

CONTEMPORARY AGENT TECHNOLOGY: LLM-DRIVEN ADVANCEMENTS VS. CLASSIC MULTI-AGENT SYSTEMS

Costin Bădică

University of Craiova, Romania

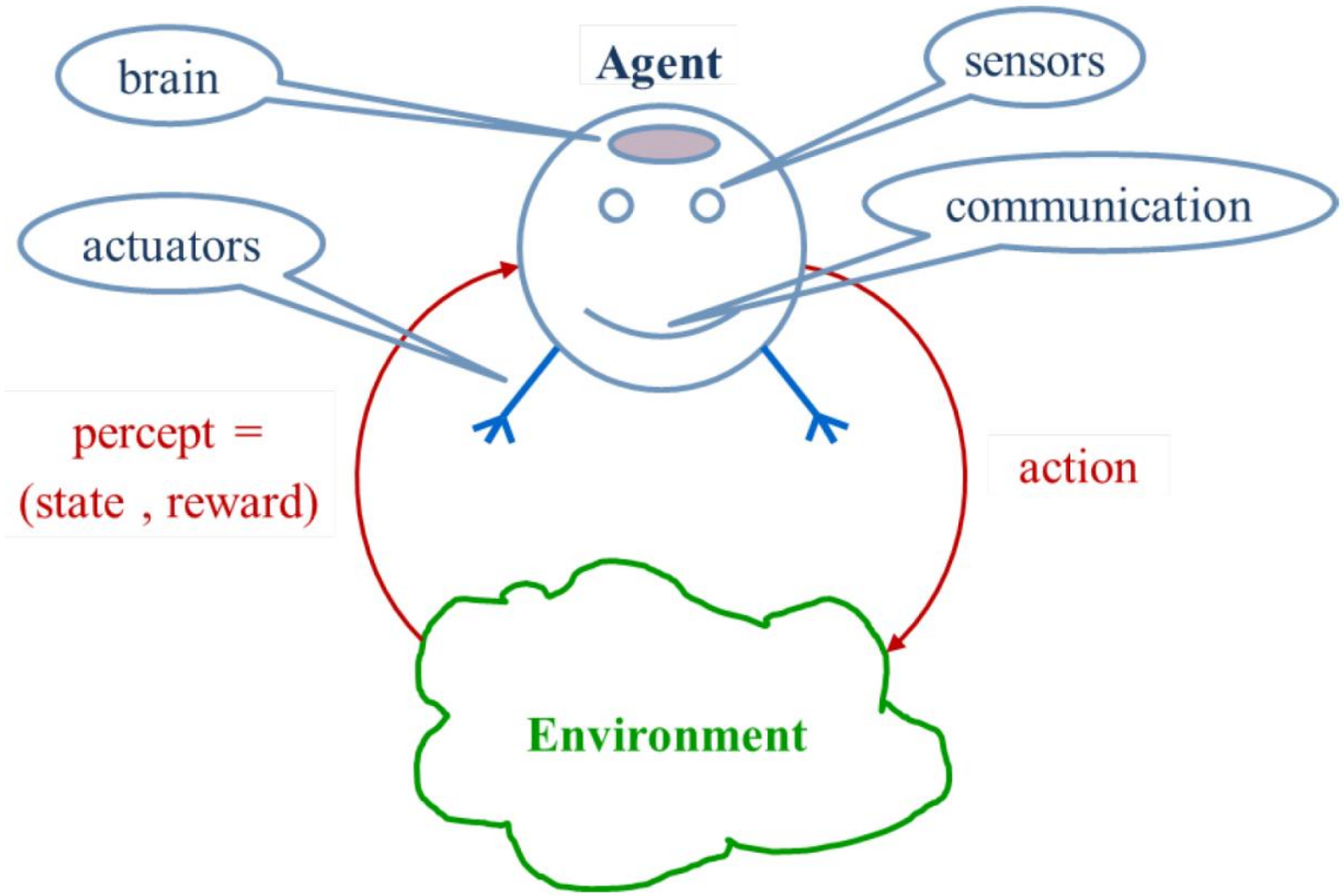
Agenda

- Introduction
- LLM-based agents
- KBS & BDI Agents vs LLMs
- Agents & Artifacts vs Agents with Tools
- Communication & Interaction Protocols
- Agentic AI vs (Agent-based) Workflows

Origin of Agents in CS

- Early days of Distributed AI research – Carl Hewitt's *actor model* in the 1970s.
- New impetus in the '90s based on developments of *interactive*, *ubiquitous*, and *distributed computing*.
- Chabot era after 2020 gave new impetus to *LLM-based agents & Agentic AI*.

What is an Agent?



A Paradigm Shift

Classic MAS

The Foundation

- Rich history rooted in Game Theory and symbolic AI.
- Agents with predefined rules and behaviors (e.g., Jason, JADE).
- Considered a niche area of computer science.

The LLM Catalyst

The Revolution

- A shift from symbolic to sub-symbolic AI.
- LLMs act as the "cognitive core" or "brain" for new agents.
- Moves agent tech from theoretical to practical, real-world applications.

- Agentic AI / LLM Agent / LLM MAS:
  - New buzzword for an established field powered by new technology. The core principles of agency remain relevant.

Classic KBSs vs. LLMs

- *Components*: Struct. KB (facts, rules) + Inference Engine
- *Interaction*:
  - Tell to add new, explicit facts
  - Ask to query them via logical inference.
- *Reasoning*:
  - Glass Box - traceable brittle and hard to scale.

Classic KBSs:

- *Knowledge*: Implicit in the weights of a neural network.
- *Interaction*:
  - No true Tell service. Knowledge update by inefficient fine-tuning or temporary in-context learning.
  - Ask performs probabilistic pattern-matching.
- *Reasoning*:
  - Black Box - powerful & flexible non-deterministic & hallucinations

LLM

PROLOG PROGRAMMING

```
should_i_learn(Language) :-
  curious_about_programming_languages,
  already_own_book_covering(Language).

curious_about_programming_languages :-
  reading(seven_languages_book).

already_own_book_covering(Language) :-
  covers(Book, Language),
  own(Book).

reading(seven_languages_book).
own(seven_languages_book).
```

Rules

"If *you're curious about programming languages* **and** *you already own a book that covers whatever language you're asking about* **then** *yes, you should learn it*"
(Jeff Dallien, Quora, 2011)

```
covers(seven_languages_book, clojure).
covers(seven_languages_book, haskell).
covers(seven_languages_book, io).
covers(seven_languages_book, prolog).
covers(seven_languages_book, scala).
covers(seven_languages_book, erlang).
covers(seven_languages_book, ruby).
```

Facts

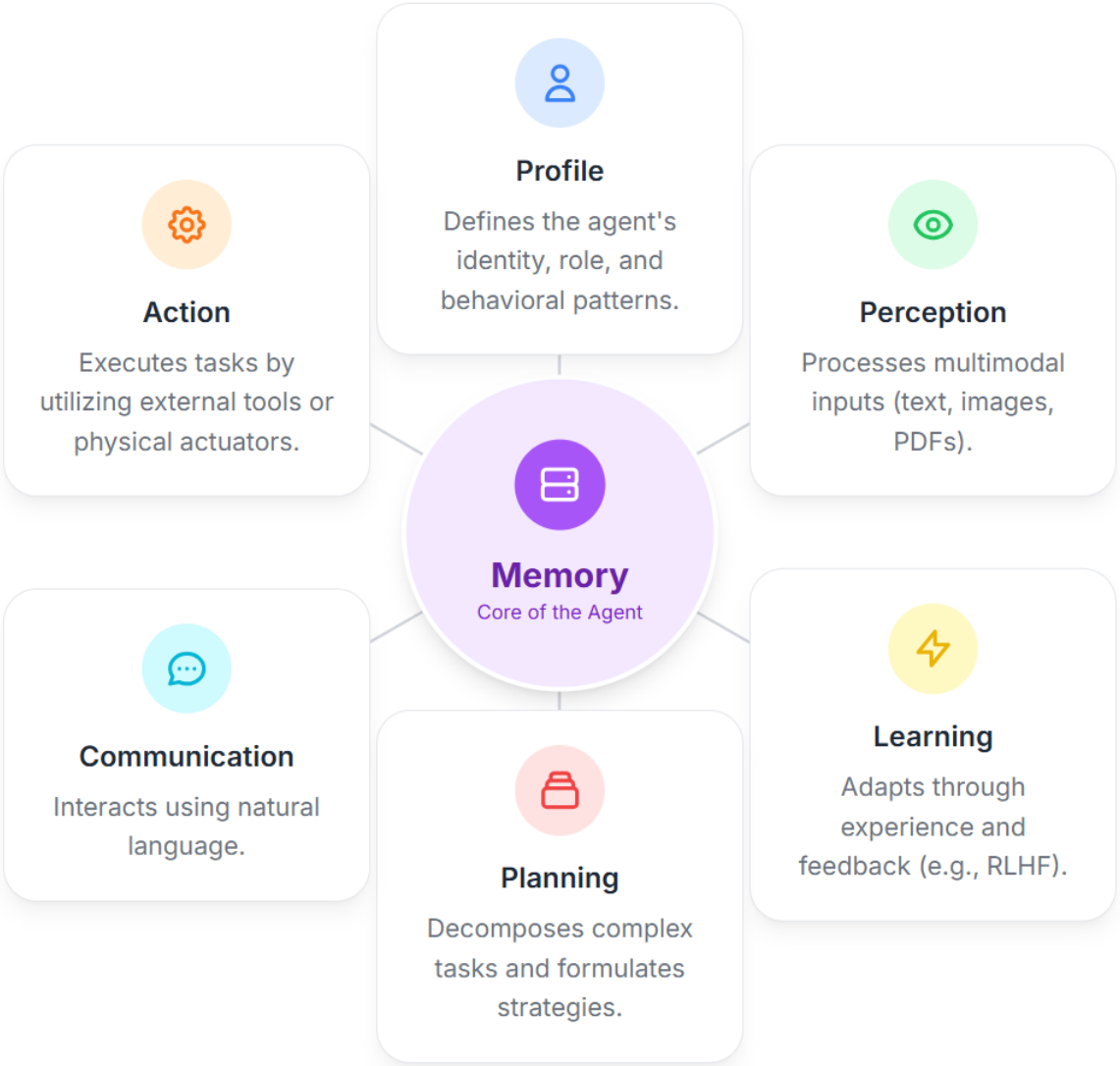
```
| ?- should_i_learn(prolog).

yes
```

Query

Architecture of an LLM-Agent

- An **LLM-agent** has several key architectural pillars, with **Memory** at the core.



Memory & Planning

Memory

Overcoming Context Limitations

Short-Term

Uses the LLM's context window for immediate dialogue history.

Long-Term

Stores information in skill libraries or experience repositories.

Retrieval Augmented Generation (RAG)

Expands knowledge by accessing external databases and documents.

Planning

From Goal to Execution

Task Decomposition

Breaks down large problems into smaller, manageable sub-tasks.

Feedback-Driven Iteration

Refines plans based on feedback, guidance, or self-critique.

Execution & Perception

How do LLM-agents interact with and understand the world ?

Action Execution

Interacting with the World

Tool Utilization

- Perform precise calculations.
- Access real-time information.
- Execute code or interact with APIs.

Physical Interaction

For embodied agents (robotics), high-level plans are translated into physical movements.

Perception: Beyond Text



- ✓ Natively **multimodal** foundation models.
- ✓ Perceive and analyze **images, video, and files.**
- ✓ Operate in more **complex, realistic** environments.

Architecting LLM-Powered MAS

- When multiple LLM-agents collaborate, their interaction is defined by the system's topology and control flow.



Centralized Control

A hierarchical model with a central "manager" agent.

- The manager decomposes tasks, assigns them to worker agents, and integrates the results.

Examples: MetaGPT, AutoAct



Decentralized Control

Peer-to-peer interaction without a central coordinator.

- Agents self-organize and coordinate through emergent communication.

Example: AutoGen



Hybrid Control

Combines elements of both for local and global coordination.

Collaboration in LLM Societies

- LLM-agents use various strategies to work together, beyond simple instructions.



Cooperative Task Decomposition

A primary agent breaks down a task and assigns sub-tasks to specialists.



Debate and Critique

Agents challenge each other's outputs to refine plans and produce more robust results.



Role-Playing & Social Simulation

Agents adopt specific personas (e.g., "Product Manager," "Engineer") and follow social norms associated with those roles.



Negotiation & Bidding

Agents dynamically negotiate over resources or responsibilities.



Emergent Collaboration

Collective goals are achieved without direct commands, as agents react to a shared environment.

Revisiting Core Agent Properties

- How do LLM-agents fit against the classic definitions of agency?



Autonomy

Can operate without direct human intervention, but behavior is heavily guided by initial prompts.



Reactivity

Can perceive and respond to environmental changes fed through prompts.



Pro-activeness

Can exhibit goal-directed behavior by generating and executing plans.



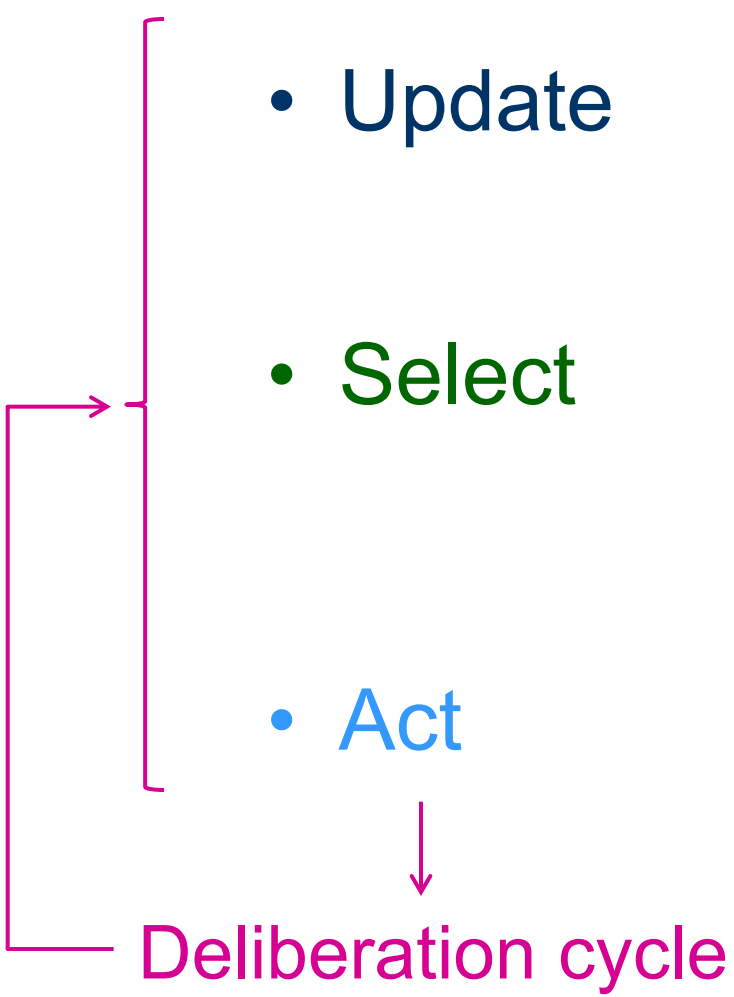
Social Ability

Natively strong due to their foundation in natural language.

BDI Concepts

- BDI follows practical reasoning model inspired by Daniel Dennett's "intentional stance" (1971) = *reasoning towards actions*.
- BDI agents are endowed with:
  - *Beliefs*
  - *Desires* or *goals*
  - *Intentions* or *plans*
- Agent behavior is *event-driven*.

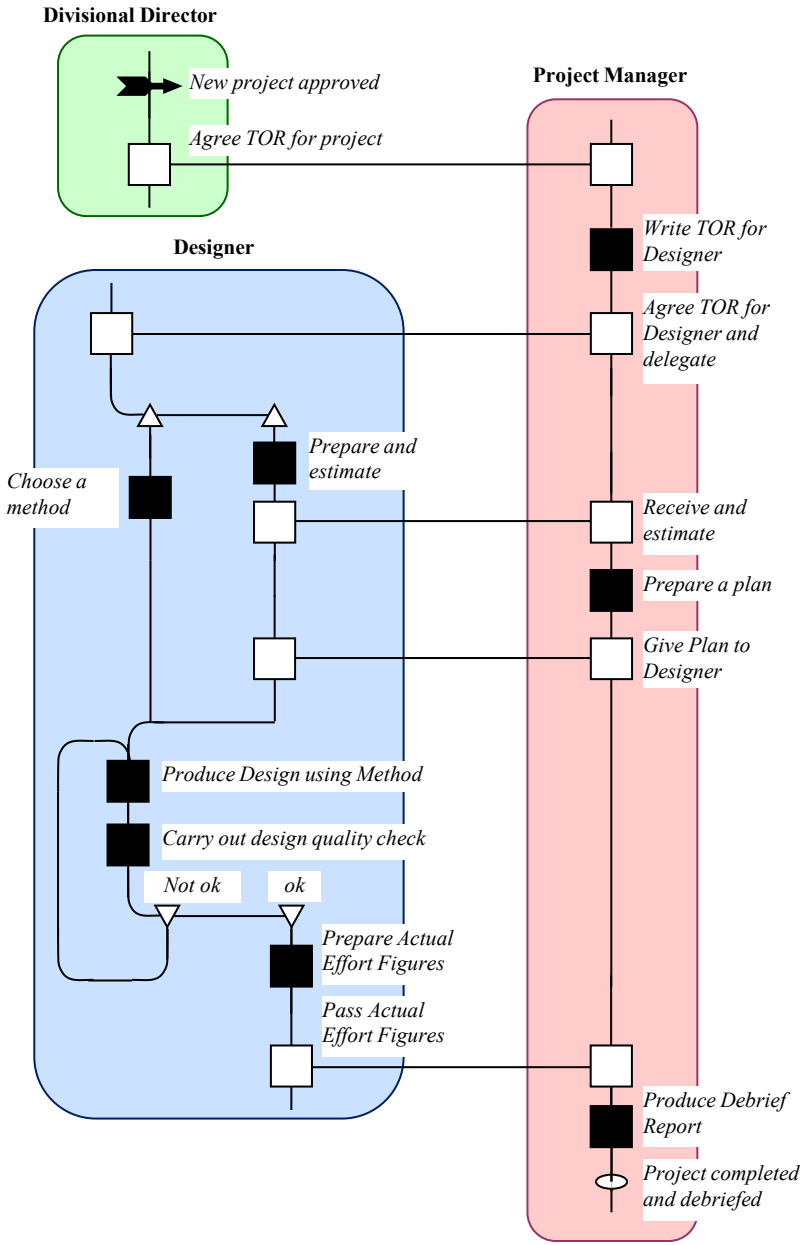
BDI Reasoning Steps



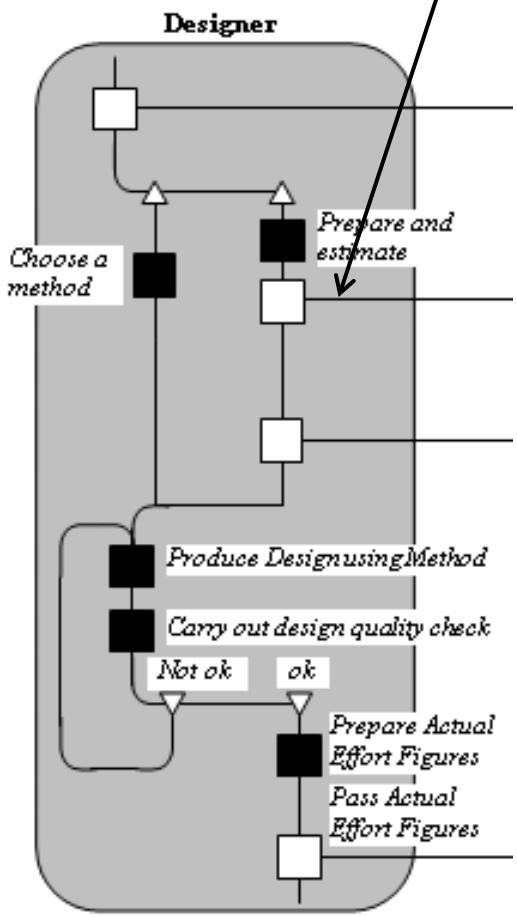
- Update
  - perceive
  - communicate
- Select
  - event
  - applicable plan
  - intention = course of action
- Act
  - internal / external
  - mental note
  - adopt / drop goal:
    - test / achievement

Business Agents

- Three roles:
  - Divisional Director
  - Project Manager
  - Designer



KNOWLEDGE BASE



"Receive and estimate"

Send shared belief

```
% rule(Action, StateIn, StateOut).

rule(task("Fork parallel threads"), [d1],
  [d2,d3]).
rule(task("Prepare and estimate"), [d3],
  [d4,s(pm,d4)]).
rule(task("Receive and estimate"),
  [d4,r(pm,pm3)],[d5,s(pm,d5)]).
...
rule(choice("Carry out design quality check",
  [ok,nok]), [d9], [[ok,d10(ok)],
  [nok, d10(nok)]]).
```

Receive shared belief

The BDI Model in the Age of LLMs

- Classic Belief-Desire-Intention is valuable for structuring LLM-agent behavior.

Classic BDI Agent



Agents have **Beliefs** (world state), **Desires** (goals), and **Intentions** (commitments to plans).

LLMs Enhance BDI Agents

- **Enriching Beliefs:** Digesting unstructured text to build nuanced world models.
- **Formulating Desires:** Interpreting high-level, ambiguous human commands into concrete goals.
- **Dynamic Planning:** Generating and adapting plans "on the fly" instead of relying on pre-programmed libraries.

Key Challenges

⚠ **Hallucinations** can corrupt agent beliefs.

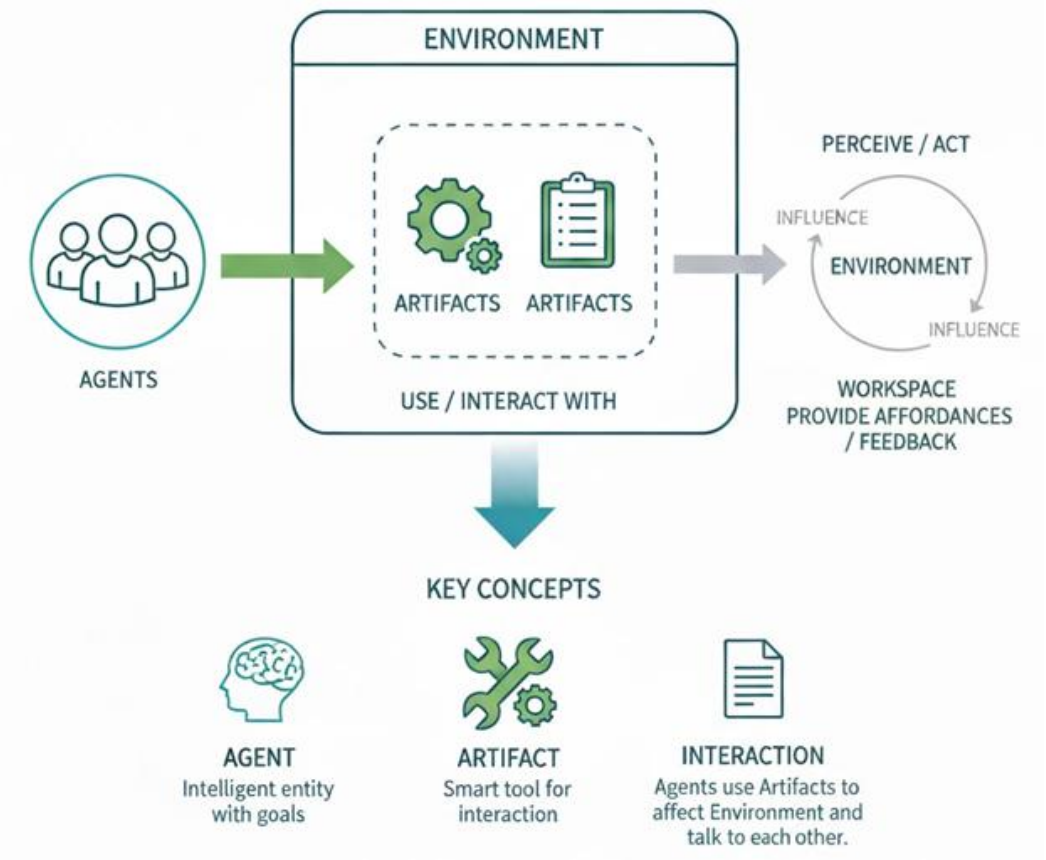
⚠ **Non-determinism** makes behavior less predictable.

BDI & LLMs: Examples & Enhancements

- ChatBDI: https://github.com/VEsNA-ToolKit/chatbdi
  - Hybrid architecture using an LLM for “human interaction" while retaining the formal BDI structure.
  - The BDI framework provides the goal-directed behavior, while the LLM acts as the "language mediator" to understand human requests and generate natural language responses between NL and KQML.
- NatBDI: https://github.com/yukioichida/nat-bdi
  - Architecture for agent instruction & programming using NL. It adapts the BDI reasoning for NL environments.
    - Beliefs are represented as natural language descriptions.
    - Plans are written in a controlled natural language.
  - Plan Selection uses a Natural Language Inference to match plan's context with agent's beliefs.

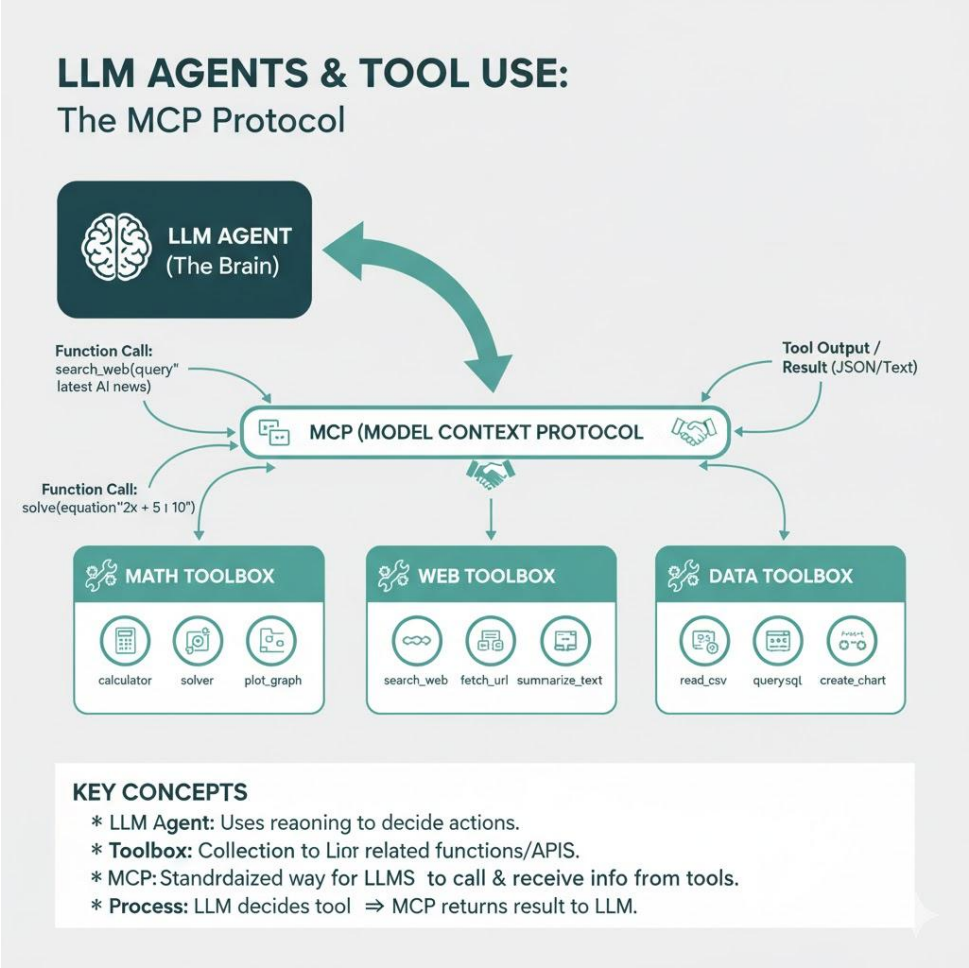
AGENTS & ARTIFACTS

- A & A was proposed by Alessandro Ricci, Mirko Viroli, and Andrea Omicini in 2006.
- Artifacts are “first-class” environment abstractions in MAS design:
  - Persistent
  - Expose operations
  - Stateful
  - Observable by agents
  - Define agent workspace



LLM AGENTS WITH TOOLS

- Environment = vast digital world of APIs, databases, and external services.
- Tool = external resource the agent can use to achieve goal: *web search API*, *calculator*, *code executor*, *db query API*.
- MCP = standard layer between LLM-agents and the *toolboxes*.



Artifacts vs. Tools

The interface between agent ↔ environment has fundamentally evolved.

Classic: Agents & Artifacts



- The environment contains programmable **"artifacts"** with formal, explicit operations.
- Agent reasoning is **symbolic**; it selects a plan that invokes a known operation.

Result: A predictable, verifiable, but rigid system.

Modern: LLM-Agents with Tools

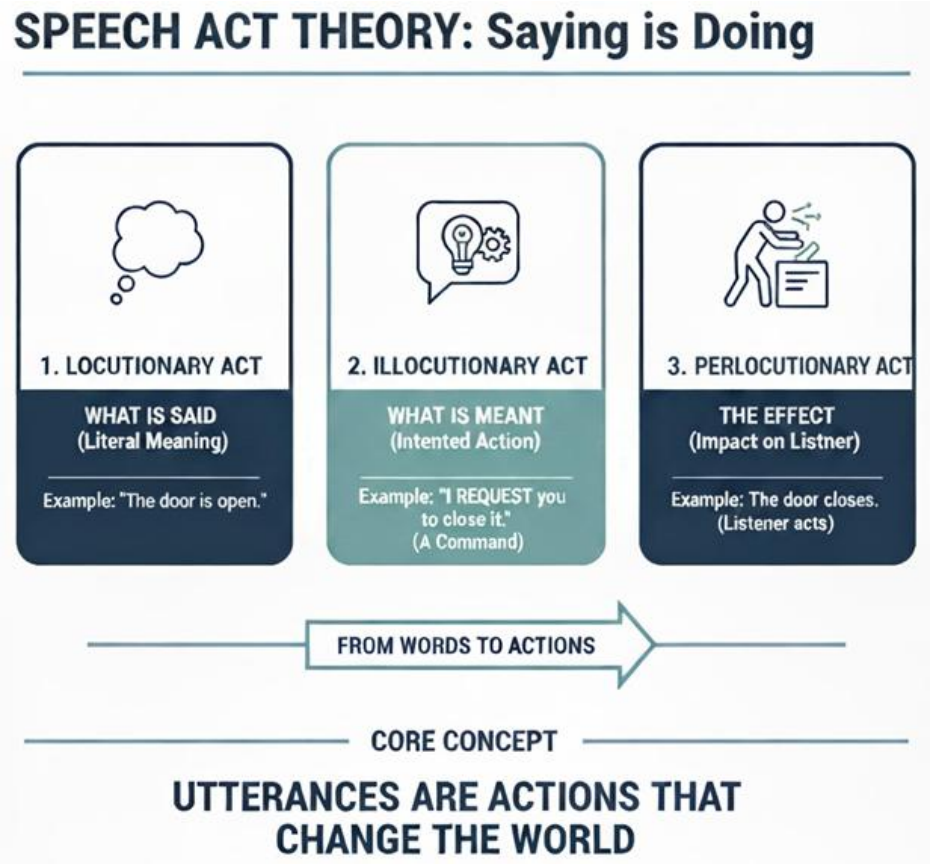


- The "environment" is the digital world of **APIs and services**.
- The interface is **semantic**; the LLM infers how to use a tool from its natural language description.

Result: A highly flexible, but less predictable system.

SPEECH ACT THEORY

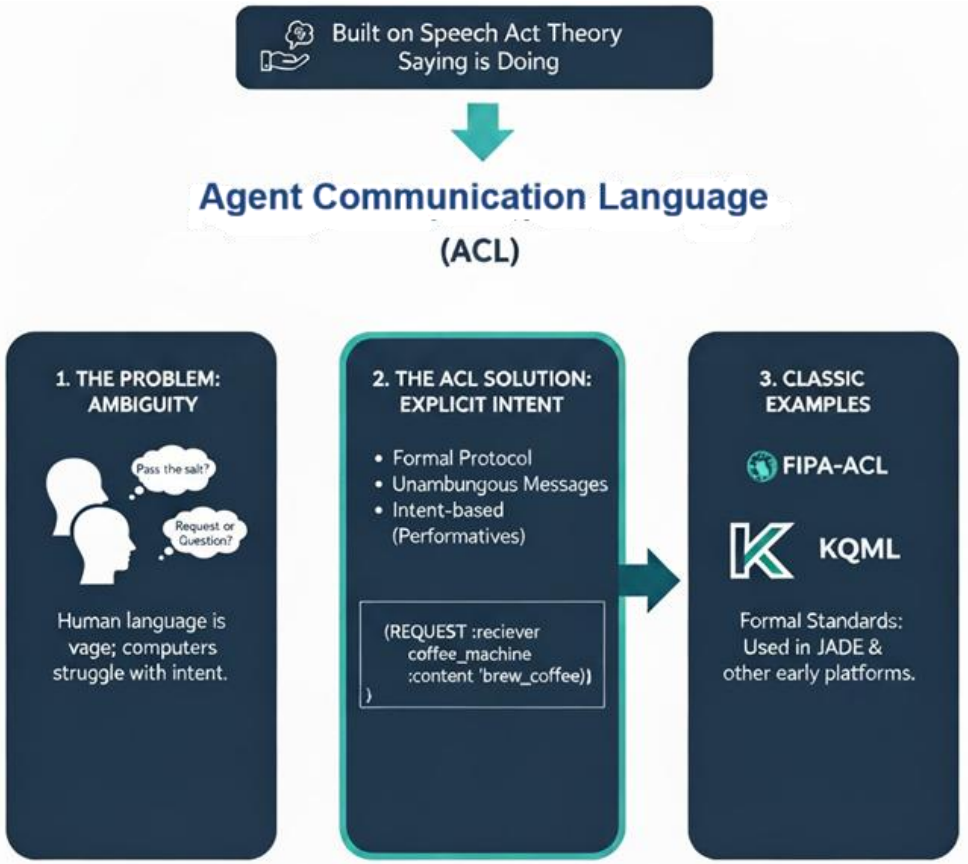
- SAT = introduced by J.L. Austin in 1962 and refined by John Searle in 1969 stating that *saying something is doing something* ⇒ **action**
- A speech act has 3 facets:
  - locution = the content of the utterance;
  - illocution the intention of the emitter;
  - perlocution the effect of the utterance on the receiver.



AGENT COMMUNICATION

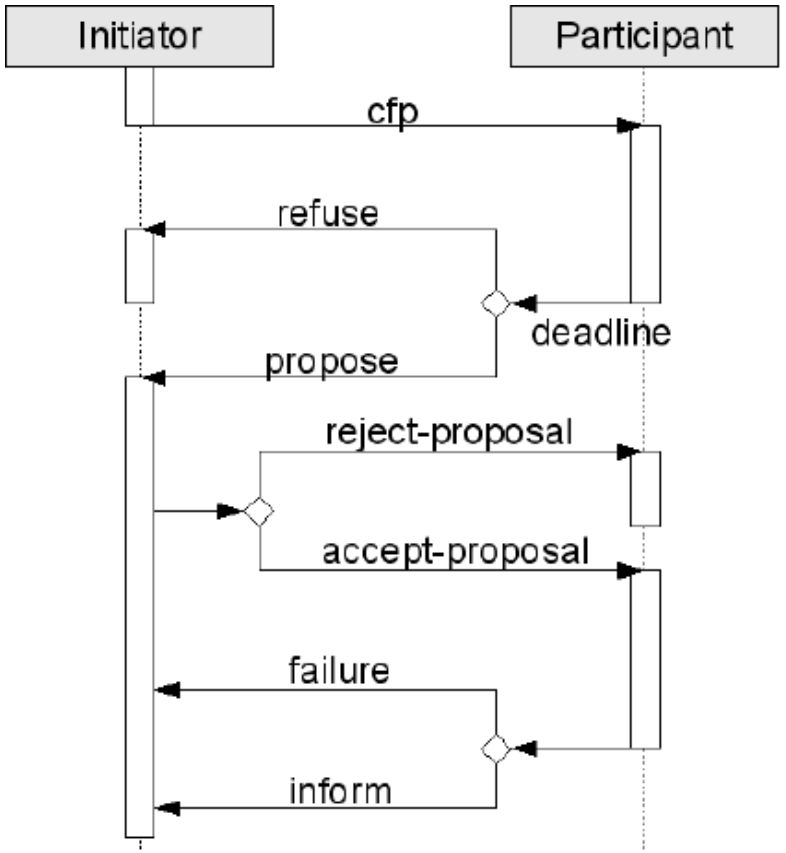
- ACL = formal protocol built on SAT that allows agents to exchange messages with explicit *intent*, e.g. REQUEST, INFORM, PROPOSE.
  - FIPA-ACL (from JADE)
  - KQML (Knowledge Query and Manipulation Language).

```
(request
    :sender BuyerAgent
    :receiver ShopAgent
    :language SL
    :ontology retail-ontology
    :content
        "price(item-66A,P)"
)
```



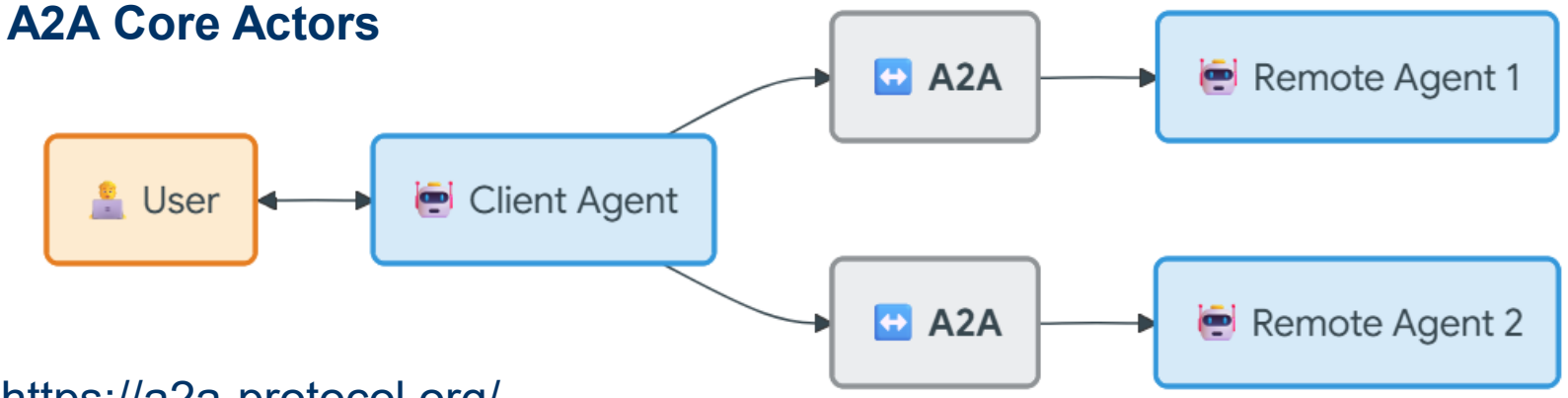
Interaction Protocols

- Agents interact peer-to-peer using conversations. A conversation is an instance of an IP.
- IP = explicit & formal definition of the "*rules of encounter*", i.e. conversations enabling agents' meaningful communication.
- Contract-Net Protocol for *cooperative task allocation*.



Google A2A Protocol

- *Agent-to-Agent (A2A) protocol* = open standard initiated by Google in April 2025 for a universal "common language" for AI agents.
- Interoperability problem of AI agents = standard way for AI agents built on different frameworks (Google ADK, LangGraph, CrewAI) to communicate or work together.



https://a2a-protocol.org/

A2A Communication Elements

Element	Description	Key Purpose
Agent Card	Metadata doc describe agent's identity, capabilities, endpoint, skills, and authentication requirements.	To discover agents & understand how to interact with them.
Task	Stateful unit of work of an agent, with a unique ID and defined lifecycle.	Tracks long-running operations.
Message	Unit of communication between a client and an agent, has content and role ("user" or "agent").	Conveys instructions, context, questions, answers, status, not necessarily formal artifacts.
Part	Fundamental content container (e.g., TextPart, FilePart, DataPart) used within Messages and Artifacts.	Enables exchange of various content types in messages and artifacts.
Artifact	Tangible output generated by an agent during a task (e.g, document, image, or structured data).	Delivers the concrete results of an agent's work as structured and retrievable outputs.

From ACLs to NL Dialogue

- ACL from formal protocols → NL fluid conversation.

Classic: Agent Communication Languages (ACLs)



- Based on **Speech Act Theory** (e.g., FIPA-ACL).
- Messages use explicit **performatives** to state intent.

Strength: Machine-readable, efficient, and predictable.

Weakness: Brittle and requires shared, pre-defined ontologies.

Modern: LLM-Mediated Dialogue



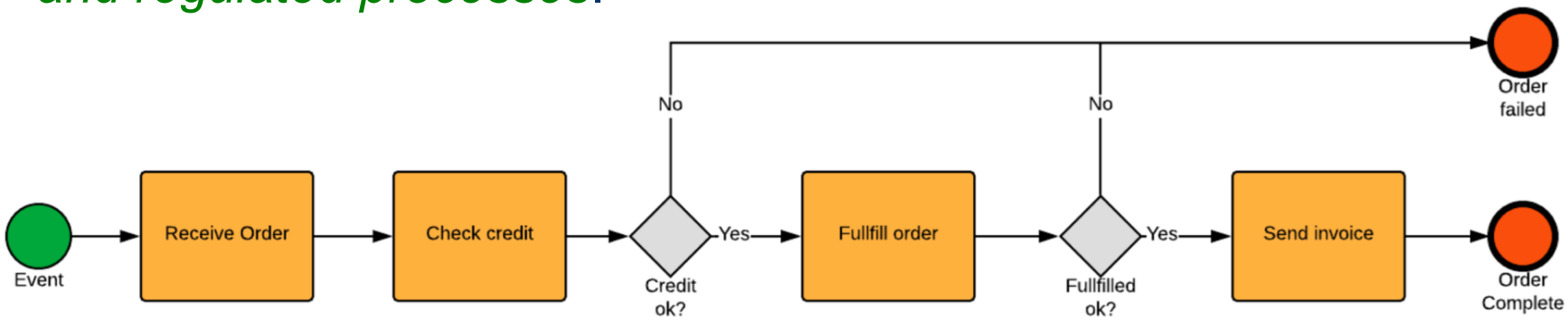
- Agents communicate directly using **natural language**.
- Intent is **inferred** from semantic context.

Strength: Extremely flexible, adaptable, and expressive.

Weakness: Prone to ambiguity, misinterpretation, and security risks.

Classic WfMS vs LLM MAS

- Classic WfMS is designed for explicit process enactment i.e. to faithfully execute a predefined and human-authored process.
- Architecture is centered on a workflow engine acting as an *interpreter for a predefined, explicit process model* expressed in formalisms like BPMN or Petri nets – the system's "source code".
- Execution and Control Flow is *deterministic and state-driven*.
- State, Data & Resource Management: *explicit & centrally controlled*.
- Primary Application of WfMS is *automating high-volume, repeatable, and regulated processes*.



Agent-Based Workflows vs LLM MAS

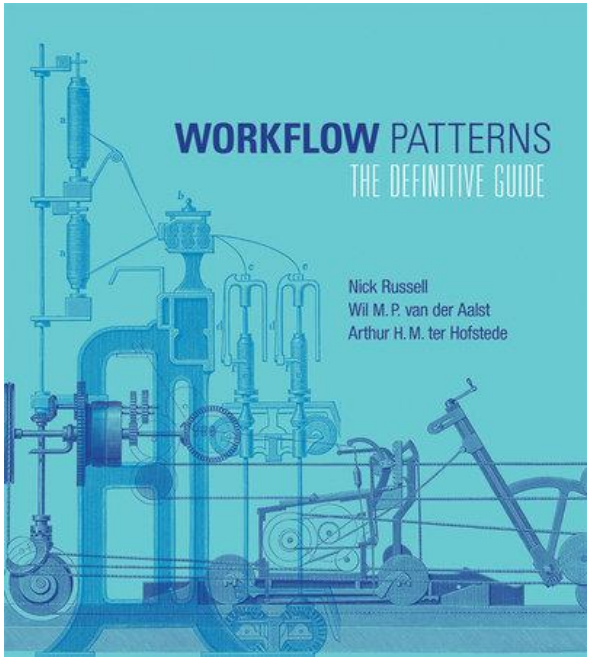
- Agent-based Process Management proposed for the first time by Chang and Scott in 1996 and O'Brien and Wiegand in 1998.
  - Creation: manual activity of analysis, modeling & definition of a business process
  - Provisioning: map resources (people, equipment, comp) to a business process
  - Enactment: execute a business process
- Several designs:
  - Agents assigned to business units. They act as service providers/consumers.
  - Agents as case managers. A case is a process instance capturing a UoW.
- Characteristics:
  - Autonomy is engineered. Agents are explicitly programmed with predefined rules, state machines, or decision-making heuristics.
  - The workflow is still defined by a formal, predefined model.
  - Roles are well-defined.
  - Coordination is achieved through formal, structured interaction protocols.

LLM MAS vs Classic WfMS / Agent-Based Workflows

- Agentic AI Frameworks act on *emergent goal achievement* – from a high-level objective to autonomously construct the process to reach it. Autonomy is *generative & emergent*.
- Architecture is "*scaffolding*" around LLM. Agent discovers and executes the process on the fly; workflow is *emergent & dynamic.*
- Execution & Control Flow: *emergent, probabilistic, dynamic*.
- State, Data & Resource Management are handled *implicitly*.
  - State is maintained in the LLM's context window
  - Data = agent's "holistic understanding"‟: prompts, tool, memory
  - Resources are "tools" that the agent dynamically selects and agents involved in collaboration.
- Primary Application is for *handling complex and novel problems* by providing *generative flexibility*.

WORKFLOW PATTERNS

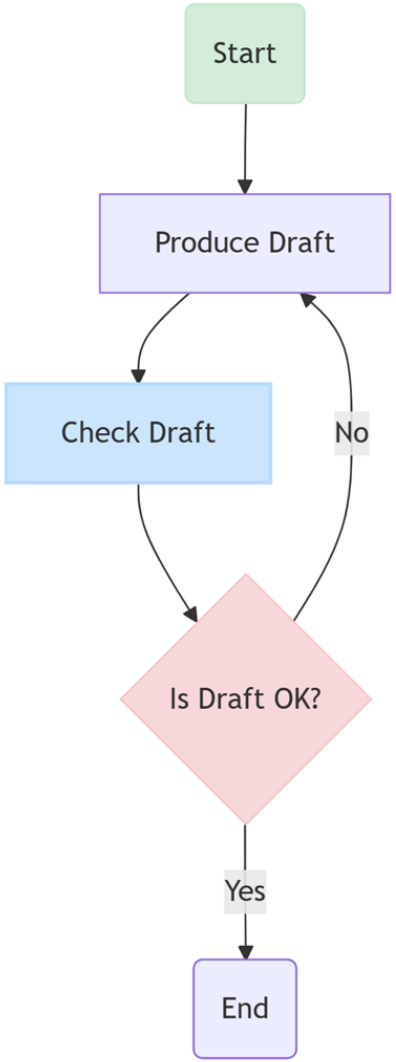
- Workflow patterns = reusable, proven solutions to common, recurring problems of designing & modeling workflows.
- Workflow Patterns Initiative started in 1999 and led by *Wil van der Aalst & Arthur ter Hofstede*.
- They identified:
  - Control-Flow Patterns describe the execution order of tasks of a workflow.
  - Data Patterns describe how data is created, handled, passed between workflow tasks.
  - Resource Patterns define how work is allocated to resources (people & systems).



@ MIT Press, 2016

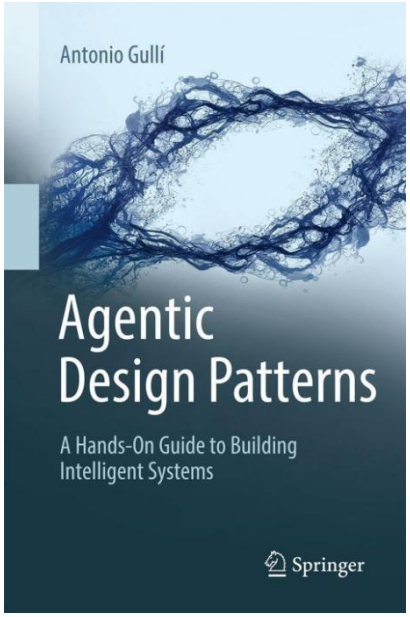
Structured Loop WCP

- Example
  - Iterative process of creation and review. Loop continues until the "Check Draft" task results in an "OK" status.
  - **Similar to agentic Reflection pattern**



AGENTIC DESIGN PATTERNS

- Agentic design patterns = reusable solutions for the recurring problems of creating Agentic AI systems.
  - Antonio Gulli book: **21** ADPs
  - Yue Liu et al.: **18** architectural patterns
  - Khan on Medium: **17** Agentic AI Patterns for Large Scale AI.



Building 17 Agentic AI Patterns and Their Role in Large-Scale AI Systems

Ensembling, Meta-Control, ToT, Reflexive, PEV and more

Fareed Khan · Follow · 77 min read · Sep 26, 2025

2.1K · 45

The Journal of Systems and Software 220 (2025) 112278

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

Agent design pattern catalogue: A collection of architectural patterns for foundation model based agents☆

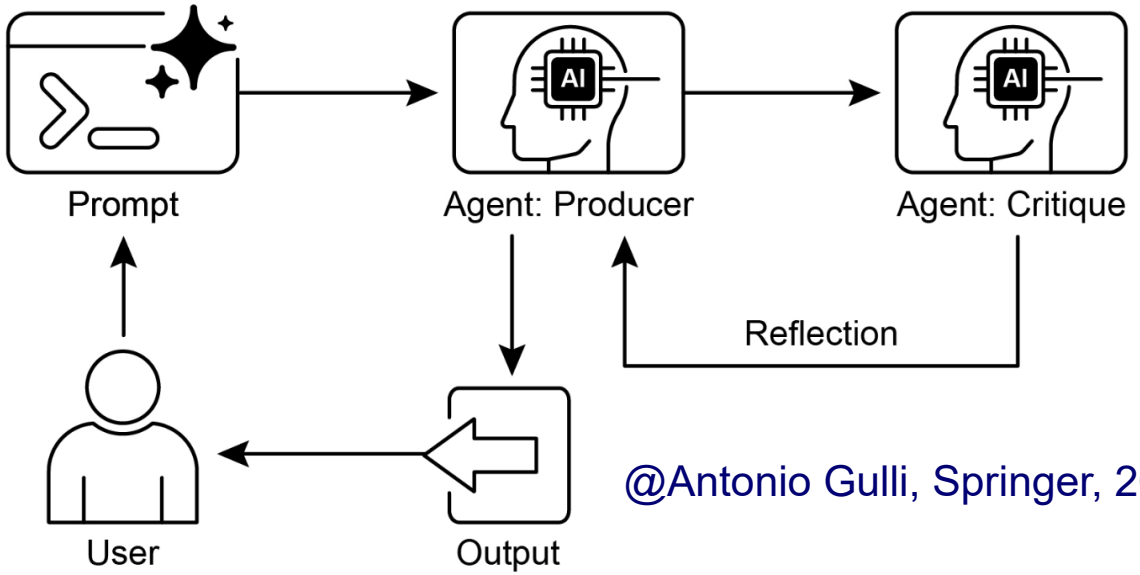
Yue Liu *, Sin Kit Lo, Qinghua Lu, Liming Zhu, Dehai Zhao, Xiwei Xu, Stefan Harrer, Jon Whittle

Data61, CSIRO, Sydney, Australia

(Cross-)Reflection ADP

1. Producer agent creates an Output for a given Task provided by User.
2. Output is sent to Critique asking for evaluation and feedback.
3. Critique agent evaluates Output producing a Review.
4. If Review is **not OK**, then Review is sent to Producer, Producer updates Output based on Review and process continues from step 2.
5. If Review is **OK**, then Output is returned to User and process stops.

Structured Loop WCP !



@Antonio Gulli, Springer, 2025

SUMMARY: CLASSIC MAS VS. LLM-AGENTS

Characteristic	Classic MAS	LLM-based Agents
Foundation	Symbolic AI, Logic, Game Theory	Sub-symbolic & Generative AI
Key Architecture	Belief-Desire-Intention (BDI)	Profile, Memory, Planning, Action
Environment	Agents & Artifacts (A&A), Standardized APIs	Tool Utilization, Model Context Protocol (MCP)
Communication	Formal ACLs (FIPA-ACL)	LLM-mediated Natural Language Dialogue
Coordination	Explicit Interaction Protocols	Emergent from Context-Aware Dialogue
Cooperation	Distributed Problem Solving (e.g., Contract-Net)	Driven by Shared Intent in a System Prompt
Negotiation	Game Theory, Utility Functions	Persuasive Rhetoric, Creative Problem-Solving
Norms & Social Laws	Formal Logic, Explicitly Enforced Rules	Implicit in Prompts, Enforced by Self-Correction

Challenges & Future Directions

Key Challenges

Lack of Standardization
Hinders interoperability and reproducibility.

Hallucinations & Reliability
Probabilistic nature poses risks for critical applications.

Privacy & Security
Unconstrained natural language creates new vulnerabilities.

Cost & Latency
LLM inference is computationally expensive.

Explainability
"Black box" nature makes debugging and auditing difficult.

Future Research Directions

Bridging LLMs and Game Theory
Designing rational agents for formal strategic interactions.

Software Engineering for Agent Systems
Creating robust development environments, APIs, and best practices.

Real-World Applications
Evaluating agent technologies on practical, real-world problems.


Thank You

