Self-Adaptation of Loosely Coupled Systems across a System of Small Uncrewed Aerial Systems

Theodore Chambers, Jane Cleland-Huang, Michael Vierhauser

PREPRINT – accepted for publication at the **12th International Workshop on Software Engineering for Systems-of-Systems and Software Ecosystems** 2024

Self-Adaptation of Loosely Coupled Systems across a System of Small Uncrewed Aerial Systems

Theodore Chambers (University of Notre Dame; tchambe2@nd.edu)
Jane Cleland-Huang (University of Notre Dame; JaneHuang@nd.edu)
Michael Vierhauser (University of Innsbruck; Michael.Vierhauser@uibk.ac.at)
DOI: 10.1145/3643655.3643882

ABSTRACT

The use of small autonomous Uncrewed Aerial Systems (sUAS) for Emergency Response requires rapid deployments into shared operational environments. We refer to these as "Pop-up Drone Zones" (PuDZ), representing a System of Systems (SoS), in which individual systems provide services such as air traffic control, environmental modeling, and support for sUAS autonomy. Each system needs the ability to configure itself dynamically at the start of the mission and adapt throughout the mission, in response to occurring changes. However, system-level choreography at the SoS level can be challenging, as it requires a global perspective of all individual systems, and a deep understanding of their inter-dependencies. We, therefore, propose a publish-subscribe architecture that enables the exchange of regional data between MAPE-K-enabled systems in support of coordinated self-adaptation whilst maintaining loose coupling between the interconnected systems. We illustrate and validate our approach through several PuDZ-related change scenarios in simulation and physical field tests.

1 Introduction

The increasing deployment of small autonomous Uncrewed Aerial Systems (sUAS) for supporting emergency response missions [20] demands new paradigms for ensuring rapid deployment and their safe, secure, and reliable operation within the real world. Based on emerging and unfolding situations in the field, missions need to be planned, and sUAS subsequently deployed on the scene within minutes or even seconds [16]. To address this problem, we are developing a self-organizing, self-managing *Pop-up Drone Zone* (*PuDZ*). *PuDZ* is implemented as a System of Systems (SoS), defined by ISO/IEC/IEEE 21839 as a "set of systems or system elements that interact to provide a unique capability that none of the constituent systems can accomplish on its own" [13], and where each system exhibits *operational* and *managerial* independence [18]. Many of the primary systems within the *PuDZ* infrastructure are self-managed around a MAPE-K loop with capabilities to *monitor* their environments, *analyze* any collected data, and then *plan* and *execute* a course of actions supported by their individual internal *knowledge base*. Furthermore, systems can be dynamically added, removed, or replaced by other systems within the SoS. An overview of the *PuDZ* SoS is depicted in Fig. 1. It constitutes of three main parts with varying degrees of managerial and operational independence: systems responsible for ATC (air traffic control); an Environmental Digital Shadow (EDS) containing an environmental model that serves as a representation of the real world; several supporting infrastructure and global services including runtime monitoring and policy management; and each individual sUAS entering the *PuDZ*. There is a delicate balance between system-level choreography and system independence. On one hand, systems incorporate their own individual MAPE-K loops and self-manage and/or self-adapt independently, while on the other hand, internal adaptations in one system can have a trickle-over effect on others, causing them to adapt in response. Weyns and Andersson [31] have proposed three distinct high-level architectures for deploying MAPE-K across an SoS, characterized by *local adaptations*, *cross-system (regional) monitoring*, and *collaborative adaptations*.

In this paper, we present a solution inspired by, and augmenting, Weyns and Andersson's earlier work, to support both SoS and system-level adaptation. The SoS is monitored for compliance with SoS policies, while at the system level co-adaptation is supported across loosely-coupled managed systems within dynamically adapting SoS. This is accomplished by connecting system-level MAPE-K loops via a dedicated message bus. Each system publishes topics of potential interest to other systems and subscribes to topics that are relevant for its own purposes. Separating the *Monitor* component into two parts ensures that both *trusted* local data and potentially *untrusted* regional data is appropriately handled. This approach differs from previously published patterns of distributed self-adaptive systems [30] as it is designed for deployment in an SoS where loosely coupled coordination is highly valued, adding the notion of policies, and an SoS-level MAPE-K loop. We illustrate the efficacy of our solution through a series of examples, simulations, and field tests with physical sUAS that the approach is effective for propagating diverse adaptation-triggering changes across various systems in the SoS. This paper, therefore, makes the following contributions: First, it extends previous work on SoS self-adaptation, proposing an architecture

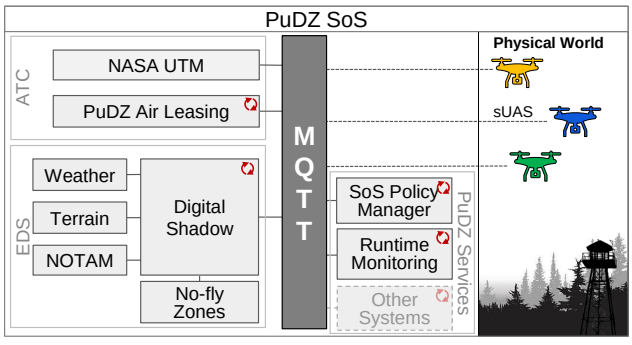

Figure 1: The *PuDZ* Ecosystem includes infrastructure services, an environmental digital shadow (EDS), air traffic control (ATC), and the sUAS that fly within the ecosystem. Systems managed by MAPE-K are annotated with the cycle icon.

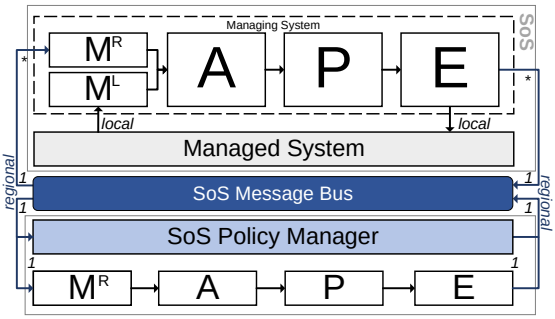

Figure 2: The MAPE-K Pattern for SoS, supporting two logical monitors responsible for local (trusted) and regional (potentially untrusted) inputs.

for self-adaptation across an SoS that is characterized by loose coupling between managed systems. Second, it describes a self-organizing, self-managing *PuDZ*, representing a challenging, real-life application in a rapidly emergent area of CPS deployment.

The remainder of the paper is laid out as follows. Section 2 presents our architecture for coordinating self-adaptation across loosely coupled services in an SoS. Section 3 describes the *PuDZ* SoS and the individual systems, while Section 4 describes cross-system adaptation scenarios. Section 5 reports results of our preliminary evaluation, Section 6 presents related work, and finally, Section 7 draws conclusions and discusses future work.

2 A Distributed SoS Architecture

The MAPE-K feedback loop has been established as the default reference model for self-adaptive systems, consisting of four main phases: **M**onitoring, where information is collected, **A**nalysis to determine if adaptation is required, **P**lanning where corresponding actions are planned, and finally, **E**xecution in which the proposed plans are enacted [3].

To address different characteristics and architectures of managed systems, prior work in the areas of self-adaptive systems has explored a variety of patterns for decentralized control [32], including cases in which self-adaptive systems are deployed across different physical nodes connected by a network. These patterns can be applied to a variety of tasks, depending on the operational needs of the system. For example, the coordination control pattern allows for greater scalability, as each phase of the MAPE-K control loop acts as an individual decentralized node that executes inter-phase interactions. This can offer improved flexibility for a distributed SoS, but can create issues in preserving sequential consistency or can cause message pile-up.

In addition, Weyns and Andersson explicitly discussed MAPE-K SoS patterns [31] and identified a specific challenge related to the decentralization of components that can carry interdependencies. Their proposed SoS patterns are characterized by local, collaborative, and regional approaches to self-adaptation. Local adaptations rely on an entirely decentralized adaptation architecture. The individual MAPE-K loops do not interact with each other, but only the respective systems do. With collaborative adaptations, not only do the managed systems themselves interact with each other but so do the feedback loops themselves. Finally, in the third pattern regional monitoring/local adaptations, MAPE-K loops of one system can collect additional runtime data from other parts

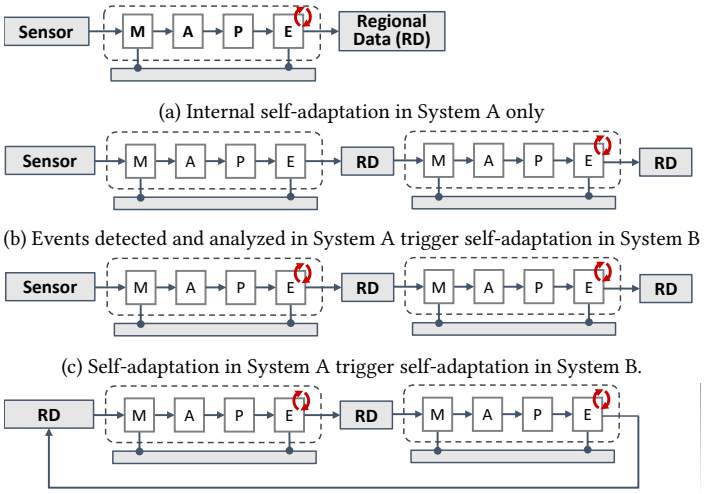

Figure 3: Patterns of loosely coordinated self-adaptation. Both sensors and Regional Data (RD) serve as input for subsequent decision-making and adaptation.

of the SOS. This additional data is fed into the feedback loop and used for the subsequent planning and adaptation process.

For our *Pop-up Drone Zone*, we face similar challenges where (1) individual systems largely need to self-adapt independently based on the input they receive from local sensors, services, or other parts of the *PuDZ* SoS; and (2) information about changes in the environment, or adaptations they have made need to be propagated to other systems in the SoS as these changes could in turn trigger subsequent adaptations in those systems.

2.1 Proposed SoS-PuDZ Architecture

To address these requirements, we have created and adopted our own MAPE-K SoS architectural pattern that borrows concepts from both decentralized SoS patterns and the IBM architectural blueprint for autonomic computing [10]. Each MAPE-K managed system implements the MAPE-K loop depicted in Fig. 2, with a global *SoS Policy Manager*, responsible for ensuring information integrity (cf. Section 2.3). The Monitor maintains two interfaces: a mechanism for receiving information from neighboring systems, and a mechanism for monitoring a locally managed system or resource. During the monitoring phase of the model, the feedback loop receives regional inputs via the *Regional Monitor* (M^R) and local sensor data updates from its own managed system via the *Local Monitor* (M^L). The primary difference is the degree of trust placed in the input data. For example, while local data from sensors should be validated to ensure it is within expected bounds; its sources and their subsequent reliability are relatively well known and the system places appropriate degrees of trust in the data. On the other hand, regional data requires higher degrees of validation to check its provenance and to decide whether it can be safely utilized locally. Data from both monitors are collected and filtered during runtime, and the analysis phase checks whether internal or external adaptation is required, guided by the local knowledge base. The planning phase then determines whether to adapt, and if necessary identifies a new configuration based on the policy/knowledge base, publishing regional outputs to

the message bus and local actuator outputs to the local managed system.

2.2 System Adaptation Patterns in an SoS

MAPE-K managed systems within an SoS adapt in several different ways as depicted in the four adaptation patterns shown in Fig. 3. In the first scenario (cf. Fig. 3a), `System A` receives data from its internal sensors, and after analysis and planning, it adapts accordingly. In the second scenario (cf. Fig. 3b) `System A` does not adapt, however, it still publishes information to a regional data topic to which `System B` is subscribed to. While no adaptations have been performed in `System A`, the regional monitoring data received from `System A` ultimately leads to B's self-adaptation. A similar example is shown in Fig. 3c, however, with the major difference that in this case, both `System A` and `System B` adapt. `System A` receives local sensor data and adapts in response. It then publishes regional data which causes B to adapt. Finally, for the last case, (cf. Fig. 3d) a dependency cycle is introduced between two or more self-adaptive systems, meaning that adaptations in one system trigger adaptions in another system, and vice versa, thereby forming a cycle. Cycles are broken when the execution plan of one system no longer publishes data that triggers adaptation.

2.3 The SoS as a Managed System

Weyns' paper on SoS challenges suggested that a management layer might be needed for managing quality requirements [31]. We include an "SoS Policy Manager" in our architectural solution to achieve quality requirements and also as an orchestrator responsible for (1) vetting each system that requests entry to the SoS, (2) monitoring system communication across the SoS to ensure proper messaging protocols are followed, and (3) ensuring rules such as "one and only one ATC must be active at all times" are enforced. The SoS level policy manager receives input from its *Monitors* via regional data, *analyzes* inputs for compliance against its internal knowledge base, and finally *plans* and *executes* updates to its current management protocols. Notably, the SoS-level Policy Manager manages the SoS itself rather than a CPS, and has no direct control over the systems apart from notifying them when protocols are not followed, and potentially evicting them from the system. This differs from centralized approaches in which a single hub is responsible for maintaining SoS functionality [11].

3 Systems in the PuDZ System of Systems

Before discussing cross-system adaptation, we describe the constituent systems, which serve as the fundamental building blocks of the *PuDZ* SoS. For purposes of this paper, we focus on four primary systems, each responsible for a critical part of sUAS coordination, management, and planning (cf. Fig. 1). These systems are (1) Air Traffic Control, which is responsible for managing and leasing airspace to support safe operations of sUAS; (2) A Digital Shadow, modeling information about the real world, such as weather and terrain data; (3) the individual sUAS that operate in the *PuDZ*, and finally; (4) the SoS Policy Manager responsible for ensuring that cross-SoS policies are followed. Each of the components subscribes to the bus to receive regional inputs and publishes regional outputs to it. Regional inputs and outputs for each system are summarized in Fig. 4 and described below.

Air Traffic Control (ATC) systems are responsible for authorizing sUAS' flight requests including permission to take off and/or to fly a specific route. The ATC is a vital system when multiple sUAS operate in densely populated areas, share the airspace with airplanes or helicopters, or generally operate (semi-)autonomously as part of a mission. However, as depicted in Fig. 1, the *PuDZ* includes two alternate services responsible for flight authorization – the NASA-controlled sUAS Traffic Management System (UTM) [1], and the *PuDZ Air-Leaser* which we have implemented as part of the *PuDZ* system for use in more congested airspaces. By SoS policy, one, and only one ATC system must be active at all times to avoid conflicting clearances for sUAS that could result in potentially dangerous situations or even collisions. The UTM is an emergent technology that has been field-tested in the US National Airspace. sUAS entering UTM-controlled areas will request flight authorization directly from the UTM using its own services. Furthermore, there are plans to extend the UTM to support local ATC managers responsible for designated regions of airspace such as an urban area or a *PuDZ*. This would allow an airspace to transition from UTM to local control and back based on the availability of a local ATC and current congestion levels, requiring a clear hand-over process. The NASA UTM is not currently a self-adapting system [1].

In contrast, the local air-leasing system that we have developed is self-adapting. It accepts flight requests from an sUAS, and converts the requested flight route (from A to B) into an air-tunnel (represented by a long, potentially curved cylinder). When a flight authorization request is received, the Air-Leaser generates the air-tunnel and checks that the minimum separation distance from currently authorized tunnels is greater or equal to the established buffer zone. If this is the case, then the Air-Leaser authorizes the flight, and otherwise, denies it. The Air-Leaser ensures that an sUAS in full compliance to its policies will always have uniquely assigned appropriately separated airspace while in flight. In addition, it can perform smart flight assignments by offering alternate routes if a current route is unavailable, and can switch modes to grid-based routing when the airspace becomes overly congested.

The Air-Leaser maintains a model of the airspace and currently allocated air-tunnels. It accepts flight authorization requests via regional inputs, monitors for changes reported by other systems, and publishes its status when activated or deactivated during a UTM handover. Internally, its adaptations focus on its internal flight planning methods such as direct or grid-based routing.

The Environmental Digital Shadow System: The concept of Digital Twins follows the idea of building a digital model of the real, physical world and is often used in Industry 4.0 and Cyber-Physical Production Systems [21], as well as Smart Cities [24]. It can be seen as a system/entity on its own, linked to the physical system [15].

The Digital Shadow, in turn, relies on *one-way data flow* from the physical world to its digital counterpart, meaning that changes in the "real world" can be observed and modeled in the virtual environment [15, 4]. With *PuDZ*, we leverage this idea, introduc-

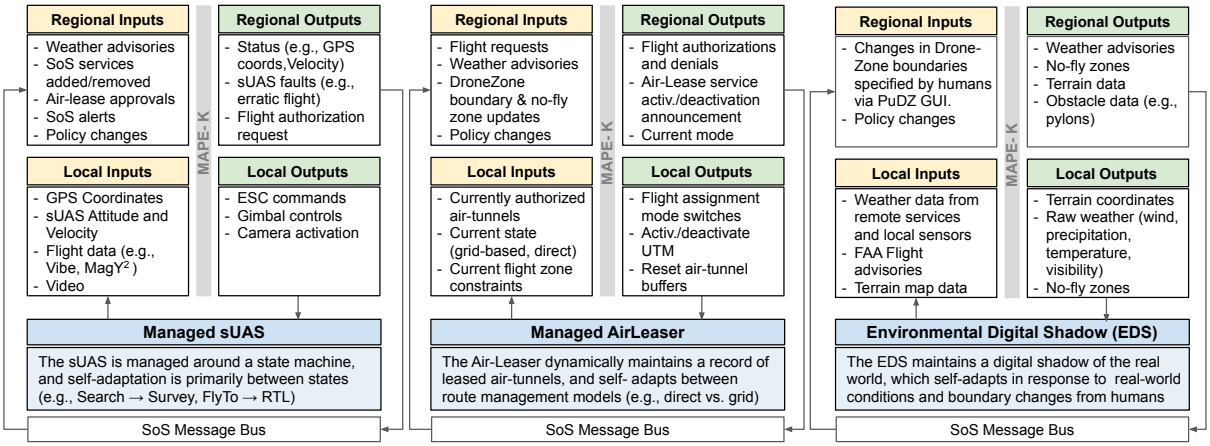

Figure 4: Local and Regional inputs and outputs, and self-adaptation capabilities summarized for three PuDZ SoS Systems.

ing an Environmental Digital Shadow System (EDS). EDS utilizes several external services to acquire awareness of its enclosed environment and applicable regulatory constraints. For example, it calls upon (1) weather services to identify, for example, temperature, wind at various altitudes, and visibility, (2) mapping services to build a model of the terrain and to identify tall buildings, (3) airspace services to identify airspace restrictions related to lighting, no-fly zones, and any other information, and (4) a database listing standard flight regulations (e.g., FAA and local by-laws) and points of contact, such as local air-traffic-control towers. The EDS implements an internal MAPE-K loop to monitor for changes in the environment, analyze the changes, and ultimately plan and execute actions to update its internal model of the environment. For example, the EDS can be in either *passive* or *active* weather reporting modes. When weather is stable without extreme conditions, it simply maintains weather models internally (passive mode) whilst monitoring the weather for changes. When weather conditions indicate the need to switch to active mode, it plans and then executes a mode change that includes a weather-appropriate broadcasting strategy over regional data.

Individual sUAS: In general, a *PuDZ* supports an *open*, *closed*, or hybrid environment. In a closed setting, all sUAS are managed directly by the *PuDZ* meaning that the SoS includes specialized services for dispatching, managing, monitoring, and coordinating sUAS; whereas in an open setting, sUAS are managed externally by third-party operators. Regardless of their management type, all sUAS must (1) communicate with the *PuDZ* manager over the established radio network or via a general web service, (2) periodically report GPS coordinates, altitude, velocity, and remaining flight time, (3) fully comply to the ATC air-leasing protocols and (4) report any detected problems using a standard protocol when feasible.

Each sUAS is equipped with onboard computation capabilities and is capable of self-adaptation using an onboard MAPE-K loop. At the start of each mission, each sUAS receives a mission specification that it uses to initialize and internally configure its mission-specific state machine. Throughout the mission, it receives mission-related updates via a system-wide MQTT message broker and utilizes its onboard sensors to monitor the environment (e.g., for geolocation, computer vision, and flight-controller status). The sUAS continually analyzes this data and based on policies and rules embedded in its internal knowledge base, it plans and ultimately executes self-adaptations, e.g., flying slower due to poor visibility, or switching from 'search' to 'tracking' modes upon detecting a person.

The SoS Policy Manager has an entirely different role compared to the other constituent systems. It takes a supervisory role with its main responsibility to check that SoS policies are followed by all other systems. While its managing system incorporates the capabilities of MAPE-K, there are two major differences: it lacks local inputs and has no actual execution capabilities with respect to its "managed" system beyond notifying when policies are not followed and blocking the system's engagement in the SoS. This can be attributed to its primary purpose, its responsibility for the overall SoS policies, rather than individual systems, or (physical) components. Examples of its responsibilities include (1) ensuring that one, and only one, ATC system is active at all times, (2) ensuring that all active systems comply with communication protocols, and (3) adjusting required security protocols when operating in high-risk environments. Its regional inputs, therefore, contain information on services that are activated or deactivated at runtime, and topics from the SoS Message Bus related to the defined policies. These, and any other established SoS policies are monitored, and several measures can be taken when a policy is violated.

While the "managed" system is the SoS itself, the Policy Manager may also internally configure its policies based on the system's current state. For example, it may update policies when a non-managed sUAS enters the *PuDZ*, or in conjunction with the SoS security service, it might activate new policies such as increased levels of encryption, associated with a `HIGH SECURITY` state if the SoS is under attack or operating in a higher-risk urban area. Such changes in policies, announced as regional data to the SoS, are likely to trigger internal self-adaptation of individual systems such as sUAS or the Air-Leaser.

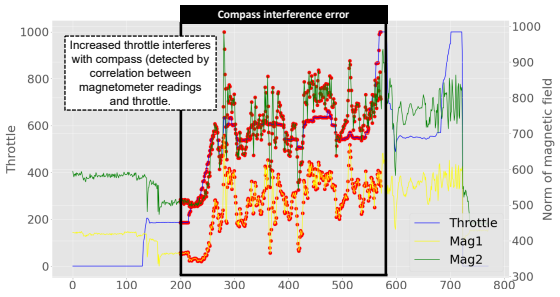

Figure 5: Compass interference is detected in real-time [12]

4 Loosely-coupled, coordinated Adaptation

We now discuss cross-SoS coordination and show how our approach supports a variety of different adaptation scenarios. The SoS-PuDZ architecture connects regional data produced by the (E)xecution of one MAPE-K managing system and consumed by the regional (M)onitor of a different MAPE-K managing system using the publish-subscribe architecture. This allows each self-adaptive system to publish and subscribe to relevant topics, empowering software engineers to design, implement, and deploy each individual system independently. New services joining the SoS must identify topics published by other services that are of interest to them – potentially supporting their own self-adaptive behavior, and to decide what topics the new system should publish in order to comply with SoS policies and/or be useful to other systems.

4.1 Coordinated Self-Adaptation of SoS

Scenario 1 – sUAS Fault Triggered: We illustrate cross-system self-adaptation with an example in which the sUAS system and Air-Leaser both self-adapt following Pattern C in Fig. 3, where self-adaptation in one system triggers self-adaptation in another. We describe the scenario in terms of the trigger and its resulting adaptation in the sUAS system, the regional data produced by the sUAS system and consumed by the Air-Leaser.

– *Initial Trigger:* To increase safety of flight operations, the onboard autopilot software *monitors* multi-variate time series data produced by the flight controller throughout the flight, and analyzes the data to detect common flight faults such as vibration, loss-of-signal, and compass interference. It does this by subscribing to the sUAS' inbuilt uORB, an asynchronous messaging API used for inter-process communication in the PX4 flight controller [22]. The data is collected as a continual multi-variate time series, and the onboard anomaly detection module *analyzes* the data in close to real-time to detect common types of runtime anomalies at runtime [12]. Detection can be performed either using a heuristic approach or a deep-learning solution such as LSTM (Long short-term memory) [9].

In our example, as depicted in Fig. 5, a "compass interference" error is detected when a correlation exists between magnetometer readings and increased throttle. Specifically, the onboard anomaly detector monitors two attributes of l2-norm (i.e., $\sqrt{SumOfMags}$, where $SumOfMags = MagX^2 + MagY^2 + MagZ^2$) and throttle (CTUN.ThO). This is a serious flight problem, as compass interference can prevent the sUAS from finding the right bearings to fly to a waypoint, and can result in a fly-away situation.

– *sUAS Autopilot Self-Adaptation:* In our example, the compass interference is detected, and, therefore, the sUAS *plans* a mitigation that could allow it to return home safely. Instead of completing its mission, it generates a new flight route that returns it directly to its home location at reduced velocity in order to reduce electrical interference caused by high throttle values. The sUAS *executes* its new plan by requesting a flight route from the Air-Leaser system and flying home once the route is approved.

– *Regional Data:* As part of executing its self-adaptation, the sUAS publishes the following announcement: `\drone-failure\BLUE\ErraticFlight` as regional data. The Air-Leaser subscribes to (monitors) all `\drone-failure` messages and, therefore, becomes aware that BLUE is experiencing a problem that might cause erratic flight.

– *Self-Adaptation of Air-Leaser:* The Air-Leaser analyses the current positions of all sUAS, and then plans and executes an adaptation strategy which causes it to switch from `NORMAL` to `EMERGENCY` operating mode. This includes adapting its internal safety policies to create a larger buffer around BLUE, temporarily halting new flight assignments, directing all flights to avoid airspace in the vicinity of BLUE, and prioritizing the monitoring of status messages from BLUE. Assuming a successful operation, when sUAS-BLUE completes its flight and has landed at its home location, the respective information is published, and the Air-Leaser can yet again adapt its internal behavior switching back to its normal operation mode.

4.2 Cross-System Adaptation Scenarios

We describe two additional scenarios to provide further examples of loosely coordinated self-adaptation across PuDZ.

Scenario 2 – Weather triggered: The EDS continually *monitors* the current environmental conditions to maintain an internal weather model. If a sudden rain storm approaches, the EDS *analyzes* the data it receives from services and sensors to determine the impact upon the PuDZ, *plans* a corresponding adaptation of the digital shadow model (e.g., by transitioning to "stormy conditions" state), and *executes* the plan by updating the model and broadcasting a weather alert via its regional outputs. The Air-Leaser is subscribed to weather-related announcements, and, therefore, receives the alert and self-adapts to accommodate the encroaching weather system, for example by increasing separation distances between sUAS. Similarly, individual sUAS could also receive weather alerts and make independent self-adaptation decisions according to their own capabilities. These could include landing to wait out the storm, or directly querying the EDS for additional data in order to plan a flight path that avoids extreme wind gusts at higher altitudes.

Scenario 3 – Air-Leaser activated: Our last example focuses on the role of the SoS Policy Manager which manages policies such as "One and only one ATC shall be active at all times". The Policy Manager checks that the Air-Leaser complies with UTM routines for hand-overs to local ATCs, and blocks air-leasing

requests until the air-leaser publishes a hand-over certificate (i.e., `\ATC\airleaser\[UTM Certificate ID]`). The SoS Policy Manager's *monitor* receives this regional announcement, and *analyzes* its authenticity by querying UTM directly for confirmation. If the UTM confirms the hand-over, the Policy Manager publishes `\ATC\airleaser \activated`; however, if UTM refutes the hand-over, the SoS Policy Manager *plans* a response and *executes* a solution such as blocking the ATC which has claimed authorization to manage ATC. Finally, if no response is received from UTM, the Policy Manager temporarily blocks the Air-Leaser, intercepts all Air-Leaser messages, and responds with `\ATC\DOWN`. When and if the Policy Manager confirms that the hand-over has been successful it publishes `\ATC\UTM\activated`.

5 Preliminary Evaluation in the Field

We now address our primary research question "To what extent does the proposed architecture support an extensive set of cross-system self-adaptation scenarios in the PuDZ SoS?" through a series of simulations and field tests.

5.1 Test and Validation Platform

All tests were run using our existing multi-sUAS DroneResponse platform [25, 26, 2, 8] and four Hexacopters equipped with PX4-compatible flight controllers, and Jetson NX onboard computation units. All systems were connected over a mesh radio with access to the internet via a multi-provider mobile access point. The message broker was implemented using Mosquitto MQTT. Frequent status messages were sent at Quality of Service (QoS) level 0 (i.e., delivered at most once) whilst regional data messages were sent at QoS level 2 (i.e., delivered exactly once). The Air-Leaser was implemented in Python, as a micro-service. It transforms each flight path into a 3D line segment that contains a self-adaptive radius – a protective extension surrounding the requested line segment that replicates an "air tunnel". Flights are authorized if a requested route maintains minimum separation distance from all other routes.

We used two different Ground Control Stations (GCS) for our experiments. QGroundControl [23], is an open-source centrally controlled system that communicates directly with sUAS flight controllers via the mavlink protocol. DroneResponse's GCS is extremely lightweight. It transmits mission data as JSON specifications to the sUAS; however, flight control and self-adaptation are all managed by an onboard autonomous pilot system. The autonomous pilot was developed using Mavros, a ROS-based package that provides communication drivers with the PX4 flight controller. It uses the SMACH hierarchical state machine library to transition between mission-related states (e.g., from searching to tracking) when an event occurs such as detection of a victim, and to adapt to failsafe states when error conditions occur such as a geofence breach.

The EDS system retrieves data from two external services including MapBox for generating a 3D map, and the USA Federal Aviation Authority's LAANC (Low Altitude Authorization and Notification Capability) for modeling airspace classes and no-fly zones. Self-adaptation is limited to updates in temporary no-fly zones. Weather-related services have not yet been integrated.

5.2 Physical and Simulation Test Environments

During the development of our sUAS platform (SoS) in which it operates, we executed many hundreds of tests. For purposes of this evaluation, we selected a subset of those tests that demonstrated self-adaptation across one or more systems. Individual tests were first run in the simulator using Gazebo [19] to validate the functionality and to address basic bugs (e.g., jerky flights, state transition failures), and then run outside with physical sUAS where new bugs often emerged. Table 1 lists the selected tests, categorizing each one by its respective coordination pattern (cf. Fig. 3), identifying the triggering and coordinating systems, and specifying whether it was conducted as a physical field test or a simulation.

T1 (Pattern A): During each flight, each sUAS self-adapts across its mission-specific states. Most state transitions occur when simple events are detected (e.g., a waypoint is reached, a victim is detected, or a geofence is breached).

T2 (Pattern A): The Computer Vision (CV) pipeline was activated. The sUAS was assigned a route to fly in `SEARCH` mode. Its onboard CV detected a person ("found" event) and the sUAS self-adapted into `SURVEY` mode.

T3 (Pattern B) sUAS-A detects the person, announces "person found". sUAS-B self-adapts from `SEARCH` to `SURVEY` mode. Multiple-sUAS coordination has been tested in simulations, and mimicked in the physical world by sending an MQTT message to sUAS-B with the coordinates of the detected person.

T4 (Pattern A): The GCS was deactivated during flight. The sUAS failed to receive a heartbeat and transitioned to `FAILURE` mode, where it autonomously adapted between hover, RTL, and mission mode with the goal of completing the mission safely.

T5 (Pattern C): This test simulated the EDS adaptation, message passing between EDS and Air-Leaser via MQTT, and subsequent adaptation of the Air-Leaser to increase spacing between flights. The Air-Leaser's performance was validated with physical sUAS under varying buffer sizes.

T6 (Pattern C): Air-Leaser adaptions from direct routing to a grid-based pattern were evaluated in simulation at various levels of congestion to demonstrate the benefits of adapting. Simulations were run in a low-fidelity environment.

T7 (Pattern C): We developed a Runtime Monitor to collect and analyze time series data from the PX4 flight controller. It was validated for detecting runtime vibration anomalies in Gazebo with transitions into failsafe states (e.g., `HOVER`, `LAND`), but can also run on a physical sUAS.

T8 (Pattern C): We validated that the PuDZ boundaries could be resized, and that the sUAS and Air-Leaser self-adapt to work within the expanded region.

T9 (Pattern C): Throughout the course of our tests, ill-formed mission specifications were consistently rejected. Originally, the specifications were analyzed by each sUAS; however, we created and validated a module that will be run by the SoS Policy Manager. This test is valid only for "closed" PuDZ systems.

T10 (Pattern C): Due to human error, an sUAS violated the geofence without a failsafe action defined. It flew off-course and ascended above the legal limit of 400ft AGL. We retroactively created a

Table 1: Test Cases Executed. ●=Triggering system, ○=Collaborating system, (PTN A,B,C)=Patterns per Fig. 3, Mode: (S)=Simulation only, (P)=Physical field tests + simulations, (R)=Retroactively fixed but not tested. Systems: AL=Air-Leaser, SoS Pol= Policy Manager, Run.Mon = Runtime Monitor

T#	Description of Self-Adaptive Events	PTN A-C	Test Mode	Triggering System	AL	EDS	sUAS A	sUAS B	GCS	SoS Pol.	Run. Mon
T1	Mission state transition during normal oper.	A	P	sUAS (Auton. Pilot)			●				
T2	sUAS-A detects and tracks a person	A	P	sUAS (Comp. Vision)			●				
T3	sUAS-A detects and sUAS-B tracks person	B	P	sUAS (Comp. Vision)			●	○			
T4	sUAS misses heartbeat from GCS	A	P	sUAS (Monitor)			●				
T5	EDS and Air-Leaser adapt due to high winds	C	S/P	EDS (Weather)	○	●	○	○			
T6	Air-leaser adapts to grid layout	C	S	Air-Leaser	●		○	○		○	
T7	Compass interference onboard sUAS	C	S	sUAS (Analytics)	○		●				
T8	Human requests extended PuDZ Boundaries	C	S	EDS (boundaries)	○	●	○	○			
T9	Ill-formed mission detected. Mission aborted	C	P	SoS Policy Manager			○	○	○	●	
T10	sUAS hits geofence & flies off course	C	R	Runtime Monitor	○		○				●

Test data available at: https://github.com/SAREC-Lab/PuDZ

runtime monitor to detect excessive altitudes and publish an alert triggering self-adaptive responses.

5.3 Summary of Test Results & Discussion

These tests demonstrate the viability of the SoS PuDZ architecture. DroneResponse requirements were elicited as part of an extensive participatory design process with emergency responders [2], and each identified self-adaptation scenario was readily specified in terms of the regional inputs and outputs of the systems in which the scenario was enacted. We not only demonstrated single-system self-adaptation, but the diverse simulations and physical field tests validated six diverse scenarios representing loosely coupled cross-system self-adaptation (i.e., T5–T10). The adaptation scenarios were triggered by five unique systems, each of which published regional output data that was ultimately consumed and acted upon by other systems. Our Scenarios represented three of our four patterns from Fig. 3, namely Patterns A, B, and C. Our SoS-PuDZ solution achieves loose coupling primarily through the use of the publish-subscribe architectural pattern; however, this can potentially cause bottlenecks during message surges. We partially mitigated this weakness by utilizing QoS level 0 for status messages where the loss of occasional messages is not a problem, and level 2 for other messages, such as those announcing weather alerts, or requesting flight authorization, as these require delivery guarantees but do not have hard real-time constraints. We did not observe any latency issues between services on the ground and sUAS.

5.4 Threats to Validity

Our work is subject to two primary threats to validity. First, while we have performed hundreds of hours of functional tests in the field, we have not yet validated the system in its entirety. However, the physical tests that we have conducted focused on highly complex parts of the system such as using CV onboard the sUAS, geolocating the detected person, and circling the person for surveillance purposes, all using MQTT and the mesh radio. Taken together, the tests provide strong evidence that the architecture is suitable for loosely-coupled, cross-system independent self-adaptation.

Second, as described, SoS are characterized by operational and managerial independence [18]; however, our system represents a mix of systems. Several are entirely independent (e.g., NASA UTM), while others contain multiple subsystems. For example, the EDS represents a system of independently managed services (e.g., LAANCs, MapBox, Weather services), and the core sUAS system is built around the independently managed PX4 OSS. However, with respect to the sUAS, we also ran tests using the independently managed QGroundControl to demonstrate PuDZ SoS as an open system. Furthermore, the core SoS services support both open and closed models and are agnostic to the inner design of each system, the only constraint being the ability to publish and subscribe to the PuDZ message broker following standard message protocols.

6 Related Work

The paper has already discussed related work by Weyns *et al.* on patterns of MAPE-K applications [32] and its application to SoS scenarios [31]. We, therefore, focus on other areas of related work.

Self-Adaptive Systems: Jahan *et al.* [14] presented a framework for dynamically maintaining functional and security concerns in autonomous systems. Like our approach, they ensured coordination between multiple MAPE-K feedback control loops, including a traditional MAPE-K loop augmented by a MAPE-SAC loop for security-related assurance cases. Vromant *et al.* [29] used intra-loop and inter-loop coordination of multiple MAPE loops to jointly perform adaptations.

Systems of Systems: As part of our own previous work in the domain of SoS, we have created ReMinds, a Runtime Monitoring Framework and Monitoring Model for Systems of Systems [27, 28]. However ReMinds does not implement a full MAPE-K control loop, and relies upon manual changes and adaptations once violations of specified requirements are detected. Maia *et al.* [17] modeled normal and exception case scenarios for SoS adaptations, and transformed them into a formal specification to ensure that global requirements always hold. In contrast, we focused on system-level coordination and self-adaptation rather than on detailed models of scenarios. Bucchiarone *et al.* [7] presented a framework for adaptations in "Service-based Collective Adaptive Systems", which like our approach, emphasized the need for simultaneous self-adaption of collaborating systems; however, each system

needed to be modeled as a domain object, which enacted its own MAPE-K loop at runtime and relied upon collective planning.

Multi-Agent and UAV Systems: In the domain of UAV systems, Bozhinoski *et al.* [5] used a collective adaptation engine to manage adaptations of UAV-based systems. Like our approach, they relied on multiple MAPE-K loops to perform collective adaptation at runtime. However, while their approach supported the adaptation of multiple UAVs, it did not integrate diverse external services providing data such as weather or terrain information in adaptation planning and execution. Braberman *et al.* [6] presented MORPH, a reference architecture for self-adaptation using the MAPE-K loop to support mission planning for UAVs. It included layers for goal management, strategy management, and strategy. While these approaches facilitate both local and collaborative self-adaptation, with *PuDZ* we emphasize the SoS characteristics, going beyond coordination and control of multiple sUAS, and considering environmental aspects (such as weather), as well as external services (e.g., no-fly zones) that affect collaborative adaptation decisions.

7 Conclusion

In this paper, we have presented an architecture for supporting self-adaptation across loosely coupled services in an SoS leveraging and augmenting ideas from earlier work on self-adaptation in SoS. Our approach places the following requirements upon systems joining the SoS. Self-adaptive systems within the SoS should be capable of dynamically reconfiguring and/or adapting themselves utilizing their own local sensor data as well as regional data to trigger the adaptation. Furthermore, they are expected to be 'good citizens' and publish regional outputs that might be useful and relevant to other systems. The SoS Policy Manager hereby serves as a higher-level authority for ensuring compliance with global SoS policies. Our simulations and field tests have demonstrated the overall viability of the SoS PuDZ architecture to support self-adaptation of diverse constituent systems. Future work will focus on more holistic, longer-duration, tests in the field to validate additional aspects of the PuDZ solution, such as more complex adaptations, weather-related updates to the EDS in the real world, and additional security and runtime monitoring services.

References

[1] N. Aeronautics and S. A. (NASA). Unmanned Aircraft System (UAS) Traffic Management (UTM). `https://utm.arc.nasa.gov`, 2023. [Accessed 01-12-2023].

[2] A. Agrawal, S. J. Abraham, B. Burger, C. Christine, L. Fraser, J. M. Hoeksema, S. Hwang, E. Travnik, S. Kumar, W. J. Scheirer, J. Cleland-Huang, M. Vierhauser, R. Bauer, and S. Cox. The next generation of human-drone partnerships: Co-designing an emergency response system. In *Proc. of CHI Conf. on Human Factors in Computing Systems*, pages 1–13, New York, 2020. ACM.

[3] P. Arcaini, E. Riccobene, and P. Scandurra. Modeling and analyzing MAPE-K feedback loops for self-adaptation. In *10th Int. Symp. on Software Engineering for Adaptive and Self-Managing Systems*, pages 13–23. IEEE, 2015.

[4] T. Bergs, S. Gierlings, T. Auerbach, A. Klink, D. Schraknepper, and T. Augspurger. The concept of digital twin and digital shadow in manufacturing. *Procedia CIRP*, 101:81–84, 2021.

[5] D. Bozhinoski, A. Bucchiarone, I. Malavolta, A. Marconi, and P. Pelliccione. Leveraging collective run-time adaptation for UAV-based systems. In *Proc. of the 42th Euromicro Conf. on Software Engineering and Advanced Applications*, pages 214–221. IEEE, 2016.

[6] V. Braberman, N. D'Ippolito, J. Kramer, D. Sykes, and S. Uchitel. Morph: A reference architecture for configuration and behaviour self-adaptation. In *Proc. of 1st Int'l WS on Control Theory for Software Engineering*, pages 9–16, New York, 2015. ACM.

[7] A. Bucchiarone, M. De Sanctis, and A. Marconi. Decentralized dynamic adaptation for service-based collective adaptive systems. In *Proc. of the Service-Oriented Computing–ICSOC 2016 Workshops*, pages 5–20. Springer, 2017.

[8] J. Cleland-Huang, A. Agrawal, M. Vierhauser, M. Murphy, and M. Prieto. Extending MAPE-K to support human-machine teaming. In *Proc. of the International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 120–131. ACM/IEEE, 2022.

[9] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[10] IBM. An architectural blueprint for autonomic computing. *IBM White Paper*, 31(2006):1–6, 2006.

[11] C. Ingram, R. Payne, S. Perry, J. Holt, F. O. Hansen, and L. D. Couto. Modelling patterns for systems of systems architectures. In *Proc. of the 2014 IEEE International Systems Conference*, pages 146–153, 2014.

[12] M. N. A. Islam, Y. Ma, P. A. Granadeno, N. V. Chawla, and J. Cleland-Huang. RESAM: requirements elicitation and specification for deep-learning anomaly models with applications to UAV flight controllers. In *Proc. of the 30th IEEE International Requirements Engineering Conference*, pages 153–165. IEEE, 2022.

[13] ISO/IEC . ISO/IEC/IEEE 21839:2019 Systems and software engineering — System of systems (SoS) considerations in life cycle stages of a system. `https://www.iso.org/standard/71955.html`, 2019. [Last accessed 01-01-2023].

[14] S. Jahan, I. Riley, C. Walter, R. F. Gamble, M. Pasco, P. K. McKinley, and B. H. C. Cheng. Mape-k/mape-sac: An interaction framework for adaptive systems with security assurance cases. *Future Generation Computer Systems*, 109:197–209, 2020.

[15] W. Kritzinger, M. Karner, G. Traar, J. Henjes, and W. Sihn. Digital twin in manufacturing: A categorical literature review and classification. *IFAC-PapersOnLine*, 51(11):1016–1022, 2018.

[16] S. Ljungblad, Y. Man, M. A. Baytaş, M. Gamboa, M. Obaid, and M. Fjeld. What matters in professional drone pilots' practice? an interview study to understand the complexity of their work and inform human-drone interaction research. In *Proc. of the 2021 CHI Conference on Human Factors in Computing Systems*, New York, NY, USA, 2021. Association for Computing Machinery.

[17] P. H. Maia, L. Vieira, M. Chagas, Y. Yu, A. Zisman, and B. Nuseibeh. Cautious adaptation of defiant components. In *Proc. of the 34th IEEE/ACM International Conference on Automated Software Engineering*, pages 974–985. IEEE, 2019.

[18] M. W. Maier. Architecting principles for systems-of-systems. *Systems Engineering: The Journal of the International Council on Systems Engineering*, 1(4):267–284, 1998.

[19] Open Robotics. Gazebo. https://gazebosim.org/home, 2023. [Last accessed 01-01-2023].

[20] P. Petrides, P. Kolios, C. Kyrkou, T. Theocharides, and C. Panayiotou. Disaster prevention and emergency response using unmanned aerial systems. In *Smart cities in the Mediterranean*, pages 379–403. Springer, 2017.

[21] F. Pires, A. Cachada, J. Barbosa, A. P. Moreira, and P. Leitão. Digital twin in industry 4.0: Technologies, applications and challenges. In *Proc. of the 17th Int'l Conf. on Industrial Informatics*, volume 1, pages 721–726. IEEE, 2019.

[22] PX4. Open Source Flight Controller. `https://px4.io`, 2021. [Accessed 01-12-2023].

[23] QGroundControl. Ground Control Station. `http://qgroundcontrol.com`, 2021. [Last accessed 01-11-2021].

[24] T. Ruohomäki, E. Airaksinen, P. Huuska, O. Kesäniemi, M. Martikka, and J. Suomisto. Smart city platform enabling digital twin. In *Proc. of the 2018 International Conference on Intelligent Systems*, pages 155–161. IEEE, 2018.

[25] M. Vierhauser, S. Bayley, J. Wyngaard, W. Xiong, J. Cheng, J. Huseman, R. R. Lutz, and J. Cleland-Huang. Interlocking safety cases for unmanned autonomous systems in shared airspaces. *IEEE Trans. Software Eng.*, 47(5):899–918, 2021.

[26] M. Vierhauser, M. N. A. Islam, A. Agrawal, J. Cleland-Huang, and J. Mason. Hazard analysis for human-on-the-loop interactions in suas systems. In *Proc. of 29th ACM Joint Meeting on European Software Engineering Conf. and Symp. on the Foundations of Software Engineering*, pages 8–19, New York, 2021. ACM.

[27] M. Vierhauser, R. Rabiser, P. Grünbacher, C. Danner, and S. Wallner. Evolving systems of systems: Industrial challenges and research perspectives. In *Proc. of the First Int'l WS on Software Engineering for Systems-of-Systems*, pages 1–4, 2013.

[28] M. Vierhauser, R. Rabiser, P. Grünbacher, K. Seyerlehner, S. Wallner, and H. Zeisel. Reminds: A flexible runtime monitoring framework for systems of systems. *Journal of Systems and Software*, 112:123–136, 2016.

[29] P. Vromant, D. Weyns, S. Malek, and J. Andersson. On interacting control loops in self-adaptive systems. In *Proc. of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 202–207, 2011.

[30] M. Weißbach, P. Chrszon, T. Springer, and A. Schill. Decentrally coordinated execution of adaptations in distributed self-adaptive software systems. In *Proc. of 11th International Conference on Self-Adaptive and Self-Organizing Systems*, pages 111–120. IEEE, 2017.

[31] D. Weyns and J. Andersson. On the challenges of self-adaptation in systems of systems. In *Proc. of the First International Workshop on Software Engineering for Systems-of-Systems*, pages 47–51, 2013.

[32] D. Weyns, B. Schmerl, V. Grassi, S. Malek, R. Mirandola, C. Prehofer, J. Wuttke, J. Andersson, H. Giese, and K. M. Göschka. *On Patterns for Decentralized Control in Self-Adaptive Systems*, pages 76–107. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.