized monitoring & logging under Azure; enterprise audit toolchain. latenode.com	Converges SK/AutoGen lineage into managed agent runtime. latenode.com	Responsible-AI features; enterprise policy controls. Microsoft Learn	Azure consumption + model/runtime costs. latenode.com	**Strong**: identity + compliance depth; good SSO/tenant isolation story for PHI. Microsoft

Mapping Tools and Skills

Anthropic – Claude "Skills" & Agent SDK	Enterprise workspace controls; skills are org-scoped; emphasis on safe code execution. The Verge+1	Console & SDK surfacing reasoning/steps; versioned skills. Tom's Guide	Composable "skills" = modular agent capabilities; aligns with multi-agent/task routing. Anthropic	Safety focus; guidance for executable tools. Tom's Guide	Model + enterprise plans; skills reduce bespoke dev overhead. The Verge	**Good**: great for controlled capabilities; pair with cloud runtime that provides BAA/logging. The Verge
OpenAI – AgentKit / Agent Builder (DevDay '25)	Enterprise controls via Connector Registry/admin panel; security posture highlighted at launch. OpenAI+1	Built-in traces/step grading for agents. OpenAI+1	Visual multi-agent canvas + SDK; tool calling and orchestration. OpenAI	Integrated evals (datasets, trace grading, auto prompt optimization). OpenAI+1	Platform + eval infra pricing; faster iteration lowers total build cost. Superprompt	**Good**: strong evals; confirm PHI/BAA posture for your deployment path before handling live patient data. OpenAI
LangGraph (LangChain)	Open-source; security depends on hosting (often paired with Bedrock/Azure). Amazon Web Services, Inc.+1	Tracing/debugging built in; full audit via your infra logs. Sider	Agentic **state-machine** with branching/loops; clean multi-agent hand-offs. Sider	Guardrails via ecosystem; deterministic control via graph/state. Sider	Infra-only (open-source); pay for models/compute where deployed. Amazon Web Services, Inc.	**Excellent as a control plane** when combined with a HIPAA-eligible cloud (e.g., AWS/GCP/Azure). Amazon Web Services, Inc.
Salesforce – Agentforce 360	Built for enterprise trust/compliance; materials cite HIPAA support & PHI masking, governance. Salesforce+1	Deep audit trails within Salesforce platform; governed data access. Salesforce	Multi-agent orchestration across CRM/Slack/Data Cloud. Salesforce Investor Relations			

Mapping Tools and Skills

How this maps to “agentic” capabilities

Most of the above support, to varying depths:

Tool use & action execution (APIs, RPA-style calls)
Memory/state & sessions (per user/task threads)
Multi-agent orchestration (supervisor/worker patterns)
Grounding & retrieval (search, RAG, enterprise data)
Safety/guardrails & audit (policy, trace logs, identity)

Skill Profiles
Civil engineer
Mechanical engineer
Avionic engineer
Electronic engineer
Software engineer
AI Engineer
Digital engineer
Prompt Engineer
Agentic Framework Engineer
Knowledge engineer
Trust and Alignment Engineer (emerging)
Privacy and compliance Engineer (emerging)

Mapping Tools and Skills

BARCELONA 2025

Era / Focus	Emerging Role	Core Competence
Classical physical systems	Civil Mechanical Avionic Electronic Engineer	Physical laws, design, control
Information systems	Software Engineer	Code, algorithms, computation
Cognitive systems	AI Engineer	Model design, learning, inference
Integrated infrastructures	Digital Engineer	Digitalization, data flows, interoperability, twins
Human–machine interfaces	Prompt Engineer	Language mediation, intent structuring, tool invocation
Autonomous coordination	Agentic Framework Engineer m	Goal management, multi-agent reasoning, policy constraints
Knowledge-based synthesis	Knowledge Engineer	Ontologies, knowledge graphs, reasoning over structured information
Next evolutionary step →	Trust and Alignment Engineer (or) Ethical Systems Engineer	Value alignment, transparency, bias management, explainable autonomy, licensing

Status

We are here, Agentic frameworks are here, too!

QUO VADIS?
Rolling up the sleeves!