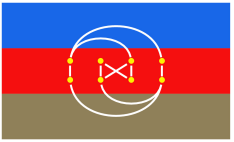
OiCOS
agents.oic.os



OiC.OS — The Operating System for Economic Organizations

Eduard Bretthauer · Viktor Winschel

June 2026

https://oicos.systems

Contents

Not a theory —
but a running system
that employees program themselves.

The Principle

1. This is our offer

We deliver a visual programming language with which organizations can be assembled from agents, like building with Lego. The result IS the ERP system itself.

- **Software is compositional (module systems exist) — management is not.** The software industry has languages, compilers and modular building blocks — exactly that is still missing in economic management and accounting.
- We are the first to bring this principle — *compositionality* — into economics, management systems, accounting and thus into the economy itself: complex organizations emerge from small, reusable parts.
- The model — i.e. the software — follows from the theory or specification **automatically and algorithmically**: a compiler does the translation and computes the program — no months-long customization cycles and feedback loops between management ↔ customizer ↔ programmer.

2. This is our universal solution

One principle, for every scale.

Whether cost centers, a department, the whole company, the holding, the supply chain or entire currency areas — all are assembled from reusable agent building blocks; only the scale changes, not the principle.

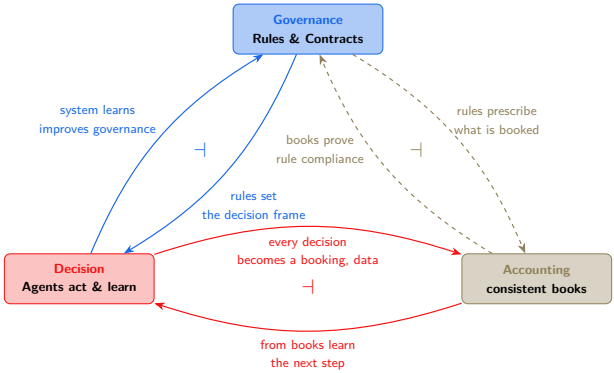
Three perspectives on the system:

- **Governance** — rules and contracts
- **Decision** — agents decide and learn
- **Accounting** — consistent books, data

What this enables:

- Learning: decision ↔ data
- System learns: governance ↔ decision
- Restructuring
- Scaling

Consistency of the three perspectives



Each pair of areas is **tightly coupled** (symbol ⊣): what one prescribes, the other reflects back *as a consistency condition in both directions.* **This is exactly why rules, decisions and books always stay in agreement** — deviations are measured, not discovered only after the fact.

Compliant by Decision, not only by ex-post audit.

3. How does this solve your problems?

Thanks to compositionality, processes and even restructuring can be simulated as scenarios before implementation. The result is a running, verified digital twin of your organization that grows with it.

- **Compute scenarios:** make-or-buy, in-/outsourcing, M&A, supply chain, holding structure — with the internal and external booking consequences of each variant.
- **Understand profitability:** where profit arises, transfer prices, contribution margins, break-even.
- **No rigid PowerPoint, no static Excel on a drive,** but a digital twin that keeps running, learns and can be reorganized.

Technology

Which agents are there?

Two kinds of artificial intelligence — we complement the current hype:

A. Statistical AI

- language models (LLM)
- strong at recognizing, phrasing, suggesting
- but: no goal, no consistency, no books

B. Logical (algebraic) AI — OiC.OS agents

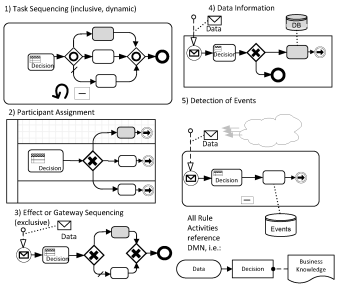
- economically specialized **agents**
- can be composed (compositional)
- can do accounting, cost accounting and financial bookkeeping

The LLM talks —
OiC.OS agents compute, book and organize.
They represent knowledge.

Example: where today's process standards stop

BPMN (Business Process Model Notation)

DMN (Decision Model Notation)



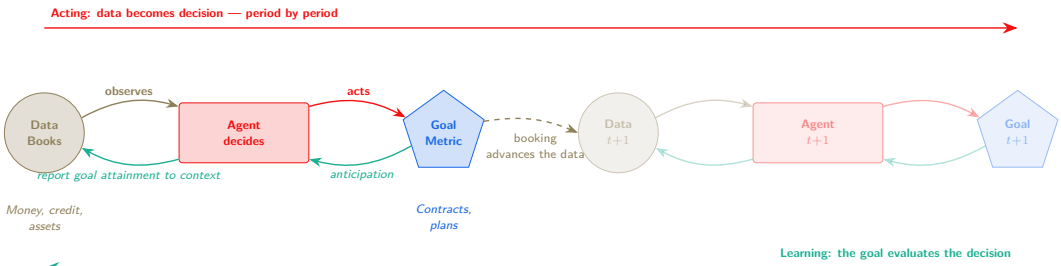
What this standard lacks — and what OiC.OS can do:

- **Goals** — why is something decided, and how?
- **Compositionality** — compose building blocks
- **Hierarchy** — compose levels
- **Contracts** — rules, smart (algorithmic) contracts
- **Learning** — agents improve their decisions
- **Units, real and monetary** — units of account (cost centers)
- **Consistent books** — consistency across several double-entry bookkeeping systems (e.g. for debt contracts, aka quadruple accounting)
- **Provability** — OiC.OS software is **provably** correct with respect to the (visual) specification

What do the explicit goals of the OiC.OS agents enable?

- **Learning becomes possible only through goals:** learning comes from the deviation between goal and actual outcome.
- **Automated adaptation:** if sub-goals change in a large network plan, the decision nodes adapt automatically — in BPMN/DMN this requires manual intervention by consultants or employees.
- **The whole system learns:** only our compositional technology makes this learning add up — so that not just individual building blocks improve, but the organization as a whole learns.

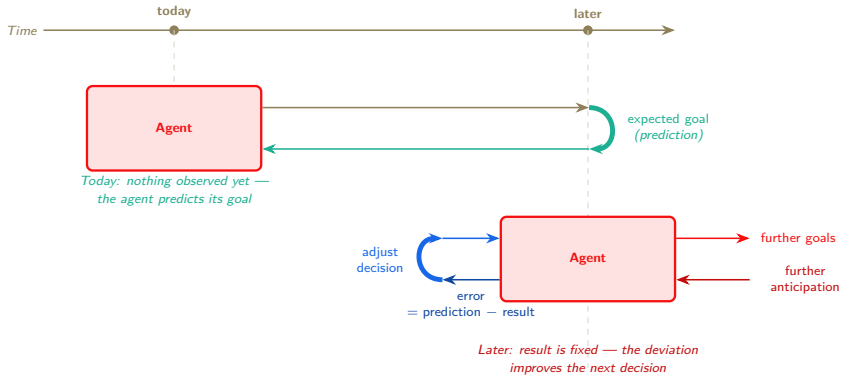
What is an OiC.OS agent? — The temporal composition



An agent observes the data in its context, **decides** with respect to its **goal** and reflects on how well that decision contributed to the goal — the backward signal is what makes agents compositional and lets organizations learn.

Temporal composition: the **booking** advances the books — the result of one period becomes the starting state of the next. In this way **the same agent, repeated over time,** links up into a continuous chain.

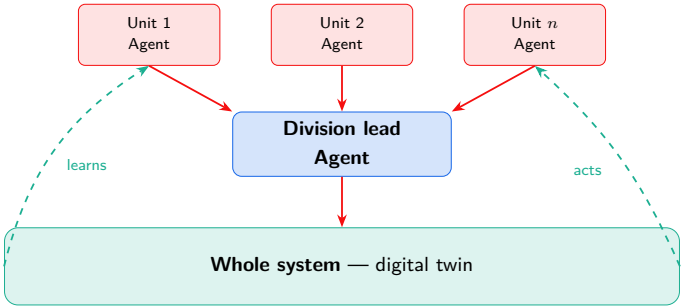
How an agent learns: expectation meets result



Today. The agent commits to an **expected goal** — its prediction, before the result is known.

Later. The **actual goal** is in. The **deviation** (result minus prediction) flows back and **adjusts the next decision** — thus the agent learns, period by period.

What does compositional mean? Why goals?



- **Compositional**: decompose complex processes into reusable building blocks — and reassemble them.
- Several agents combine into another agent — a system at the next level.
- **Why goals?** Without a goal definition, no learning. Without hierarchical compositionality, no system learning.
- Universality: cost center → department → company → holding → supply chain → currency area.

We apply the architectural method of computer science to economics.

Every concept from your organization has an exact counterpart in software architecture — the same proven design principle, just applied to the economy:

Economics	Computer Science
Organization — rules, decisions, books	Runtime environment, logic in a *topos* with *internal type theory*
Consolidation: local books into the group balance sheet	Distributed data into *one* consistent state
Plan forward, learn backward from actuals	Lossless back and forth *(compiler ↔ decompiler)*
Agent: act *and* evaluate against the goal	Bidirectional program — same form as backpropagation
Contracts: rights and obligations	Machine-checked assertions *(propositions-as-types)*
One period: read → decide → book	Unfold-then-fold without intermediate storage *(hylomorphism)*
Same function on every level	Parametric building block *(polymorphic function)*
Closed money circuit	Feedback as a fixed point *(trace / fix)*

Standard in software for decades, now in OiC.OS for the first time in the management of organizations — specification and software become one, and the compiler does the translation.

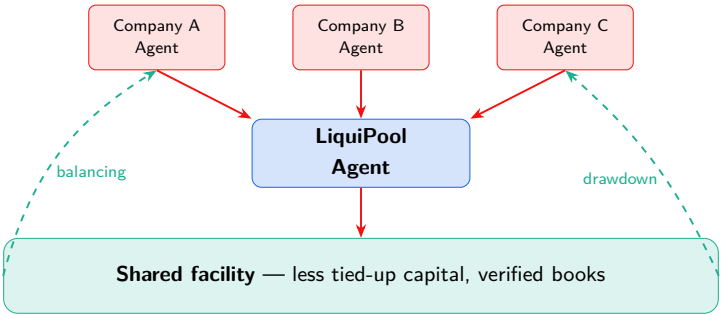
Current Project

A simple application to develop the complex system.

So far it exists only in an academic monetary-theory model (5 agents: sectors, banks, central bank).

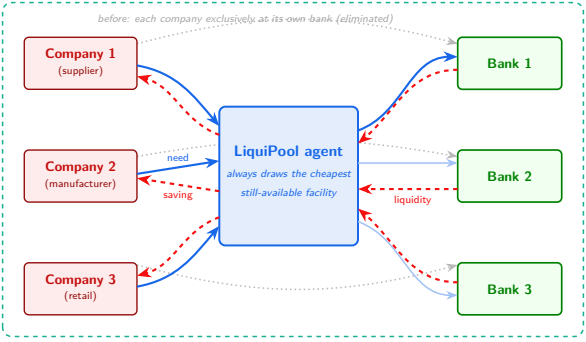
- **Easy entry:** several companies share a common liquidity facility instead of three separate ones.
- **Develop the complex system:** the same mechanism later carries cost accounting and holding control, all the way up to banking logic.
- **Usable by domain experts:** operable without programming (low-code / visual modeling).
- **Training on a simple example:** employees learn the system on a manageable, immediately useful case.

LiquiPool: three companies, one pool — one agent



Each company reports its **liquidity need**; the LiquiPool agent allocates by priority and balances out. Three separate facilities cost more than one shared one — **the composition creates the value** ($3 \times 1 \neq 1 \times 3$). The same building block is later a banking consortium and central bank — just larger.

LiquiPool in detail: the pool manages all bank relationships



Before (gray): each company depends *exclusively* on its own bank — three separate lines, three credit limits. **LiquiPool:** all companies report their **need** to the **LiquiPool agent** (blue). It does **not draw on all banks at once**, but always taps the **cheapest still-available facility first** (merit order: first bank 1, then 2, then 3 — only as much as needed) and distributes the **liquidity** (red); the savings are shared fairly (Shapley). **Result:** less tied-up capital, one shared pool instead of three islands — *the composition creates the value* ($3 \times 1 \neq 1 \times 3$). **Moreover:** offsetting invoices within the consortium are netted *before* settlement — only the balance is paid. **And:** surplus cash is lent directly across companies — *without a bank*.

The next step

- **Quickly visible benefit** via LiquiPool, then expansion to cost accounting, planning and restructuring.
- **Joint co-development** on your concrete use case — e.g. model your cost accounting in OiC.OS and ship it automatically as software.
- **You keep a learning, running digital twin** of your organization — not a deck of consulting slides, but an asset.

OiC.OS — organizations from building blocks. The system learns.