# Goals Processing and Task Allocation via Agentic Frameworks

Prof. Dr. Petre Dini, ex-AT&T/Cisco Systems. Inc., USA (retired)

# Themes



- Agents: Before and Now
- Monitoring and Management Agents (Syslog)
- Status: Useful + To be Improved
- Case study: Selling football tickets
- Agentic Design Pipeline
- Selecting/Building Agents
- Requitements → Goals
- Goals Derivation and Processing
- When, Who, Conflict Handling

# Agents Were and Are Available



**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

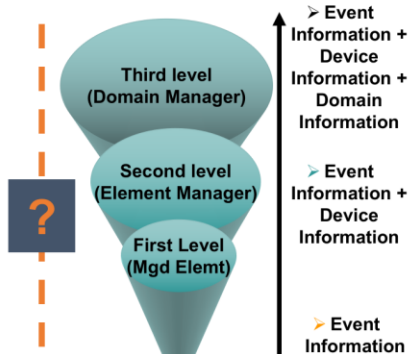
## Industrial Challenges in Working with Events

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

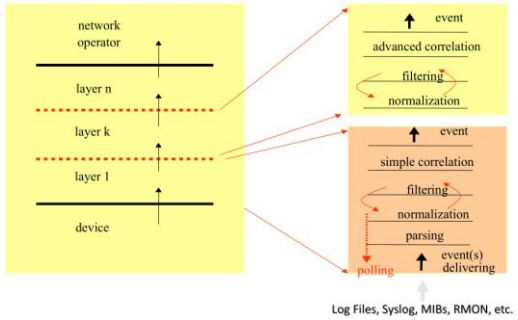
**DEBS 2004**
**Edinburg**

## Bottom-up vs. Top-down

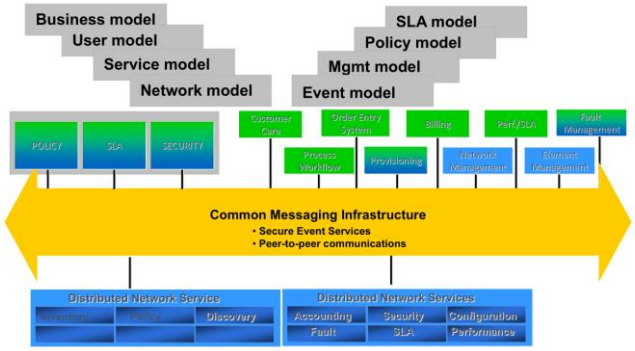
- Domain Manager enriches with domain information
- EMS enriches with multi-device information
- Notification Engine collects OS notifications



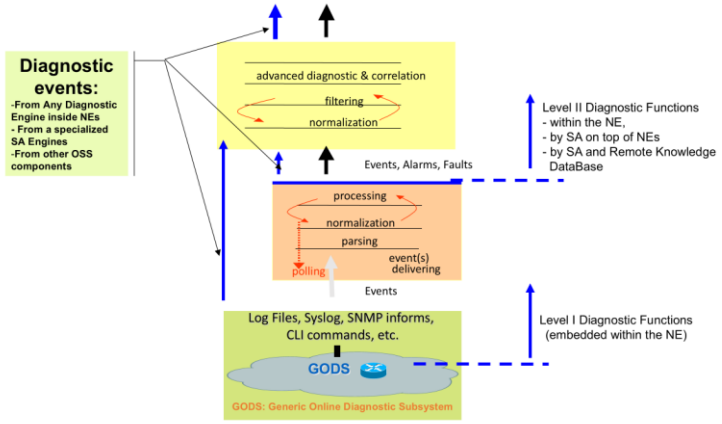

## A Layered Processing View




## Communication Bus

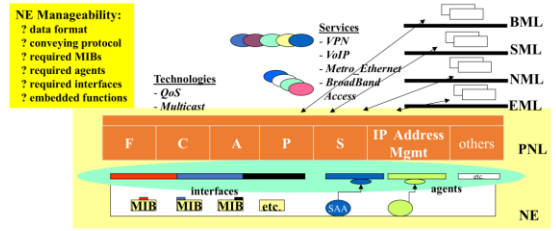



## Multi-level diagnostic

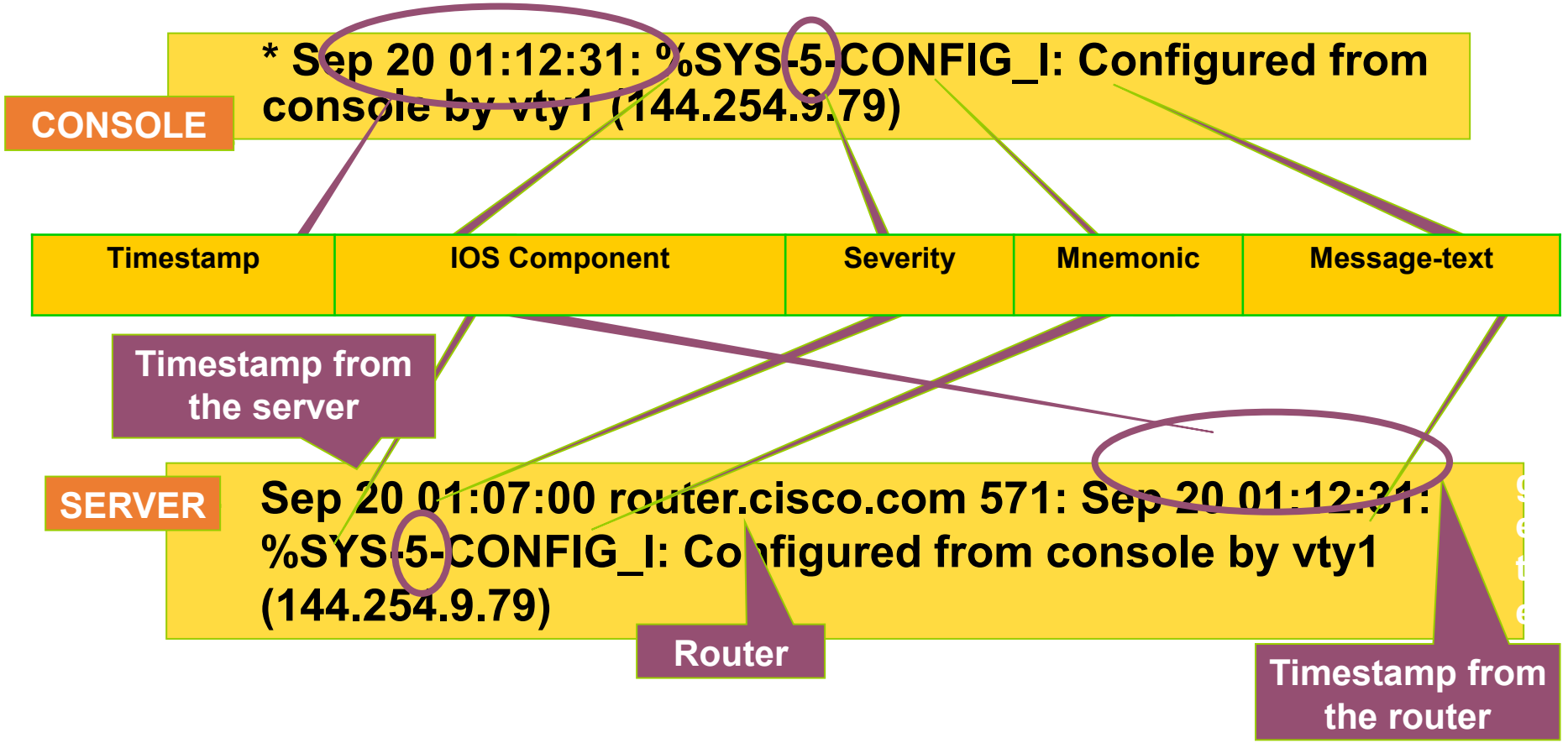



## Syntax Issues

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



# Syslog Message "Body" Format in the IOS



# Before and After



## 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

# Achievements + To be Improved



## 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

# Case study: Selling Football Tickets



**(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, …)**

# Agentic Design Pipeline



## 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.

## 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
```

# Requirements → Goals



## 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.

# Goals Derivation and Processing (i)



## 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*.

### a. Goals Composition: 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**.

### b. Goals splitting (decomposition): 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.

# Goals Derivation and Processing (ii)



## c. Goals Derivation: 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**.

## d. 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.

## e. Integration in Agentic Frameworks

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.

## 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.




14

# Who? + Conflict Handling



## WHO Takes the Duty

In **multi-agent frameworks**, delegation is essential:

### 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.

## Conflict Handling

- **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).