EDITOR: Schahram Dustdar, dustdar@dsg.tuwien.ac.at

DEPARTMENT: INTERNET OF THINGS, PEOPLE, AND PROCESSES

AICon: Agentic Intelligence for the Computing Continuum

Ildefons Magrans de Abril, Nefeli-Marina Rouska, and Victor Casamayor, *Pompeu Fabra University, 08018,* Barcelona, Spain

Schahram Dustdar, *Pompeu Fabra University, 08018, Barcelona, Spain, and TU Wien, 1040, Vienna, Austria*

Computing continuum systems combine Internet of Things devices, edge, fog, and cloud resources to reduce latency, cost, and energy consumption in highly heterogeneous environments. However, existing simulators that rely on centralized architectures have tightly coupled components, fragile artificial intelligence/machine learning integrations, steep learning curves, and oversimplified agent models. These limitations make it difficult to develop realistic decentralized and resource-aware management approaches. We introduce AICon, an open source Python framework built on YAFS (Yet Another Fog Simulator) that overcomes these issues through a clean, modular three-layer design: infrastructure, application, and a novel management layer composed of autonomous agents. These agents are treated as first-class simulated workloads with realistic features: configurable work–sleep cycles, partial observability, and quality-of-service-aware actions (e.g., scaling application intensity or tuning node performance). AICon significantly lowers the barrier to entry, improves code reuse, and accelerates prototyping and validation of adaptive, decentralized management strategies that optimize service-level objectives under real-world constraints.

To reduce latency, cost, and energy consumption, cloud-based services are increasingly deployed closer to users. This trend is enabled by a spectrum of devices with diverse tradeoffs in computation, storage, power, and mobility, giving rise to highly heterogeneous infrastructure deployments collectively known as *computing continuum systems* (*CCSs*).[1] CCSs typically integrate the Internet of Things (IoT), edge, fog, and cloud components, resulting in a high-dimensional, multiobjective challenge for instantiating and managing distributed applications.

Simulators, variously labeled as fog, edge, or continuum computing tools, have emerged as essential instruments for evaluating initial placement strategies and online management policies.[2] By enabling cost-effective experimentation across diverse deployments, these tools allow researchers to validate solution robustness far beyond the scope of small-scale, expensive real-world pilots. As an alternative, *emulators* offer higher fidelity at the cost of greater computational overhead, bridging the gap between simulation and physical deployment by replicating real hardware and network behaviors with near-real accuracy.[3]

A foundational concept in this domain is the MAPE-K control loop (monitor–analyze–plan–execute over a shared knowledge base), introduced by IBM in the early 2000s as part of the Autonomic Computing Initiative.[4] Over the past decade, MAPE-K-inspired feedback loops have been integrated into simulators such as iFogSim2,[5] Yet Another Fog Simulator (YAFS),[6] and CloudSim[7] to prototype self-adaptive behaviors, train learning-based controllers (e.g., reinforcement

Digital Object Identifier 10.1109/MIC.2025.3645964
Date of current version 20 February 2026.

learning or active inference), and study emergent properties of autonomic systems.

Despite these advances, current fog and edge computing simulators face significant challenges in supporting the development of realistic, decentralized online management systems. The key limitations include

- *Centralized control assumptions*: Most MAPEK integrations[8,9,10] assume a single, fully observable controller with unrestricted actuation. This oversimplification neglects the critical aspects of distributed autonomy—partial observability, constrained intervention, and coordination among heterogeneous, privacy-constrained stakeholders—prevalent in real CCS environments.
- *Convolved architectures without clear separation of core autonomic entities*: Existing simulators entangle *agents* (decision makers), *metrics* (observables), and *interventions* (actuators) in monolithic or tightly coupled designs, preventing modular reuse, composability of decentralized strategies (e.g., hierarchical, peer to peer, or stigmergic/environment mediated), and straightforward transfer of simulated policies to real deployments.[11,12]
- *Complex integration of artificial intelligence/machine learning (AI/ML)-based controllers*: Java-based frameworks require complex and fragile interlanguage bridges or full rewrites to incorporate Python-centric AI/ML libraries, entangling simulation logic with foreign runtime concerns and undermining clean separation of concerns.[13]
- *High learning curves, which limit interdisciplinary participation*: Monolithic, nonmodular architectures—coupled with an implicit assumption of advanced programming proficiency—create a significant barrier for domain experts without deep software engineering skills, hindering conceptual and interdisciplinary progress.[14,15]
- *Failure to model the dual nature of autonomic agents*: Current simulators treat management agents as external, zero-cost oracles with instantaneous decision making, neglecting their role as *simulated workloads* that consume node resources (CPU, memory, or network) and introduce perception–action delays. This omission produces unrealistic control behavior and overlooks self-induced load and resource contention in real deployments.[16]
- *Insufficient support for application-level adaptability*: Most CC simulators model applications as fixed instruction counts with constant resource demand, providing no mechanism to adjust application-level parameters (e.g., image resolution, frame rate, inference precision, or early-exit branches) and observe their effect on output quality or user-perceived quality of service (QoS). As a result, they cannot evaluate real-world adaptation techniques that trade accuracy or fidelity for reduced latency and resource usage—today a core strategy in edge AI deployments.[17]

In this article, we present AICon, an open source simulator-based framework for developing distributed management agents in the computing continuum (CC), directly addressing the aforementioned limitations. Available at https://github.com/ildefons/aicon, AICon features a modular, agent-first Python architecture with native support for partial observability, resource-aware control loops, and composable multiagent coordination—enabling rapid prototyping and realistic simulation of management overhead. By reducing development and learning barriers, AICon empowers both newcomers and experienced teams to explore more complex, diverse, and realistic validation scenarios, accelerating progress in the CC field.

AICon draws inspiration from foundational platforms that democratized research fields: OpenAI Gym for single-agent reinforcement learning, and PettingZoo for multiagent settings. Similarly, AICon aims to become the standard environment for *agentic intelligence in the CC*, shifting focus from *whether* decentralized management is possible to *how* to optimally compose heterogeneous agents under real-world constraints. Its design naturally supports emerging paradigms such as the free energy principle and active inference, where agents can be implemented as inference loops that minimize surprise over observable metrics while respecting resource and intervention budgets.[18,19]

This article is organized as follows. The "Usage Model" section introduces the AICon usage model, describing its three-layer structure and how it enables modeling of decentralized management systems in the CC. The "Architecture" section details the system architecture, explaining the main classes and their interactions within the simulator. Finally, the "Conclusion and Future Work" section summarizes the contributions of AICon and outlines future research directions.

USAGE MODEL

The AICon usage model (Figure 1) streamlines translating high-level concepts into executable simulations

via a three-layer architecture. It extends an existing CC simulator with our core contribution: the management network layer, which orchestrates the underlying infrastructure and application layers.

Infrastructure and Application Layers

Most AI/ML packages, required to implement intelligent management systems, are written in Python (see *shortcoming 3*). Therefore, to simplify the project architecture, we prioritized using an existing python-based CC simulator. YAFS,[6] currently the sole CC simulator actively maintained and written in Python, provided us with the ideal baseline.

The AICon usage model requires the initial definition of layers 1 (infrastructure) and 2 (application), following the YAFS usage model. The infrastructure layer, also known as the *topology* layer, requires the specification of four components: 1) computational nodes, 2) node to node bidirectional links, 3) centralized placing logic, and 4) message-routing logic. Each node requires three mandatory attributes: a unique integer identifier, instructions per time (IPT) unit, and memory capacity (random-access memory). Network connections are defined by two attributes: bandwidth and link propagation delay. Additionally, the user must specify two centralized controllers: 1) a placement logic class with two methods to manage the initial and runtime placement of application module instances and 2) a routing logic class with a method that determines how to route a message from the current node to the next to reach its target application module. For further details, refer to the original documentation of the baseline simulator.[6]

The application layer implements the distributed application running atop layer 1. Applications are described in the application layer using a dataflow model that consists of 1) application modules further divided as sources, computation, and sink modules, and 2) a set of message request types exchanged between pairs of application modules. Each message request is meant to model a particular service call and is parameterized with the number of instructions required to compute that request.

Sources generate requests according to some user-defined pattern; computation modules receive requests, simulate the corresponding resource consumption, build a response message, and send it; finally, sink modules are basically like computation modules without the need to generate a response message. Each computation and sink module is instantiated for

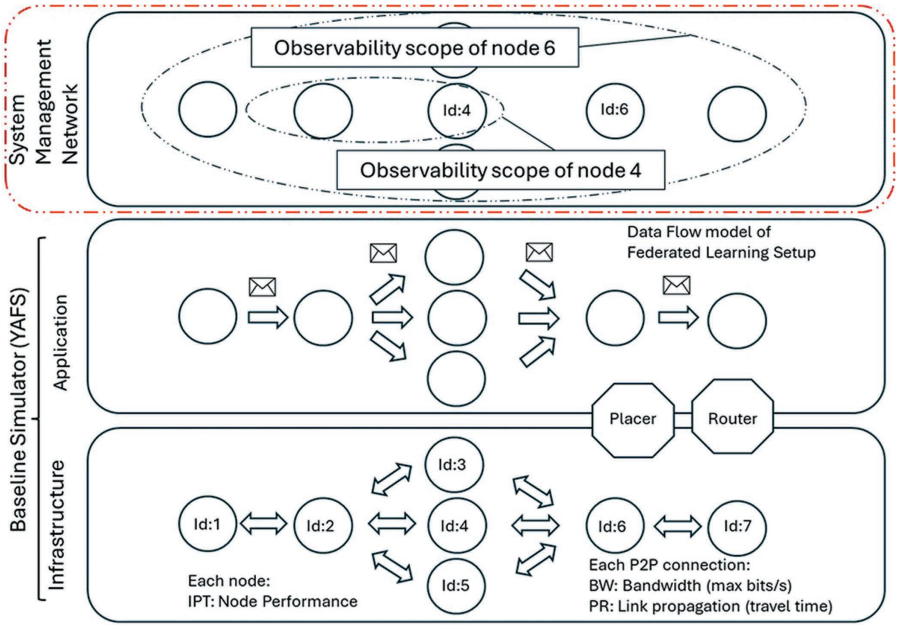

FIGURE 1. Three-layer usage model. The two first layers, infrastructure and application, are facilitated by the baseline simulator (YAFS). Our contribution, the management network layer, orchestrates both to enable cross-layer optimization. P2P: peer to peer.

at least in one node of the infrastructure layer, and it remains active until the simulation ends or the instance is released.

Figure 2 shows an application module instantiated in a computational node. Each instance maintains an infinite-length input queue and operates iteratively throughout the simulation. In each iteration, the instance retrieves the top message from its queue, triggers a "time advancement event" Δt in the discrete-event simulator's central clock, and generates an output message that is transmitted through the simulator's network. Crucially, Δt is computed as a function of two parameters: the node's IPT and the instructions required to compute the message request.

We have extended the computation of Δt with an additional parameter to modulate the QoS of a message request as a function of the percentage of nominal instructions. Listing1 shows how we can extend the declaration of an application message with QoS modulator class. This lightweight extension to the baseline YAFS simulator introduces a standardized intervention point that allows agents to dynamically adjust the QoS delivered to the user in exchange for reduced computation time or energy.

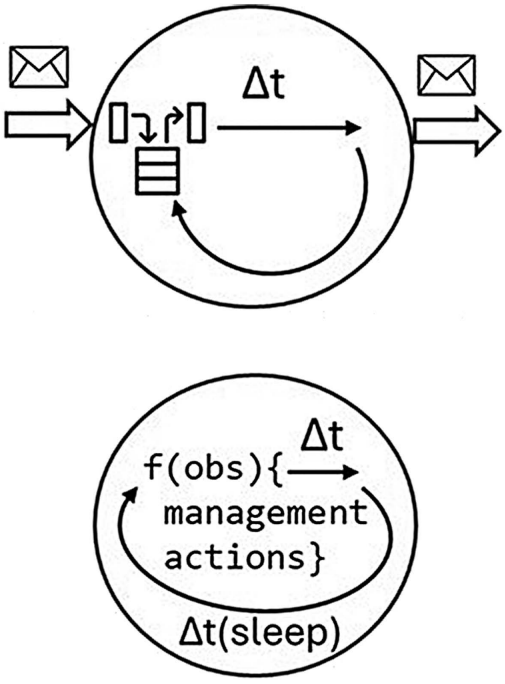

FIGURE 2. (a) Application module instance, which features an input queue of infinite length and iterates until the end of the overall simulation. In each iteration, it retrieves the top message from its queue and issues a time advancement event to the common clock managed by the core discrete-event simulator. Finally, it generates an output message and pushes it to the simulator network. (b) Agent instance. In each iteration, the agent wakes up, execute a custom function [f(obs)], where obs is a set of metrics captured since the agent waked up the previous cycle, then it issues a second time advancement event corresponding to the simulated time required to execute the logic of f. Finally, the agent goes to sleep until the next work–sleep cycle starts. Note that agent instances don't use messages or queues.

Listing 1: Declaration of an application message with a QoS modulator.

```
m1 = Message("M1", "Camera",
"ImageRecognition," instructions=10**6,
qos=LinearQoS(L=0.05,R=1))
```

The mechanism directly overcomes *shortcoming 6* by enabling application-level adaptations and making their impact on user-perceived quality explicitly observable. Table 1 summarizes the modulator classes currently implemented (linear, exponential with saturation, and exponential with saturation and diminishing results).

Network Management Layer

Our primary contribution is the Network Management layer—which addresses most of the shortcomings of existing fog and edge computing simulators in supporting the development of online management systems. Listing 2 presents an example JavaScript Object Notation (JSON) object defining the configuration of the network management layer. The layer models the

TABLE 1. Overview of classes relating the percentage of nominal message instructions and the QoS percentage.

QoS class name	Brief description of the function between the percentage of nominal message instructions x and the QoS percentage y.
LinearQoS	Clamped linear function.
SaturatingExpQoS	Saturating exponential model: $Q(x)=Q_{max}(1-\exp(-\alpha x))$.
USLQoS	Universal scalability law (USL) QoS model with normalized input: $Q(x)=Q_{max}[N/(1+\text{alpha}(N-1)+\text{beta}N(N-1))]$.

management network as a collection of agents, each defined by the following fields:

- *node_id*: The infrastructure node on which the agent is instantiated (e.g., line x or 2).
- *agent_type*: The class of the management agent.
- *sleep_time*: Duration of the agent's sleep phase (in simulated time units).
- *instructions_per_wakeup*: Computational cost of the agent's work phase (in simulated instructions).
- *agent_ipt_percentage*: When an agent is co-located with an application module instance, this field specifies the percentage of node resources reserved for the agent. It is used to derive execution times for both the agent work phase and the co-located application module.
- *observable_node_ids*: List of nodes that the agent may observe and potentially intervene on. This list abstracts data jurisdiction and private-network constraints, enabling agents with differing observation spaces
- *Metrics*: Performance indicators to monitor.
- *Actions*: Interventions available to the agent.

Reusing predefined agent, metric, and intervention classes simplifies the configuration and reduces the learning curve for AICon, directly addressing *shortcoming 4*.

Management Agent

Similar to an application module instance, an agent instance operates iteratively throughout the simulation duration, as illustrated in Figure 2. However, unlike the application module instance, the agent exhibits the following distinguishing characteristics:

- *Customizable perception–action logic*: Actual agent behavior is defined within a customizable method, denoted as f(obs). Users may specify custom perception–action logic by leveraging the metrics collected since the start of the previous working phase (see Figure 3) from the infrastructure nodes listed in observable_node_ids and use them to apply interventions specified in the agent's configuration field actions.
- *Dual-phase temporal dynamics*: An agent instance runs with alternating work and sleep phases, each corresponding to a simulated time-advancement event. The number of time units of the work–phase event Δt is computed using the agent's configuration field instructions_per_wakeup, and the node IPT, where the agent is instantiated.

Listing 2: Example JSON configuration for the simulator, illustrating agent settings, metrics, and actions.

```
[
  {
    "node_id": 0,
    "agent_type": "CloudAgent",
    "sleep_time": 500,
    "instructions_per_wakeup": 5e10,
    "agent_ipt_percentage": 0.5,
    "observable_node_ids": [0, 1],
    "metrics": {
      "service_node_utilization": { "module":
          "aic.managementnetwork", "class":
          "ServiceNodeUtilization",
        "post": {
          "module":
            "aic.managementnetwork",
          "class": "PostDiscretize",
          "params": {"bins": [0, 20, 40, 60, 80, 100]}
        }
      },
      "linear_cost_buyya": { "module":
          "aic.managementnetwork",
        "class": "LinearCostBuyya",
        "params": {"cost_alpha": 1} }
    },
    "actions": {
      "msg_instructions_pctl": { "module":
          "aic.managementnetwork",
        "class": "PCTMsgInstructs",
        "params": {"pctls": [0.1, 0.3, 0.5, 0.7, 1]}
      }
    }
  },
  {
    "node_id": 1,
    "agent_type": "SensorAgent",
    "sleep_time": 500,
    "instructions_per_wakeup": 1e8,
    "agent_ipt_percentage": 0.5,
    "observable_node_ids":[1, 2]
    "metrics": {
      "service_node_utilization": { "module":
          "aic.management_network", "class":
          "ServiceNodeUtilization"
      }
    }
  }
]
```

The number of time units of the sleep-phase event Δt_{sleep} is directly specified in the same configuration field. It is noteworthy that an agent has both a simulated and behavioral nature, which enables taking

into account the agent's cost and performance in the overall simulation. This feature alleviates the limited transferability to real deployments of current CC simulators, as discussed in *shortcoming 2*.

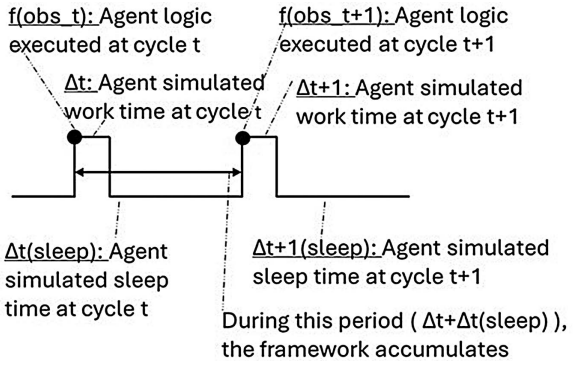

FIGURE 3. Work–sleep cycle of an agent instance.

TABLE 2. Overview of metric classes.

Metric Class Name	Brief Description
ServiceNodeUtilization	Node utilization percentage due to application module instance.
AgentNodeUtilization	Node utilization percentage due to management agent instance.
NodeAverageWaitingTime	Average waiting time for a request to start being served.
NodeRequestsWaitingIn	Number of requests waiting to be served.
NodeRequestsOut	Number of service requests serviced by node since the agent woke up last time.
NetBufferSize	Number of messages moving in the network (not yet in the input queue of the application module instances).
NodeNominalIPT	Node IPT.
LinearCostModel	Linear cost model based on utilization rate and node performance, inspired by resource management studies in fog and cloud computing.[22]
NodeDeliveredQoS	Delivered percentage of application module QoS (this changes due to a node intervention, like PCTMsgInstructs).

› *Absence of message-based communication*: Agent interaction does not rely on traditional message passing or input/output queues. Instead, communication is realized through customized metrics and interventions, which enable reading from and writing to the infrastructure, application, and management-network layers. This abstraction provides a unified and flexible mechanism for information exchange, supporting both direct agent-to-agent communication and indirect coordination mediated through shared application or infrastructure states.

Agent Metric and Intervention Spaces

The agent periodically executes its perception–action logic defined by the function f(obs), where the argument obs contains the metrics collected since the beginning of the last work phase (see Figure 3). These metrics summarize various quantities and statistical indicators of the agent's observable nodes. The set of observable nodes is specified through the configuration field observable_node_ids and formalizes the agent's *observability scope*, providing a means to implement abstractions of real-world concepts such as the data jurisdiction and privacy constraints imposed by different infrastructure stakeholders (see *shortcoming 1* for a discussion motivating this requirement).

The agent logic is ultimately responsible for performing the necessary interventions on these nodes by invoking any of the actions defined in the configuration field actions. Tables 2 and 3 list the currently available metrics and interventions, respectively. The most recent version of AICon implements two types of interventions (see the "Conclusion and Future Work" section for future planned interventions):

› *PCTMessageInstructions*: This intervention modifies the percentage of nominal instructions executed by a given application module instance. It applies to message requests declared with a QoS modulator, as illustrated in Listing 1. When an

TABLE 3. Overview of intervention classes.

Metric Class Name	Brief Description
PCTMessageInstructions	Updates the percentage of nominal instructions to be executed for a given application module instance. It has a direct effect on the QoS of the application module.
NodeIPT	Updates the node performance parameter (IPT).

agent reduces the proportion of nominal instructions required to process a message request, the corresponding application module instance operates more efficiently, thereby decreasing node utilization and operational cost. However, this gain in efficiency comes at the expense of reduced module QoS, as defined by the function specified in the message declaration.

- *NodeIPT*: This intervention adjusts the node performance parameter's IPT. Increasing IPT enhances the performance of the associated application module instance and may reduce node utilization (provided that the node is not saturated), but typically increases operational costs. Conversely, decreasing IPT reduces performance and operational costs, while typically increasing node utilization.

The metrics and interventions initially developed were chosen to allow us to develop decentralized management strategies that dynamically control the service-level objectives (SLOs) related to performance, QoS, and cost.[20,21] These strategies can act through interventions on individual node performance (trading cost for performance) and adjustments to the percentage of nominal instructions per input request (trading performance for QoS, and cost).

ARCHITECTURE

The AICon architecture comprises several new classes alongside targeted modifications to the core modules of the baseline simulator (YAFS). Figure 4 illustrates the four primary AICon components: management network, agent, metric, and intervention, along with multiple instantiations of the metric and intervention classes. The architecture also includes a modified discrete-event simulator core class and an enhanced event database capable of reading and writing simulation events at runtime.

The ManagementNetwork class implements the management network layer as a collection of agent instances. At the core of AICon is the abstract agent class, which users subclass by implementing the f(obs) method to define domain-specific perception–action logic. It provides two essential mechanisms: 1) it orchestrates explicit work–sleep cycles by issuing time-advancement requests Δt (active phase) and Δt (idle phase) to the simulator, and 2) it continuously aggregates agent-specific metrics from all observable entities using events collected since the

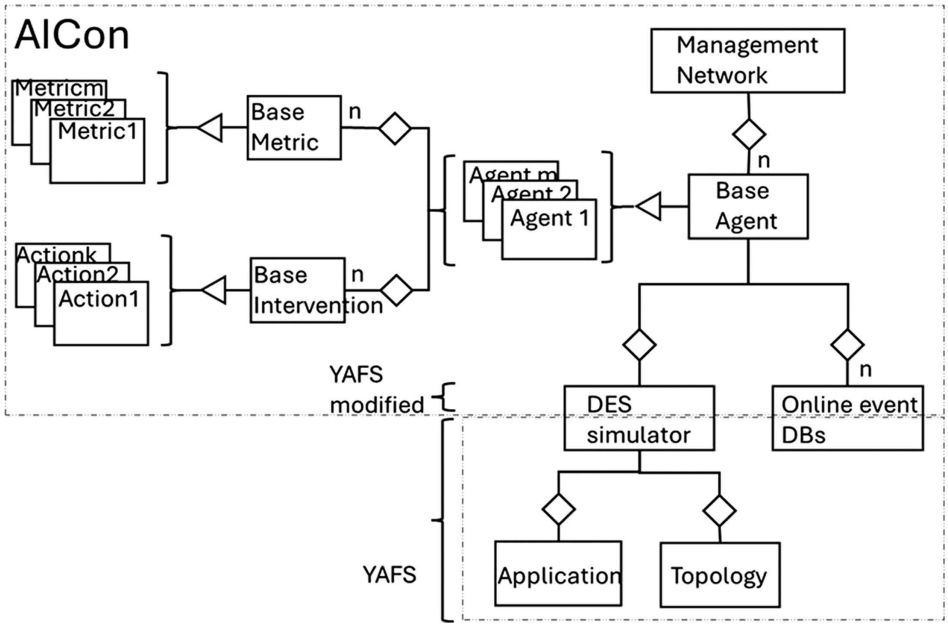

FIGURE 4. The AICon architecture consists of four main classes: management network, agent, metric, and intervention as well as several instantiations of the metric and intervention classes. It also has a modified version of the discrete-event simulator core class and modified version of the event data bases to be able to write and read simulation events at simulation time. DES: discrete event simulator; DBs: data bases.

beginning of the current work phase (Figure 3) and delivers these metrics as input to the user-defined f(obs) function.

This architecture enables agents to combine genuine online decision making with fully simulated dynamics (governed by the simulator's virtual clock), thereby making the execution of the perception–action loop truly simulated, directly overcoming the fundamental limitation of conventional simulators (*shortcoming 5*). This modular agent design directly addresses *shortcoming 1* by reducing the overall complexity of user solutions, and *shortcoming 2* by enhancing reusability of existing solutions. Lower complexity and higher reusability also help alleviate *shortcoming 4*.

The metric class is an abstract class that needs to be specialized for each specific metric. The base class facilitates access to the agent object using this metric. This agent is responsible for maintaining four data frames with the simulation events that have happened since the last work phase started on each of the observable nodes (see Figure 3). Table 2 presents the metric classes implemented so far. Similarly, the intervention class is also an abstract class that needs to be specialized for each specific intervention, and it facilitates access to the agent object and through it; it gains access to the simulator and to the parameters of the infrastructure and application layers. Table 3 presents the intervention classes implemented so far.

The project architecture is intentionally designed to mitigate the steep learning curve associated with existing CC simulators when developing decentralized management systems, as discussed in *shortcoming 4* earlier in the article. To foster accessibility across different expertise levels, AICon provides tiered customization modes that allow researchers with diverse technical backgrounds to engage comfortably. Specifically, users can operate at three levels of complexity:

1. *Configuration level*: Defining the management network via JSON while reusing existing agent, metric, and intervention classes.
2. *Extension level*: Implementing custom agents, metrics, or interventions (with deeper knowledge of the architecture, enabling an "expert mode" for advanced customization).
3. *Core level*: Modifying the simulator's internal components to meet new or specialized requirements (discussed in detail later).

This layered approach reduces entry barriers, promotes modular reuse, and supports both rapid prototyping and advanced experimentation.

CONCLUSION AND FUTURE WORK

In this article, we identified the key limitations in existing CC simulators that significantly hinder the development of realistic, decentralized online management systems. These include centralized control assumptions, tightly coupled autonomic components, limited integration with modern AI/ML ecosystems, steep learning curves for domain experts, and the failure to model management agents as resource-consuming entities with perception–action delays and overhead. Such shortcomings restrict the exploration of distributed autonomy under partial observability, constrained actuation, and multistakeholder coordination—core requirements for production-grade CCS deployments.

To address these challenges, we introduced AICon, an open source, Python-native simulation framework built atop the YAFS simulator. AICon establishes a clean three-layer usage model—infrastructure, application, and a novel *management network* layer—that explicitly decouples decision-making agents, observable metrics, and executable interventions. By treating agents as first-class simulated workloads with a configurable work–sleep phase, resource consumption, and scoped observability, AICon faithfully captures control overhead, self-induced contention, and coordination dynamics, yielding results with far greater realism and transferability to real systems.

Key contributions include 1) a modular agent abstraction with native support for partial observability and periodic execution, eliminating centralized biases; 2) reusable metric and intervention libraries with runtime event querying, enabling composable multiagent strategies (hierarchical, peer to peer, or stigmergic); 3) QoS-aware message processing via instruction scaling modulators, supporting explicit tradeoffs between latency, accuracy, and cost; 4) seamless integration with Python's AI/ML ecosystem, avoiding fragile interlanguage bridges; and 5) a JSON-driven configuration model with tiered customization levels, significantly lowering the entry barrier for interdisciplinary researchers.

Looking ahead, several directions will extend AICon's capabilities: 1) scalability via distributed simulation, offloading complex agent logic to remote workers (e.g., Ray and Dask) to support thousands of nodes; 2) an expanded intervention space, including dynamic deployment/migration of application modules and agents, and input queue redirection for load balancing and fault tolerance; 3) a simulation-to-reality pipeline, replacing the discrete-event core with a lightweight emulator (e.g., Kubernetes based) for

the infrastructure and application layers while retaining AICon's management network in simulation; and 4) a standardized benchmark suite of CCS scenarios (smart cities, Industrial IoT, or vehicular networks) with well-defined SLOs to enable reproducible evaluation.

In conclusion, AICon provides a flexible, realistic, and accessible foundation for advancing decentralized management in the CC. By enforcing modular design, realistic control modeling, and low-barrier entry, it empowers a broad research community to prototype, validate, and deploy the next generation of autonomous, agentic CCS systems.

ACKNOWLEDGMENTS

This work is supported by CNS2023-144359 and financed by MICIU/AEI/10.13039/501100011033 and the European Union NextGenerationEU/PRTR.

REFERENCES

1. S. Dustdar, V. C. Pujol, and P. K. Donta, "On distributed computing continuum systems," *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 4, pp. 4092–4105, Apr. 2023, doi: 10.1109/TKDE.2022.3142856.
2. C. Mechalikh, Z. Safavifar, F. Golpayegani, and F. Golpayegani, "Quality matters: A comprehensive comparative study of edge computing simulators," *Simul. Modell. Pract. Theory*, vol. 138, Jan. 2025, Art. no. 103042, doi: 10.1016/j.simpat.2024.103042.
3. R. Mayer, L. Graser, H. Gupta, E. Saurez, and U. Ramachandran, "EmuFog: Extensible and scalable emulation of large-scale fog computing infrastructures," in *Proc. IEEE Fog World Congress (FWC)*, 2017, pp. 1–6, doi: 10.1109/FWC.2017.8368525.
4. "An architectural blueprint for autonomic computing," IBM White Paper, Jun. 2005. [Online]. Available: https://users.cs.fiu.edu/~sadjadi/Teaching/Autonomic%20Grid%20Computing/CIS-6612-Summer-2006/AC-Blueprint-WhitePaper-V7.pdf
5. H. Gupta, A. V. Dastjerdi, S. K. Ghosh, and R. Buyya, ""iFogSim: A toolkit for modeling and simulation of resource management techniques in the internet of things, edge and fog computing environments," *Softw.: Pract. Experience*, vol. 47, no. 9, pp. 1275–1296, 2017, doi: 10.1002/spe.2509.
6. I. Lera, C. Guerrero, and C. Juiz, "YAFS: A simulator for IoT scenarios in fog computing," *IEEE Access*, vol. 7pp., pp. 91,745–91,758, 2019, doi: 10.1109/ACCESS.2019.2927895.
7. R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, and R. Buyya, "CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Softw.: Pract. Experience*, vol. 41, no. 1, pp. 23–50, 2011, doi: 10.1002/spe.995.
8. D. G. D. L. Iglesia and D. Weyns, "MAPE-K formal templates to rigorously design behaviors for self-adaptive systems," *ACM Trans. Auton. Adapt. Syst.*, vol. 10, no. 3, pp. 1–31, 2015, doi: 10.1145/2724719.
9. P. Arcaini, E. Riccobene, and P. Scandurra, "Modeling and analyzing MAPE-K feedback loops for self-adaptation," in *Proc. IEEE/ACM 10th Int. Symp. Softw. Eng. Adaptive Self-Manag. Syst.*, Piscataway, NJ, USA: IEEE Press, 2015, pp. 13–23, doi: 10.1109/SEAMS.2015.10.
10. Y. Abuseta and K. Swesi, "Design patterns for self adaptive systems engineering," 2015, *arXiv:1508.01330*.
11. J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, Jan. 2003, doi: 10.1109/MC.2003.1160055.
12. D. Weyns, S. Malek, and J. Andersson, "Lessons from the trenches and challenges for the future," in *Proc. ICSE Workshop Softw. Eng. Adaptive Self-Manag. Syst. (SEAMS)*, 2010.
13. D. Weyns, "An introduction to self-adaptive systems: A contemporary software engineering perspective," *IEEE Softw.*, vol. 37, no. 5, pp. 16–23, 2020.
14. T. Bures et al., "Software engineering for smart cyber-physical systems: Challenges and promising solutions," *ACM SIGSOFT Softw. Eng. Notes*, vol. 42, no. 2, pp. 19–24, 2017.
15. M. Macıas et al., "Domain-specific languages for CyberPhysical Systems: A systematic mapping study," *IEEE Access*, vol. 9, pp. 163–178, 2021.
16. R. Calinescu, L. Grunske, M. Kwiatkowska, R. Mirandola, and G. Tamburrelli, "Dynamic QoS management and optimization in service-based systems," *IEEE Trans. Softw. Eng.*, vol. 37, no. 3, pp. 387–409, May/Jun. 2011, doi: 10.1109/TSE.2010.92.
17. L. F. Bittencourt, J. Diaz-Montes, R. Buyya, O. F. Rana, and M. Parashar, "Mobility-aware application scheduling in fog computing," *IEEE Cloud Comput.*, vol. 4, no. 2, pp. 26–35, Mar./Apr. 2017, doi: 10.1109/MCC.2017.27.
18. B. Sedlak, V. C. Pujol, P. K. Donta, and S. Dustdar, "Equilibrium in the computing continuum through active inference," *Future Gener. Comput. Syst.*, vol. 160, Nov. 2024, pp. 92–108, doi: 10.1016/j.future.2024.05.056.
19. V. C. Pujol, B. Sedlak, T. Salvatori, K. Friston, and S. Dustdar, "Distributed intelligence in the computing continuum with active inference," 2025, *arXiv:2505.24618*.
20. V. C. Pujol and S. Dustdar, "Towards a prime directive of SLOs," in *Proc. IEEE Int. Conf. Softw. Services Eng. (SSE)*, Piscataway, NJ, USA: IEEE Press, 2023, pp. 61–70, doi: 10.1109/SSE60056.2023.00019.

21. H.-L. Truong, S. Dustdar, and F. Leymann, "Towards the realization of multi-dimensional elasticity for distributed cloud systems," *Procedia Comput. Sci.*, vol. 97, pp. 14–23, Dec. 2016, doi: 10.1016/j.procs.2016.08.276.
22. R. Mahmud, R. Kotagiri, and R. Buyya, "Fog computing: A taxonomy, survey and future directions," in *Internet of Everything: Algorithms, Methodologies, Technologies and Perspectives*, B. Di Martino, K. C. Li, L. Yang, and A. Esposito, Eds., Singapore: Springer-Verlag, 2017, pp. 103–130.

ILDEFONS MAGRANS DE ABRIL is with Pompeu Fabra University, 08018, Barcelona, Spain. He is the corresponding author of this article. Contact him at ildefons.magrans@gmail.com.

NEFELI-MARINA ROUSKA is with Pompeu Fabra University, 08018, Barcelona, Spain. Contact her at nefelimarina.rouska@upf.edu.

VICTOR CASAMAYOR is with Pompeu Fabra University, 08018, Barcelona, Spain. Contact him at victor.casamayor@upf.edu.

SCHAHRAM DUSTDAR is with Pompeu Fabra University, 08018, Barcelona, Spain, and TU Wien, 1040, Vienna, Austria. Contact him at schahram.dustdar@upf.edu.