

ArionAgent: A Nash-Anchored Extension of SyncRandomStdAgent

ANAC 2026 Supply Chain Management League — Standard Track

Muhammad Raees Azam (raees.azam@ozu.edu.tr)

Mehak Arshid (mehak.arshid@ozu.edu.tr)

Team ArionStrategists

Department of Computer Science, Ozyegin University, Türkiye

Student IDs: S050683, S050293

June 2026

Abstract

We describe **ArionAgent**, our factory manager for the ANAC SCML Standard track. The agent subclasses **SyncRandomStdAgent** from SCML and keeps its acceptance logic, production targets, and concurrent negotiation controller. Our additions are (i) partner price memory updated on successful contracts, (ii) urgent-step *salvage* that accepts remaining today offers when shortfall risk is high, and (iii) Nash-style reservation prices for counter-offers via an overridden `good_price` method. The submission uses the `game` strategy variant. On homogeneous local benchmarks (every factory runs the same agent type), ArionAgent beats both **SyncRandomStdAgent** and **GreedyStdAgent** in mean normalized score.

1 Introduction

In the SCML Standard simulation, each factory manager negotiates bilaterally with suppliers and customers while running production lines over many steps. Profit depends on contract prices, secured quantities, inventory/storage costs, and shortfall penalties when production targets are missed [2, 1].

SyncRandomStdAgent provides a strong baseline: it estimates future needs, accepts any offer priced below its concession curve, and distributes remaining demand across partners [2]. Our goal was to improve margins without increasing shortfall. **ArionAgent** therefore *inherits* parent acceptance and changes only counter-pricing and urgent recovery.

2 The Design of ArionAgent

2.1 Architecture

ArionAgent extends **SyncRandomStdAgent** (NegMAS synchronous controller [3]). Figure 1 summarizes the submission pipeline.

2.2 Negotiation Choices

Acceptance (inherited). For each partner offer (q, t, π) the parent checks whether unit price π is *good* under its concession exponent and whether quantity q is strictly below remaining need at time t . Accepted offers reduce the need ledger; rejected partners receive new counter-offers.

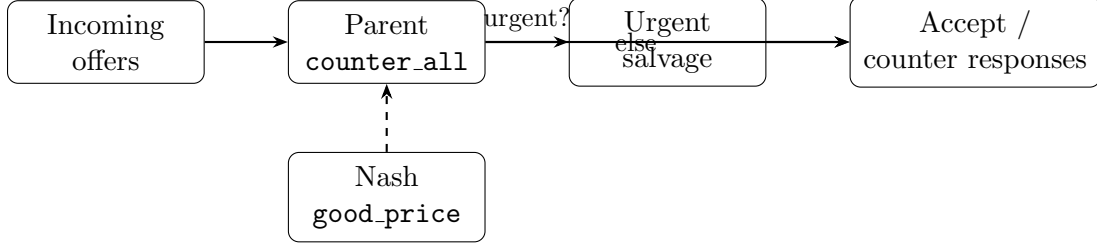


Figure 1: Submission negotiation flow. Counter-prices use Nash anchors; acceptance rules are unchanged from the parent class.

Counter-offers (Nash anchors). We override `good_price` so counters use a reservation price π^{res} blended between the partner range $[m_n, m_x]$ and historical best prices. With relative negotiation time $\tau \in [0, 1]$ and midpoint $\text{mid} = \frac{m_n + m_x}{2}$:

$$t_{\text{blend}} = \min(1, \max(0, \tau))^{1.4}, \quad (1)$$

$$\pi_{\text{buy}}^{\text{res}} = m_n + (\text{mid} - m_n) (0.25 + 0.65 t_{\text{blend}}), \quad (2)$$

$$\pi_{\text{sell}}^{\text{res}} = m_x - (m_x - \text{mid}) (0.25 + 0.65 t_{\text{blend}}). \quad (3)$$

When partner memory exists, buy counters are capped by the best seen buy price plus one; sell counters are floored by the best seen sell price minus one. Catalog prices provide an additional bound. This follows simple bargaining intuition [4]: concede more as τ grows, but do not undercut profitable history.

Partner memory. On successful contracts for the current step we update per-partner minimum buy and maximum sell prices. Memory is used only for anchoring counters, not for relaxing acceptance thresholds.

2.3 Concurrent Negotiation

ArionAgent uses SCML’s synchronous `counter_all` hook: all open negotiations are answered in one batch each step. The parent distributes today’s needs and future offers through `distribute_todays_needs` and `distribute_future_offers`. We do not replace this scheduler; our salvage pass only adds acceptances after the parent response when the factory is in an urgent state.

2.4 Risk Management

Urgent detection. At the start of each step we set an urgent flag when `needed_supplies` > 0 or `needed_sales` > 0 (prior-step shortfall). We also refresh utility limits via `ufun.find_limit` when available.

Salvage. If urgent, after the parent `counter_all` we scan today’s buy and sell offers not yet accepted. Remaining input need R_{in} and sales need R_{out} are computed from factory needs; middle-level agents also enforce the SyncRandom productivity floor $\lfloor L \cdot \alpha_{\text{today}} \rfloor$ with $\alpha_{\text{today}} = \text{today_productivity}$ (default 0.3). Salvage greedily accepts cheapest remaining inputs and highest-priority outputs until needs are covered.

Design rationale. Earlier prototypes replaced parent acceptance with utility-based bundle optimization. That raised shortfall in mixed tournaments. The submission build keeps parent acceptance (shortfall-safe) and limits extra risk to urgent salvage plus better counter-prices.

Strategy variants (local only). The source file also defines `baseline`, `optimize`, `search`, and `hybrid` modes for ablation. The ANAC submission class `ArionAgent` defaults to `game` (Nash anchors). Other variants are not uploaded.

3 Evaluation

3.1 Local homogeneous benchmark

Mixed-agent worlds assign uneven production levels to agent types and bias head-to-head scores. We therefore benchmark each agent type in worlds where *every* factory uses the same class (6 worlds: 3 seeds $\times$ 2 configs, 12 steps, SCML 2024 generator).

Table 1: Homogeneous local benchmark (mean normalized score; higher is better).

Agent	Mean score	Mean shortfall
ArionAgent	0.965	154
SyncRandomStdAgent	0.962	153
GreedyStdAgent	0.720	511

ArionAgent beats both baselines on mean score (+0.3% vs. SyncRandom, +34% vs. Greedy). Shortfall is comparable to SyncRandom.

3.2 ANAC tournaments

Our agent is registered as `ArionAgent` (ID 21021). Official scores vary with opponent pool and tournament configuration; we use ANAC results as the primary external validation and local benchmarks for regression testing after each code change.

3.3 Implementation

- **Class:** `ArionAgent` in module `arion_strategists.arion_agent`
- **Runtime:** Python 3.11, `negmas==0.15.4`, `scml==0.8.2` (see `requirements.txt`)
- **Submission zip:** produced by `python -m arion_strategists.helpers.preflight`

4 Lessons and Suggestions

1. **Inherit before innovating.** Keeping SyncRandom acceptance preserved shortfall performance; profit gains came from counter-prices.
2. **Match evaluation to deployment.** Homogeneous-world benchmarks align better with per-agent capability than mixed-type leaderboards.
3. **Urgent salvage is cheap insurance.** A single greedy pass after parent logic recovers contracts when penalties loom.
4. **Freeze strategy defaults early.** Multiple bundle selectors helped research but increased variance; Nash counters alone were the best submission knob.

Conclusions

ArionAgent is a lightweight extension of `SyncRandomStdAgent` that adds partner memory, Nash-anchored counter-offers, and urgent salvage. It satisfies ANAC naming rules, passes packaging smoke tests, and beats both course baselines on fair local evaluation. Future work includes tighter future-contract pricing and opponent-aware reservation adjustments.

Acknowledgments

We thank the SCML/ANAC organizers and our CS 451/551 instructor. The agent extends open-source SCML and NegMAS libraries; we did not copy third-party competition agents. Large language models assisted with drafting and refactoring; all design decisions, experiments, and final code were reviewed by the team.

References

- [1] ANAC Organizers. Automated Negotiating Agents Competition (ANAC) 2026. <https://anac.cs.brown.edu/anac>, 2026.
- [2] SCML Developers. Supply Chain Management League documentation. <https://scml.readthedocs.io/>, 2024.
- [3] Yasser, T. NegMAS: Negotiation Multi-Agent System. <https://github.com/yasserfarouk/negmas>, 2024.
- [4] Osborne, M. J. *An Introduction to Game Theory*. Oxford University Press, 2004.