NRG-CHAMP: Responsive Goal-Driven Cooling and Heating Automated Monitoring Platform

Andrea Cantarini
Macroarea di Ingegneria
Università di Roma "Tor Vergata"
Roma, Italia
andrea.cantarini@alumni.uniroma2.eu

Giuseppe Valerio Cantone
Macroarea di Ingegneria
Università di Roma "Tor Vergata"
Roma, Italia
giuseppevalerio.cantone@alumni.uniroma2.eu

***Abstract*—Buildings, especially commercial and public facilities, consume significant energy for heating, ventilation, and air conditioning (HVAC)—often on the order of 40% of total usage. NRG-CHAMP is an integrated platform designed to reduce HVAC energy consumption and promote sustainability in large buildings through a combination of real-time monitoring, autonomic control, transparent data logging, and user engagement. A dense IoT sensor network feeds environmental and usage data into a Monitor-Analyze-Plan-Execute (MAPE) control loop that dynamically adjusts HVAC operations to meet zone-specific temperature targets. The system is built with a microservices architecture that leverages a high-throughput messaging backbone (Apache Kafka) to decouple data ingestion, analysis, and actuation. An immutable blockchain ledger records aggregated sensor readings and control actions for auditability and compliance, while a gamification module uses the recorded performance metrics to incentivize energy-saving behavior among occupants via competitions and leaderboards. Key design features include hierarchical zone modeling, fault-tolerant orchestration with a circuit breaker pattern, and modular deployment in containerized services. Preliminary implementation and design rationale are presented, highlighting how NRG-CHAMP provides a responsive, scalable, and transparent solution for smart energy management in buildings.**

I. Introduction

Energy efficiency in building climate control is a pressing concern, as HVAC systems can account for a substantial fraction of a facility's energy consumption. Traditional building management systems often rely on static schedules or manual tuning, which may fail to respond optimally to changing conditions or occupant behavior. In recent years, emerging approaches have explored the use of IoT sensors and automation to create "smart" HVAC control loops. In particular, the autonomic computing paradigm—exemplified by the Monitor-Analyze-Plan-Execute (MAPE) loop—provides a framework for self-adaptive climate control that continuously monitors conditions, analyzes needs, plans adjustments, and executes changes without human intervention. Another trend is the integration of blockchain technology to ensure transparent and tamper-proof logging of energy data and control actions, enabling trustworthy verification of savings commitments. Likewise, gamification techniques have been applied to encourage energy-conscious behavior among building occupants, showing promising results in reducing consumption through competition and feedback. NRG-CHAMP (Responsive Goal-driven Cooling and Heating Automated Monitoring Platform) is proposed as a holistic solution that synergistically combines these strategies. It features a dense network of sensors feeding a rule-based MAPE control engine, a blockchain-backed ledger for data traceability, and a gamified scoring system to incentivize users. The platform targets large multi-zone buildings (e.g., office campuses, hospitals, university complexes), emphasizing scalability, resilience, and compliance with energy regulations. A microservice-based architecture underpins NRG-CHAMP, enabling independent deployment and scaling of its functional components (data ingestion, control, ledger, analytics, etc.) across distributed infrastructure. Services communicate via asynchronous messaging (Kafka) to decouple real-time control loops from data persistence and external queries. Additionally, all external interactions are unified behind secure APIs to facilitate integration with dashboards and third-party systems. This paper describes the design and implementation of NRG-CHAMP. Section II provides background on the relevant technologies and concepts – including autonomic control loops, microservice architectures, blockchain in energy systems, and gamification for sustainability. Section III outlines the overall solution design, detailing the system architecture and major components. Section IV delves into solution details such as the data flow, module functionality, and implementation specifics. Section V discusses key design decisions, trade-offs, and the rationale behind the chosen architecture (e.g., messaging topology and fault tolerance patterns). Finally, Section VI concludes the paper and highlights future work.

II. Background

Autonomic Control for HVAC: The self-managing MAPE loop originates from autonomic computing and provides a conceptual model for automation in complex systems. In the HVAC context, this translates to continuously monitoring environmental variables (temperature, humidity, occupancy, etc.), analyzing deviations from comfort or efficiency targets, planning corrective actions (such as adjusting thermostats or fan speeds), and executing those actions via actuators (heating/cooling units). This closed-loop control enables real-time adaptation to disturbances (like weather changes or occupancy fluctuations) without human intervention. By applying the MAPE framework, NRG-CHAMP inherits proven

principles of self-optimization and self-correction, leading to more efficient energy usage compared to fixed schedules. Each autonomic controller (for a zone or building) operates with high-level goals (temperature setpoints and policies) defined by administrators, aligning with the idea of management by objectives in autonomic systems. **IoT and Building Management**: Modern buildings are increasingly instrumented with IoT sensors and connected actuators. These devices produce high-frequency data streams that legacy building management systems may struggle to ingest and respond to in real time. NRG-CHAMP builds on IoT infrastructure to obtain fine-grained, up-to-the-moment data about the indoor environment and system performance. A message-oriented middleware (Kafka) is employed to handle the high throughput and distribution of sensor data, providing durability and decoupling between producers (sensors/ingestion services) and consumers (analysis, control, and logging services). The use of Kafka topics and partitions is tailored to the building's physical layout (as detailed in Section IV) to ensure that data from different zones is processed in parallel without interference. This design reflects best practices in IoT system scalability, where streaming data pipelines absorb bursts and allow independent scaling of downstream processing for different data sources. **Microservices Architecture and Resilience**: NRG-CHAMP is designed as a suite of containerized microservices rather than a monolithic application. This architectural style offers several benefits in the context of building management: each service (ingestion, aggregator, control engine, ledger, etc.) can be developed, deployed, and scaled independently, which is crucial when dealing with potentially thousands of sensors and multiple buildings. The services communicate through well-defined APIs and message queues, isolating internal failures and enabling modular upgrades. To ensure high availability, the platform employs the circuit breaker pattern – a defensive design technique that prevents cascading failures across microservice calls. A shared circuit breaker module intercepts inter-service requests; if a downstream service is unresponsive or repeatedly failing, the circuit opens to stop immediate retries, and only after a cooldown period will a health probe allow calls to resume. This approach, popularized by Nygard's reliability patterns, improves overall system resilience by avoiding resource exhaustion during outages.

III. Solution Design

Architecture Overview: NRG-CHAMP's architecture follows a distributed microservices design based on decomposition by business capability, and deployed in a cloud-native environment. Figure 1 illustrates the high-level architecture and data flow. At the edge, each building zone is equipped with at least one sensor device, monitoring temperature, and one or more actuator devices (such as smart thermostats, HVAC fan coil units). These edge devices communicate with the core platform over network channels (e.g., Kafka/HTTP). The core platform comprises multiple collaborating services:

Data Aggregator: a service that subscribes to raw sensor data streams, supporting persistent communication via Kafka,

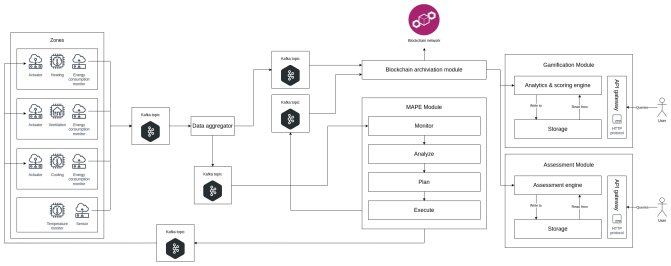

Fig. 1. NRG-CHAMP data flow

performing preprocessing such as outlier filtering and aggregation on a per-zone basis. The aggregator groups readings into discrete time windows (e.g., one-minute intervals) to produce a summary or batch for each zone. This summarized data is forwarded both to the control engine and to the ledger (for record-keeping). **MAPE Control Engine**: the heart of the adaptive control loop, conceptually divided into four stages (Monitor, Analyze, Plan, Execute). The Monitor stage continuously receives the cleaned, aggregated sensor metrics for each zone from the aggregator (via Kafka subscription) and maintains a sliding window of recent data points. The Analyze stage checks current conditions against desired targets and identifies anomalies or inefficiencies (for example, detecting if a zone's temperature is drifting beyond acceptable bounds or if a room is being conditioned while unoccupied). The Plan stage formulates adjustments: if the temperature deviates from the target, it decides whether to activate heating or cooling, taking into account a hysteresis band to avoid rapid oscillations. It also sets an appropriate fan speed proportional to the temperature difference, aiming for a gradual correction rather than an overshoot. The Execute stage then issues the calculated commands to the actuators in the field via the messaging system. Each actuator (or group of actuators) in a zone listens on its dedicated channel for new commands and applies only the latest command received, discarding any stale commands that arrived earlier. This guarantees that if the control loop produces frequent updates, the devices always act on the most up-to-date decision (preventing backlog execution of outdated actions). The MAPE Engine's design allows multiple zones to be managed concurrently by one instance if needed, and it ensures that each zone's control decisions are made with fresh data.

Blockchain Ledger Service (Ledger): responsible for secure logging of system state and actions. Both the Data Aggregator and the MAPE Engine feed into this service: for every time epoch (e.g., each minute or control cycle), the aggregator sends a summary of sensor readings and the MAPE Engine sends the corresponding control action outcome. The ledger service pairs these inputs by zone and epoch to form a complete record. If one part of the pair is missing, the ledger can estimate a placeholder based on previous and next values to maintain continuity. These matched records are then added to the current block in an internal blockchain. When a block's data capacity is reached, the ledger service finalizes

the block by computing a cryptographic hash of its data and storing that in the block header, then hashes the entire header, and starts a new block, linking it to the chain by including the prior block's header hash. This creates a tamper-evident chain of records. The ledger also maintains an index mapping zones to the blocks that contain their data, which accelerates queries (e.g., quickly finding all records for a specific zone during an audit). In addition, a Ledger Query API is exposed for external auditors or other services to retrieve blockchain records in a controlled manner (with permissions). The blockchain integration in NRG-CHAMP is modular – it could operate as a private blockchain network specifically for the building or integrate with an existing blockchain (for example, writing hashed records to a public chain for external notarization).

Analytics and Gamification Engine: a post-processing component that consumes data from the ledger to derive higher-level insights. This engine computes performance metrics such as energy efficiency scores, comfort compliance (how often temperature stayed in target range), and aggregate consumption over various periods. Crucially, it implements the gamification logic: updating the scoreboards and leaderboards for the ongoing energy-saving competitions. The gamification engine receives a trigger whenever the ledger writes a new block or entry (indicating new validated data). It then calculates, for example, the average energy usage across all competing entities (floors or offices) and awards points to those that performed better than the average, while penalizing those that performed worse. Over time, these points contribute to each unit's ranking. The engine is designed to support customizable hierarchical competitions – e.g. individuals within an office, offices within a floor, floors within a building, and building vs. building – ensuring that participants compete at comparable scales and everyone has an incentive to improve relative to their peers. The results are stored in a database accessible by the dashboard/UI.

All these services are containerized (Docker) and orchestrated using Kubernetes (or a similar platform) for deployment on cloud or on-premise servers. This setup can provide elasticity (services can be replicated for load balancing), self-healing (failed containers can be automatically restarted or relocated), and rolling update capabilities. The system's design also emphasizes easy maintainability: logging and monitoring are built into each service so that the health and performance of the platform can be continuously observed.

Facility Topology and Zoning: A core design principle of NRG-CHAMP is alignment with the physical topology of the facility. Buildings are modeled hierarchically in zones and zones-groups, reflecting real groupings like rooms, collections of rooms, floors, and buildings. This hierarchy (e.g., Level 0: Room, Level 1: Group of Rooms, Level 2: Floor, Level 3: Building, Level 4: Campus) is used both for control (e.g., a MAPE instance might manage all zones on one floor) and for gamification (competition can be structured at any level). By mirroring the topology in the system architecture, the solution can enforce policies consistently at different scales – for example, a comfort policy might require that all rooms maintain 21–23 °C, while an energy policy might limit a building's total HVAC power draw during peak hours. The message broker topics are likewise partitioned by zone to localize data flows and control actions (detailed in Section IV). This design eases scaling (a new building or floor can be added by spinning up corresponding service instances and topics) and limits interference between zones – heavy activity in one zone (say a busy conference hall) will not delay processing for another zone (like a small office), as their data and control loops are kept separate.

Data Flow Summary: The end-to-end data flow can be summarized as follows: Sensors publish telemetry onto zone-specific message queues. The aggregator service reads from these queues, cleans and batches data, then publishes summaries to downstream topics. The MAPE engine consumes aggregated data, computes control decisions, and sends commands to actuators. Actuators apply the latest commands, affecting the environment (closing the loop as new sensor readings will reflect these changes). In parallel, both sensor summaries and control actions stream into the ledger, which securely logs them on the blockchain. Analytics derive insights from stored data, and the gamification module updates user-facing performance metrics. Throughout this process, each component operates asynchronously, communicating via messages, which allows the system to handle bursts (by buffering in Kafka) and remain robust if certain services are temporarily slow or offline (they can catch up from the message queue once available).

IV. Solution Details

The following details highlight key implementation aspects and how the platform realizes the design.

A. Kafka Topic Topology

Apache Kafka serves as the communication backbone between NRG-CHAMP services. The topic design is carefully chosen to mirror the system's operational domains:

Devices → Aggregator: Each physical zone is allocated its own Kafka topic for raw sensor data. This topic-per-zone strategy naturally isolates traffic – e.g., "device.readings.Room101" topic for Room 101's sensors – and allows independent scaling and access control per zone. Within each such topic, each device (sensor) publishes to a dedicated partition (identified by the device ID). Partition-per-device preserves the chronological order of readings from any single sensor and prevents high-frequency devices from starving others. The Aggregator service for that zone (or set of zones) is the consumer: it implements a round-robin poll across partitions, reading the next unread message from each device in turn to ensure fairness. It marks messages as processed up to the current time epoch and temporarily skips ahead when it encounters data from a future epoch, cycling through devices so that no single fast stream monopolizes attention. This design guarantees bounded latency for every device's data to be picked up, even if some devices send more frequently than

others. Each aggregator instance can handle multiple zones, hence it also round-robins between zone topics to balance its work. After cleaning and aggregating the sensor readings (removing any outliers/noise and computing aggregate values like average temperature and total electric consumption over the window), the aggregator produces two outputs: it writes the processed batch to a MAPE input topic, and in parallel to a Ledger topic (both described below).

Aggregator → MAPE: For the pipeline from data aggregator to the control engine, Kafka topics are set up on a per Aggregator/MAPE pair basis. In practice, an aggregator writes into one topic and one dedicated MAPE service instance reads from it. However, to allow a single MAPE service to handle all the zones served by the aggregator, the topic is partitioned by zone (one partition per zone under that MAPE's purview). This way, a single control service instance can sequentially handle updates from multiple zones without intermixing their messages. The MAPE consumer performs its own round-robin over the zone partitions, processing data in each partition one zone at a time. Importantly, the MAPE only needs the latest state per zone for decision-making, so if multiple batched readings queued up (e.g., during a brief backpressure or if the control loop runs slightly slower than ingestion), it will read all messages in the partition but only act on the most recent one, discarding older, now-obsolete batches. This ensures that any backlog of outdated sensor data doesn't cause a cascade of outdated control actions; instead, the control jumps straight to using the newest information available. This design choice reduces unnecessary actuation oscillations and computations, at the cost of ignoring some intermediate data – a reasonable trade-off given that the environment's state has superseded those older readings.

MAPE → Actuators: After computing actions, the MAPE Engine publishes commands to actuator topics organized per zone. Each zone has its own control topic (e.g. "zone.commands.Room101"), and within that topic, each actuator device (heater, cooler, fan) is assigned a partition, since they operate independently. Each actuator device subscribes to its partition and, similar to the MAPE, will take the latest command and disregard older ones to avoid any queue buildup. This mechanism is particularly useful if network or processing delays cause a small backlog – the device won't oscillate by executing now-irrelevant older commands, thereby maintaining stable operation. The use of zone-scoped topics also means actuators from different zones (which are logically independent) never consume from the same queue, eliminating any chance of mix-up or interference.

Aggregator MAPE → Ledger: For feeding the blockchain ledger, Kafka topics serve to funnel data to the ledger service reliably. The design uses one ledger topic per zone, with two partitions per topic: one designated for aggregator outputs (sensor data summaries) and one for MAPE outputs (control decisions). Both the aggregator service and the MAPE service produce messages to this topic for their respective partition. The ledger service is the sole consumer of all these topics. By reading from both partitions of each zone's topic, the ledger can match sensor data with the corresponding action based on a timestamp or epoch number. If an entry from one side is missing for a given epoch (e.g., sensor data was momentarily unavailable), the ledger applies an estimation strategy to fill the gap (this performing interpolation from adjacent readings). This ensures that every time slot has a complete record before committing it to the blockchain, which simplifies later auditing (the records are uniform and self-contained). The two-partition scheme cleanly separates the provenance of data within the combined record (so one can always tell which part came from sensors vs. controllers) and aligns with the CQRS (Command Query Responsibility Segregation) concept of separating write models – here, the aggregator and MAPE provide two streams of "commands" that are merged for the read model in the ledger. The use of Kafka here also buffers the ledger input: if the blockchain insertion is slower than real-time, the backlog accumulates in the Kafka topic rather than dropping data. The ledger service can then catch up or scale out (if multiple ledger writer instances handle different zone sets) to meet throughput demands.

This Kafka-mediated design contributes greatly to the platform's scalability and reliability. By mapping each logical unit (device, zone, etc.) to Kafka topics/partitions, NRG-CHAMP achieves strong isolation and parallelism. As noted in the design rationale, this mapping minimizes cross-talk and allows independent scaling and flow control per zone or device group. For example, if one building wing is extremely sensor-dense or active, its dedicated topics can be handled by a separate set of aggregator/MAPE instances, scaled out as needed, without affecting the rest of the system. Kafka's persistent log also provides a replay capability — if a service goes down and recovers, it can reread recent messages to restore state, adding fault tolerance.

V. Discussion

Developing NRG-CHAMP involved several design trade-offs and architectural considerations influenced by the project's goals of scalability, energy efficiency, and trustworthiness. This section discusses the rationale behind key choices and their implications. **Hierarchical Zoning and Scalability**: One fundamental decision was to architect the system around the building's hierarchical topology (rooms → floors → buildings). This led to the Kafka topic partitioning scheme where each zone's data is handled quasi-independently. The benefit of this approach is clear in terms of scalability and containment: processing for one zone can be scaled out simply by adding more instances for that zone's consumers, without affecting other zones. It also localizes the impact of any sensor malfunctions or anomalies – for example, a sensor that suddenly outputs a flood of erroneous data will only congest its zone's pipeline and not the entire system. The trade-off is a slightly more complex configuration (topics and partitions need to be defined for each zone and maintained as zones are added or removed). However, this complexity is mitigated by automation scripts and Kubernetes orchestration (which can template out new deployments for new zones). Moreover, the

strict partitioning pays dividends in performance: it minimizes cross-talk and preserves ordering where needed, as intended. In essence, the system scales in a modular fashion – each added building or floor increases load mostly in a linear, isolated way – which is crucial given the project's requirement to potentially support thousands of zones and millions of data points in real time. **Microservices vs. Monolith**: Given the breadth of functionality (ingestion, control, ledger, analytics, etc.), a microservices architecture was chosen over a monolithic design to allow independent development and deployment of components. This choice aligns with modern enterprise software practices, especially considering that different components may have different resource profiles and update cycles. For instance, the control logic (MAPE engine) might be refined frequently (or even replaced with more advanced algorithms like machine learning models), whereas the blockchain ledger code might remain relatively stable but need tight security audits. By decoupling them, changes in one do not risk destabilizing others. Additionally, microservices provide fault isolation – if the analytics service has a heavy computation that lags, it will not directly stall the ingestion of sensor data, thanks to the buffered messaging in between. One potential downside of microservices is the added complexity in deployment and inter-service communication. NRG-CHAMP addresses this with container orchestration and a service registry (or Kubernetes service discovery) so that services can find each other reliably. The inclusion of a global circuit breaker mechanism was also motivated by this architecture: in a distributed system with many network calls, graceful handling of partial failures becomes vital. The shared circuit breaker ensures that microservice interactions are robust, preventing failures from cascading – a known challenge in microservice systems. Without it, a slow response or a downed service could trigger a chain reaction of threads blocking across multiple services. With it, NRG-CHAMP demonstrates resilience: for example, if the blockchain node goes offline, the ledger service's breaker opens and the system switches to a mode where sensor data queues in Kafka and control continues unaffected. Only once the ledger recovers will those records be flushed to the blockchain, maintaining eventual consistency. This design choice to emphasize resilience reflects the project's non-functional requirements for high availability. **Use of Blockchain Ledger**: The decision to integrate a blockchain for data logging was driven by requirements for traceability, transparency, and compliance. By having an immutable ledger of all key events, NRG-CHAMP can produce auditable evidence of energy performance – for example, to prove that a building stayed within a mandated energy budget or to verify savings under an ESCO (Energy Service Company) contract. An alternative could have been a traditional database with strict access controls and regular backups, which is simpler and faster. However, a conventional database does not provide the same level of intrinsic trust: records can be altered by those with high privileges, and proving data integrity to a third party is non-trivial. The blockchain, in contrast, inherently assures that once data is committed with the hash chaining mechanism, any tampering would break the chain and be evident. The trade-off is performance and complexity. Writing to a blockchain (even a private one) is slower than to a normal database, and it requires handling cryptographic operations and possibly consensus overhead. NRG-CHAMP's design mitigates these issues by a) batching data into blocks (amortizing the overhead per record), and b) segregating the ledger writes out of the critical control path (so HVAC adjustments are not delayed by ledger operations). Essentially, the system opts for a slight delay in recording data in exchange for robust security of records. During discussions, it was acknowledged that blockchain integration might be overkill for some deployments (e.g., a small building that doesn't need such strong auditability). Thus, the implementation was made modular – the ledger service could be configured in "stub" mode to simply hash and log data to a secure database if blockchain infrastructure is unavailable, with the option to switch to a full blockchain later. This flexibility was a conscious design choice to not overly tie the project to one technology, while still validating the concept that external regulators can directly query an unforgeable log via the Ledger Query API. **Kafka Communication Patterns**: The use of Kafka and the specific topic structures were justified by the need for high-throughput, ordered, and decoupled messaging. Other message brokers (RabbitMQ, MQTT brokers, etc.) were considered, but Kafka's ability to persist streams and allow multiple consumer groups made it ideal for the multi-branch workflow in NRG-CHAMP (data goes to control, ledger, and storage in parallel). The topic-per-zone and partition-per-device pattern is a direct result of mapping the physical world to the messaging domain. This design was validated against expected device counts and message rates – by sharding topics per zone, the system can naturally distribute load (e.g. across Kafka broker clusters or across consumer threads) and keep each consumer's workload manageable. A possible concern is the overhead of managing many topics in Kafka (since large building portfolios could mean hundreds or thousands of topics). In practice, Kafka can handle a high number of topics, and naming conventions plus automated scripts help in provisioning them. Moreover, the benefit in throughput and independence outweighs the management overhead: the project documents highlight that this layout avoids collisions and enables back-pressure control per boundary. In other words, if one zone's processing is slow, Kafka will only build up that zone's topic, and won't block others – back-pressure is isolated. During design, an alternative considered was to use a single topic for all sensor data with keys for zones, but that would interleave data and complicate consumers. The chosen route, although more granular, is clearer and was deemed more robust. **Design Patterns – Circuit Breaker, Decomposition by Business Capbility, and CQRS**: The adoption of CQRS (Command Query Responsibility Segregation) is evident in the separation between the write model (i.e. the blockchain ledger ingesting sensor and action commands) and the read model (the analytics and dashboard APIs optimized for queries). This segregation allows the write side to be

tuned for efficient, append-only operations, while the read side can be structured for fast lookups and aggregations without interfering with the write path. Meanwhile, the Circuit Breaker pattern is applied across inter-service communication to prevent cascading failures: when a service repeatedly fails, the circuit "opens" and temporarily halts requests until a probe confirms recovery, thereby preserving system stability. Finally, the architecture uses Decomposition by Business Capability to divide the system into services aligned with coherent business domains (e.g., sensing and ingestion, control decision, ledger, gamification). Each service encapsulates one or more business capabilities, enabling teams to evolve and scale components independently without exposing internal internals. **Fault Tolerance and Bulkheads**: In addition to the circuit breaker, the architecture implicitly implements bulkheads – the isolation of components so that a failure in one does not flood others with exceptions. The use of message queues acts as a buffer and bulkhead. For instance, if the MAPE engine fails, the aggregator will continue to publish to Kafka; the data will queue up until the MAPE restarts, rather than being lost or causing upstream to crash. This was a conscious resilience strategy: using asynchronous messaging wherever possible to decouple execution. Only in cases where waiting is not a problem (e.g. a synchronous API call from a user to fetch data) does the system use direct calls. This approach was informed by the requirement for reliability and availability; it acknowledges that partial failures will happen (sensors might disconnect, services might restart) and ensures the system can recover gracefully or continue in a degraded mode. Logging of failures and automated alerts are in place so operators know when something needs attention, but the design aim is to minimize human intervention by making the system self-stabilizing. **Gamification and Behavioral Impact**: On the human side, one consideration was how to design the gamification in a way that is fair and effective. The hierarchical competition model was chosen so that occupants are compared within similar groupings (one office vs another, rather than an absolute metric that might unfairly disadvantage those in tougher conditions like south-facing rooms). This relative performance approach (comparing against peer average) is intended to ensure that everyone can contribute, as noted by making sure each participant has a chance regardless of initial ranking. The platform's flexibility allows tweaking the rules as more is learned about user responses. The push vs. pull design for gamification and assessment reflects different needs: Assessment (for compliance verification) is done on-demand to avoid constant overhead – an auditor can query when needed – whereas Gamification is push-driven to give immediate feedback and keep users engaged. This dichotomy was a deliberate decision to balance network usage and freshness of information; continuous pushing of audit data was not necessary and would create noise, while delaying gamification updates would reduce user engagement. In summary, the design choices in NRG-CHAMP were guided by a combination of established best practices in distributed systems and the specific domain requirements of building energy management. The use of Kafka and microservices addresses scalability and performance head-on, the blockchain ledger addresses trust and compliance concerns, and the gamification and MAPE control loop address the dual challenge of technical optimization and human behavior. The project demonstrates how these elements can be orchestrated into a cohesive platform. There were trade-offs – complexity vs. transparency, immediacy vs. buffering, centralization vs. decentralization – and the chosen solutions aim to strike a balance that meets the primary objectives. Future evaluations (e.g. pilot deployments) will further inform these choices, but the current architecture is flexible enough to adjust parameters (like threshold settings, component replication counts, or even swap out modules) as needed based on real-world feedback.

VI. Conclusions

This paper presented NRG-CHAMP, a comprehensive platform for smart HVAC energy management that leverages IoT-driven control, blockchain-based data auditing, and gamified user engagement. We described how the system implements a responsive autonomic control loop (MAPE) across a distributed microservice architecture, coordinating sensors, analytic engines, and actuators to continuously optimize building climate control. By integrating a tamper-proof blockchain ledger, NRG-CHAMP provides a high level of transparency and trust in reported energy performance – a feature that sets it apart from conventional building management systems. The inclusion of gamification adds a novel dimension that tackles the human factor, encouraging occupants to participate in energy saving in a measurable and competitive way.

Key architectural features of NRG-CHAMP include its scalable messaging backbone (with Kafka topics partitioned per zone/device to ensure throughput and isolation) and its robust fault-tolerance measures (such as the global circuit breaker and modular fail-safe design). These enable the platform to maintain real-time responsiveness and high availability in the face of growing scale and component failures. The design patterns adopted (CQRS, service isolation, etc.) and the hierarchical modeling of facilities contribute to a solution that is both extensible and adaptable to different deployment sizes – from a single large building to a campus of buildings. The microservices approach facilitates ongoing development: each module (ingestion, control, ledger, analytics) can evolve independently, allowing incorporation of new technologies (for instance, machine learning for anomaly detection, or integration with smart grid demand-response signals) without a complete system overhaul.

Our implementation thus far has validated the core data flows and control logic under simulation and via a private blockchain ledger. The system demonstrates the ability to maintain consistent control, resilience to missing or asynchronous inputs (e.g. by interpolation), and ledger integrity under adverse conditions. Moving forward, rather than focusing solely on real-world deployment, we propose enhancements directed inward toward the system's architecture.

- Introduce a hierarchical MAPE engine: instead of one global controller, deploy nested MAPE loops (e.g. per room, per floor, per building) that coordinate via supervisory control, enabling local optimizations while preserving global objectives.
- Incorporate zone-intrinsic energy differentiation: classify zones by intrinsic usage profiles (e.g. public-facing rooms vs. staff-only rooms) and tailor control strategies and target setpoints accordingly to reflect their baseline loads and operational constraints.
- Strengthen secure communication and access control: enforce end-to-end encryption (TLS or equivalent) across all inter-service links and device communication channels, and introduce finely-grained ACLs (access control lists) or capability-based security to restrict which components or operators may issue control commands, view analytics, or query ledger entries.
- Evolve the gamification and incentive subsystem to reflect zone-level heterogeneity: reward energy reductions relative to zone-specific baselines rather than global averages, allowing more equitable competition among zones with different intrinsic consumption.
- Enable adaptive policies and predictive control using historical data and machine learning within modules, making planning more proactive based on anticipated occupancy, weather, and usage variance.

These internal evolutions aim to increase the autonomy, resilience, fairness, and security of NRG CHAMP itself, making it more robust and adaptable—without presupposing external pilot deployments.

In conclusion, NRG-CHAMP demonstrates an innovative convergence of IoT, distributed systems, and behavioral incentives to address the challenge of sustainable climate control in buildings. By adhering closely to the requirements and real-world constraints documented in the project, we have shown a pathway to significantly reduce energy waste while engaging stakeholders through transparency and competition. We believe this platform, with further development and deployment, can become a valuable tool in the toolkit for smart building management and contribute to broader energy efficiency and carbon reduction initiatives in the built environment.

REFERENCES

[1] Department of the Environment and Energy (Australia), "HVAC Factsheet – Energy Breakdown," Sep. 2013.

[2] J. O. Kephart and D. M. Chess, "The Vision of Autonomic Computing," *IEEE Computer*, vol. 36, no. 1, pp. 41–50, 2003. DOI: 10.1109/MC.2003.1160055.

[3] L. Sun, "Exploration of Blockchain-Based Building Energy Management," *Smart Systems and Green Energy*, vol. 6, no. 1, pp. 60–67, 2024. DOI: 10.23977/ssge.2024.060108.

[4] M. Nygard, *Release It! Design and Deploy Production-Ready Software*, 2nd ed. Dallas, TX, USA: Pragmatic Bookshelf, 2018.

[5] M. Casals et al., "Assessing the effectiveness of gamification in reducing domestic energy consumption: Lessons learned from the EnerGAware project," *Energy and Buildings*, vol. 210, Art. no. 109753, 2020.