Security and Autonomic Management in System of Systems

Silia Maksuti, Mario Zsilak, Markus Tauber and Jerker Delsing

Abstract—**A system of systems integrates systems that function independently but are networked together for a period of time to achieve a higher goal. These systems evolve over time and have emergent properties. Therefore, even with security controls in place, it is difficult to maintain a required level of security for the system of systems as a whole because uncertainties may arise at runtime. Uncertainties can occur from internal factors, such as malfunctions of a system, or from external factors, such as malicious attacks. Self-adaptation is an approach that allows a system to adapt in the face of such uncertainties without human intervention. This work outlines the progress made towards security mitigation in system of systems using a generic autonomic management system to assist engineers in developing self-adaptive systems. The manuscript describes the proposed system design, its implementation as part of the Eclipse Arrowhead framework, and its functionality in a smart agriculture use case. The system is designed and implemented in such a way that it can be reused and extended for a variety of use cases without requiring major changes.**

Index Terms—**System of Systems, Security, Self-Adaptation, Autonomic Management, Eclipse Arrowhead**

I. Introduction

System of Systems (SoS) are large-scale integrated systems that can operate independently but are networked together for a period of time to achieve a higher goal, e.g., performance, robustness, security, etc. [1]. One of the main characteristics of SoS is the operational independence of the integrated systems. A system with low security level can compromise a system requiring high security level, and the compromise of such systems can lead to the compromise of the whole SoS, so security is an important concern.

Another characteristic of SoS is their distributed nature. In this manuscript, we use a drone-based application as an example of such a SoS. In [2], we proposed a use case for smart agriculture to assist winemakers and minimize travel time to remote and poorly connected infrastructures. The drone acts as a gateway by collecting sensor data and multispectral images of the vines and sending this data to a base station for offline analysis. In some cases, the drone is not always connected to the sensors and base station because the infrastructures are remote and poorly connected. Thus, it is a sporadically connected SoS where frequent changes may occur. If the security of one system (e.g., a wireless sensor network) is compromised, it may also affect the operation of other systems (e.g., the drone). Attackers can exploit these vulnerabilities to remotely control and disrupt the flow of data to/from the sensors and the drone. The ability to conduct a malicious attack on such systems can have serious consequences, and a large-scale, coordinated attack can disrupt national economies [3].

To establish a chain of trust between use case components, the Eclipse Arrowhead framework is used [4]. The goal of the framework is to efficiently support the development, deployment, and operation of SoS based on the fundamentals of service-oriented architecture (SoA): loose coupling, late binding, and lookup. The sensor nodes, drone, and base station are integrated into Arrowhead's local cloud through an automated onboarding procedure to ensure mutual authentication and thus secure communication [5]. A local cloud implements a set of services potentially used by all SoS applications.

While ensuring secure communication, the SoS should remain operational over a long period of time. To meet these requirements, the sensor nodes, the drone, and the base station must be optimally configured. However, due to the evolutionary development of SoS and emergent behavioral characteristics, ensuring these requirements can become a complex task. Uncertainties can occur due to internal factors (e.g., malfunction of a sensor node) or external factors (e.g., malicious attacks, weather conditions, etc.) that can affect secure communication between use case components. Even with mutual authentication, attackers can gain physical access to a sensor node and replicate many clones that have the same identity as the compromised node. The malicious node can then send additional sensor data to the drone. Similar behavior can occur when the sensor node malfunctions, such as when the battery is low. In this case, the sensor node cannot send enough data to the drone. To solve this problem, SoS must have mechanisms that allow them to self-adapt in the face of such uncertainties without human intervention.

In this manuscript, we propose to extend the smart agriculture use case with self-adaptation capabilities. We build on our previous work on the Generic Autonomic Management Framework [6, 7, 8] and extend it to support SoA-based frameworks as well. We propose a Generic Autonomic Management System (GAMS) to assist engineers in developing self-adaptive systems. Due to its generic nature, the system can be reused and extended for a variety of use cases without requiring major changes. This reduces the software engineering effort since the generic control mechanisms do not need to be (re)implemented for different use cases. A first concept of such a system is presented in [9]. In this manuscript, we present the design and implementation of a proof-of-concept for the proposed system and demonstrate its functionality in a smart agriculture use case.

The reminder of this manuscript is structured as follows. Section II reviews existing work on security and self-

DOI: 10.36244/ICJ.2021.3.7

adaptation in SoS. Section III provides the technical description of the smart agriculture SoS and motivates the need to extend the use case to include self-adaptation capabilities. Section IV describes the design and implementation of GAMS as part of the Eclipse Arrowhead framework. Section V presents the configuration of GAMS for the smart agriculture use case and experimental results. Section VI provides an overview of the results and future work.

II. Related Work

SoSs have several characteristics that distinguish them from traditional systems, such as the operational and managerial independence of their integrated systems, evolutionary development, emergent behavior, and geographic distribution. When designing a SoS, it is of utmost importance to understand the security implications of its features. For example, to address security-related aspects of a SoS that evolves over time and exhibits emergent characteristics, security mitigation approaches should be integrated. One approach is to augment the SoS with self-adaptive capabilities, as proposed in this manuscript. Therefore, we examine existing work on this topic.

Existing frameworks such as SASSY [10], MOSES [11], etc. have been developed to enable self-adaptation in service-oriented systems. Compared to these frameworks, our proposed system is intended to be generic so that it can be used in different SoA frameworks by a wide range of application systems without requiring a large amount of adjustments.

Ruz et.al. [12] have proposed a generic, self-adaptive framework to support monitoring and management tasks of component-based SoA applications. They separate the MAPE phases (Monitor, Analyze, Plan, and Execute) and implement them as distinct components that interact and support multiple sets of monitoring sources, conditions, policies, and distributed actions. Their main focus is on high scalability. Other works [13, 14, 15] justify the need for a generic solution for building self-adaptive systems. However, none of these works address security-related challenges that can be addressed by their solution.

Vishwa et al [16] have studied the adaptability of wireless sensor networks with the aim of highlighting the need for protection against malicious activities in such networks. They provide an evaluation of immune-based intrusion detection systems to determine that the functional requirements of wireless sensor networks such as self-organization, adaptability, fault tolerance, and self-healing are similar to human immune system mechanisms. The authors discuss the applicability of these theories to wireless sensor networks, and the paper ends with recommendations for expanding the study in the future.

There are other works dealing with SoS security such as [17], [18], [19], etc., which mainly focus on the engineering process that allows systems to integrate security at the design stage. They provide security artifacts (threats, attacks, assumptions about security properties, etc.) about the interaction with other elements or distributed systems to enable easy integration. In comparison, our work considers the evolving nature of SoS and its uncertainties that may arise at runtime, and proposes to address this problem by extending SoS with self-adaptation capabilities.

III. Smart Agriculture SoS

In this section, we present a use case from the smart agriculture domain, where the SoS approach is used to support winemakers and minimize travel time to remote and poorly connected infrastructures. In the following sections, we use this use case as a running example to show the functionality of the proposed system. An illustration of the use case is shown in Figure 1.

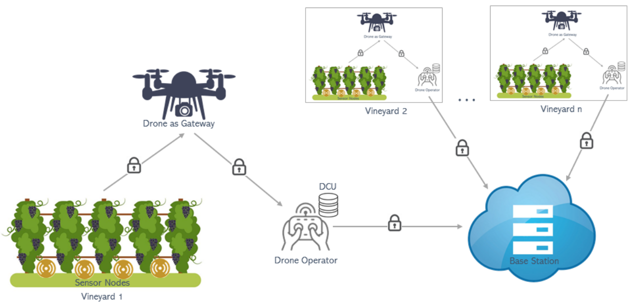


Fig. 1: High level view of the smart agriculture SoS use case.

A. Technical Description

The proposed SoS consists of a collection of sensor nodes connected via a wireless sensor network. These sensor nodes are strategically placed over the vineyard to obtain accurate measurements that can help detect diseases and conditions at early stages. The results of the sensor positioning are documented in [2]. The sensor nodes are equipped with sensors that collect environmental data such as air temperature, humidity and pressure, precipitation, wind speed and direction, sunlight, soil temperature and moisture, leaf wetness, etc. The sensor data is relayed to a drone, which acts as a gateway. The sensor nodes are constantly searching for the drone. When the drone is in range, a protected communication channel is established between the sensor nodes and the drone gateway. The drone sends sensor data from all sensor nodes and multispectral images of the vines to a base station for further analysis. However, in some cases, because the vineyards are located in harsh environments with poorly connected infrastructure, the sensor nodes first send the data to a data collection unit (DCU) located at the drone operator. Thus, this is a sporadically connected SoS. After the connection with the sensor nodes is established, the drone establishes a connection with the DCU and starts data transmission. The Wireless Local Area Network (WLAN) IEEE 802.11 is used for the communication link.

We have used the SysML modelling language to create a SysML block definition diagram for the smart agriculture use case, as shown in Figure 2.

1) Sensor Node: The sensor node consists of a single board computer, a Raspberry Pi, connected to several sensors, e.g. an air temperature and humidity sensor, a leaf wetness sensor, etc. The node has several *Python3* scripts that read the sensor data and write it to a comma-separated values (csv) file. Another *Python3* script continuously pings the drone gateway and transfers the csv files to the drone gateway if the connection is successful. The data transfer from the sensor nodes to the drone gateway is done using Hypertext Transfer Protocol Secure (HTTPs). For HTTPs, the communication protocol is encrypted using Transport Layer Security (TLS). After a successful file transfer, the file is moved to the archives and an entry is created in the log file.

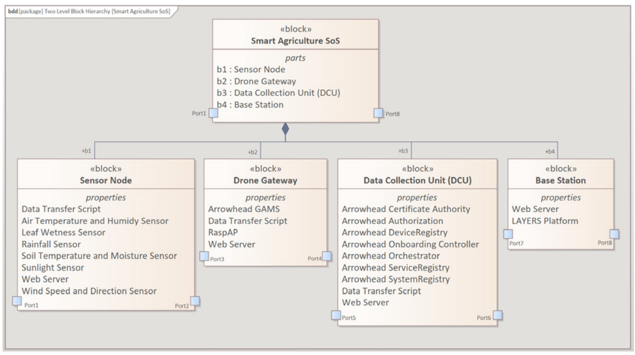


Fig. 2: SysML block definition diagram showing the hierarchy of the blocks composing the smart agriculture SoS use case.

2) Drone Gateway: The gateway mounted on the drone is a Raspberry Pi. RaspAP[1], a feature-rich wireless router software that works on Debian-based devices, is installed on the Raspberry Pi to configure the host access point daemon (hostapd). A web server receives the files sent by the sensor nodes. It also determines how the received files are handled, such as where they are stored. The gateway also contains a data transfer script similar to the script on the sensor node for post-processing the csv files. To ensure reusability and keep runtime resource costs low, both the web client on the sensor node and the web server on the drone gateway are written in *python*. To enhance the application with self-adaptation capabilities, the proposed Arrowhead support system (GAMS) is installed on the drone gateway. Details of its operation are described in Section IV.

3) Data Collection Unit: The DCU is a Debian system installed in a virtual machine. The supporting core systems of the Eclipse Arrowhead framework, which are invoked during the automated and secure onboarding procedure described later in the manuscript, are all installed on the DCU. In addition, a web server is installed to receive the files sent from the drone gateway and a data transfer script that sends them to the base station for further offline analysis.

4) Base Station: After the flight, the sensor data and the multispectral images stored in the base station are used for offline analysis. The LAYERS[2] platform is used to build a model and provide the necessary information about the condition of the vineyard. LAYERS is a platform that combines agronomic knowledge, earth observation remote sensing (drones, satellites, etc.) and artificial intelligence to create a proactive field monitoring system. It consists of a web tool with a map viewer and dashboard for field analysis, and an iOS and Android application for field sampling. However, the offline data analysis that takes place at the base station is beyond the scope of this manuscript.

[1] https://raspap.com/ [2] https://hemav.com/en/services/digital-agriculture-en/

B. Security and Adaptation Problem

Smart agriculture improves conventional farming methods by using sensors in vineyards to collect environmental data and autonomous vehicles (e.g., drones) to collect multispectral images of vineyards. These data are used for further offline analysis to improve production by optimizing crop management, such as accurate planting, irrigation, pesticide use, harvesting, etc. Despite the benefits, the use of internet-connected SoS can expose the agricultural sector to potential cyberattacks and vulnerabilities. Even if the vineyard is not connected to the Internet, it is an insecure network because an attacker only needs to be close enough to connect. These attacks can be used to remotely control and exploit sensors, actuators, and drones to destroy an entire field of standing crops, flood the vineyards, use smart drones to spray pesticides, etc. [20].

Therefore, it is of utmost importance to ensure trustworthy and secure communication of the drone with the sensor nodes in the vineyards and the base station. This ensures that only valid data is retrieved, damaged sensors are detected, and only authenticated and authorized systems participate in the communication. Even though all HTTP connections are secured via TLS (HTTPs), clients must be authenticated to ensure that sensor data is trusted.

To meet this security requirement, we use the automated and secure onboarding procedure of the Eclipse Arrowhead framework. An Arrowhead-compliant SoS is defined as a set of systems managed by the mandatory Arrowhead core systems that exchange information via services. Thus, a local cloud becomes an SoS. Also, two systems located in different local clouds and exchanging services form a SoS. The onboarding procedure [5] enables secure and trusted communication between such systems by using a chain of X.509 certificates generated at runtime. When a new device (e.g., a sensor node, drone, or base station) wants to interact with the Arrowhead local cloud, it should first authenticate itself using a valid preloaded Arrowhead certificate, manufacturer certificate, or shared key through the Onboarding Controller system. Each system hosted in this device will get a runtime certificate issued by Arrowhead. In Arrowhead, each local cloud has its own Certificate Authority (CA) system that issues and signs the runtime certificates of the systems. The CA system is the root of trust within the local cloud and can be signed by a central Arrowhead consortium, creating a chain of trust that allows different Arrowhead local clouds to be interconnected. Securely onboarding sensor nodes, drones, and base stations within the Arrowhead local cloud enables mutual authentication, allowing not only a TLS client to authenticate a server, but also a server to authenticate its client via X.509 certificates. The Arrowhead systems that are invoked during the onboarding procedure (ServiceRegistry, SystemRegistry, DeviceRegistry, Onboarding Controller, Orchestartor, Authorization, and Certificate Authority) are all located in the DCU. The source code and description of these systems can be found in the EclipseArrowhead GitHub[3] repository.

[3] https://github.com/eclipse-arrowhead/core-java-spring

This still leaves one attack vector open: clone attacks. Even with mutual authentication, attackers can gain physical access to a sensor node and replicate many clones with the same identity of the compromised node. The clones contain all the data of the legitimate sensor node and can successfully pass the onboarding procedure. Once the clones are on the network, they can exploit network operations such as routing, data collection, and key distribution, and even launch other attacks. This problem can be solved either by integrating secure elements into sensor nodes and drones (e.g., hardware security modules) to store keys and certificates in protected storage [21] or by extending the use case with self-adaptation capabilities that allow the system to adapt itself to a changing environment. The latter is described in the following sections.

IV. Generic Autonomic Management System

The Generic Autonomic Management System (GAMS) is designed and implemented as an Arrowhead support core system. A system is Arrowhead-compliant if it produces at least one service and consumes at least the three mandatory core services of the Eclipse Arrowhead framework, namely *ServiceDiscovery*, *AuthorizationControl* and *Orchestration* [4]. The *ServiceDiscovery* service is used to register and unregister services and to locate services among the registered services in the ServiceRegistry system. The *AuthorizationControl* service provides two different interfaces for retrieving authorization rights: (i) intra-cloud authorization, which defines an authorization right between a consumer and a provider system in the same local cloud for a particular service and (ii) inter-cloud authorization, which defines an authorization right for an external local cloud to consume a specific service from the local cloud. The *Orchestration* service provides application systems with orchestration information: where to connect. The output of this service includes rules that tell the application system which service provider systems to connect to and how (as a service consumer). Such orchestration rules include information about the reachability of a service provider (e.g., network address and port), service instance details within the provider system (e.g., base URL (Uniform Resource Locator), interface design specification, and other metadata), authorization-related information (e.g., access token and signature), etc.

The *GenericAutonomicManagement* service produced by GAMS is designed and implemented as a REST web service that can be invoked by different SoA-based frameworks. REST stands for representational state transfer and is a set of architectural constraints. Thus, a REST API is used for the interaction with the *GenericAutonomicManagement* service. A REST API is an application programming interface that conforms to the constraints of REST architectural style and enables interaction with REST web services [22]. The REST API has the following methods: (i) GET to retrieve information about the REST API resource, (ii) POST to create a REST API resource, (iii) PUT to update a REST API resource, and (iv) DELETE to delete a REST API resource. Compared to other protocols e.g. SOAP (Simple Object Access Protocol), REST APIs are faster and more lightweight for IoT applications [23].

A. System Description

We have used Systems Modeling Language (SysML) to create an internal block definition diagram of GAMS, as shown in Figure 3. The system enables autonomic control loops using MAPE-K (Monitor, Analyze, Plan, Execute and SharedKnowledge) as a reference feedback loop for self-adaptive systems [24]. An example of such interaction is a set of sensors and actuators (managed system), where GAMS is the autonomic manager (management system).

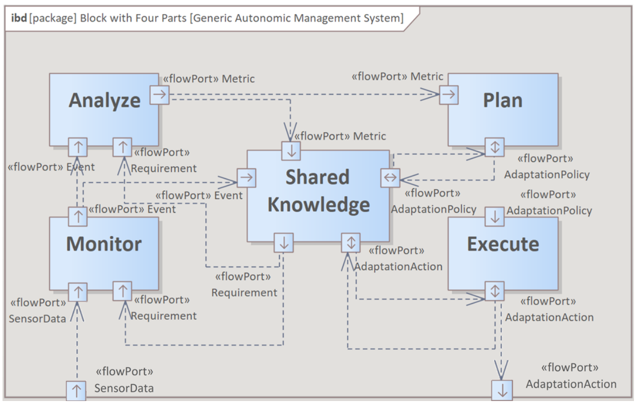


Fig. 3: SysML internal block definition diagram showing the internal structure of GAMS.

a) Monitor: The Monitor component continuously collects monitoring data from the sensor. The component performs a pre-analysis based on the incoming sensor data and the requirements stored in the SharedKnowledge. In case of a significant deviation, an event is generated and stored in the SharedKnowledge. The functions in this phase can aggregate the incoming data before passing it on to the next phase. GAMS allows you to specify the number of events to be considered for aggregation. If the specified number of events is not yet present, processing in this phase is aborted. The implemented functions are the following:

- **Sum:** creates a sum of sensor values.
- **Average:** creates an average of sensor values.
- **Trend:** indicates if the sensor values are increasing or decreasing.
- **Maximum:** uses the highest value in the next phase.
- **Minimum:** uses the lowest value in the next phase.
- **Count:** counts the number of incoming data in a specified time frame.
- **None:** does not aggregate the sensor value, but forwards it without change.

The SysML activity diagram of Monitor component is shown in Figure 4.

b) Analyze: The Analyze component evaluates the events received from the Monitor component with regard to the requirements and context data in the SharedKnowledge. If the requirements cannot be met, a change request is sent to the Plan component with a description of the metrics. Similar to the Monitor component, processing can be stopped if the result of the analysis shows that no action is required at this time. The implemented functions are the following:

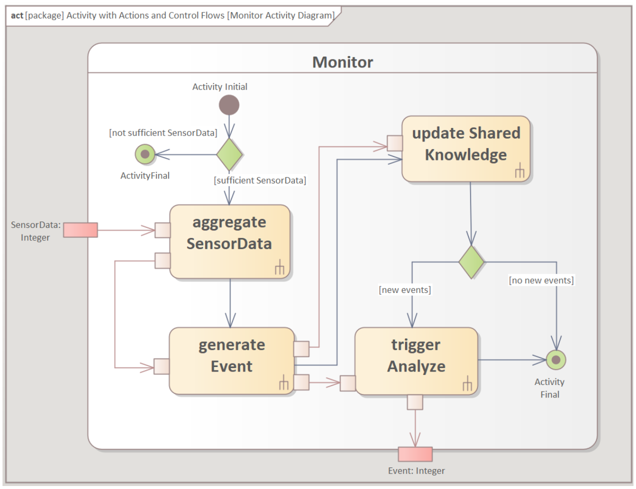


Fig. 4: SysML activity diagram showing the sequence of actions that are called as invocations of activities in the Monitor component of GAMS.

- **Count:** counts the number of generated events in a specified time frame.
- **Set-Point:** compares the incoming data with a configured target value (set-point). The target value could be a range with a lower and an upper set-point. In case both are used, this function acts as a double set-point.

The SysML activity diagram of Analyze component is shown in Figure 5.

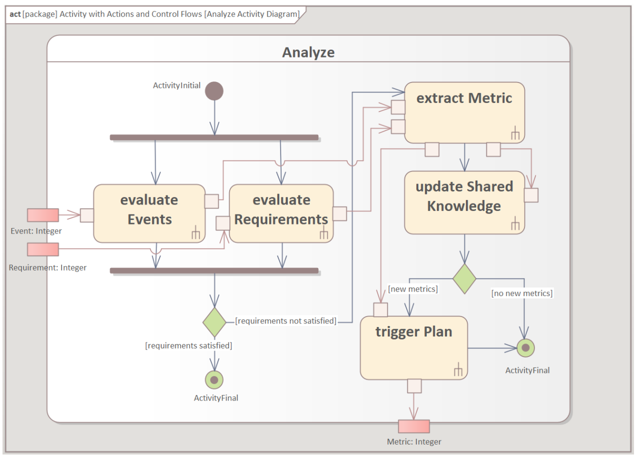


Fig. 5: SysML activity diagram showing the sequence of actions that are called as invocations of activities in the Analyze component of GAMS.

c) Plan: The Plan component is able to understand the metrics received from the Analyze component and to derive adaptation policies. It sets a corrective action and mandatory parameters for the autonomic element. An example of this is the use case of room temperature. A thermometer would be used as an input sensor and a heating or air conditioning system as an actuator. The planning component converts the incoming signal into a value that the actuator understands using the accumulated knowledge from the previous phases. The implemented functions are the following:

- **Match:** matches the incoming value as a key in a key-value structure and forwards the value to the next phase.
- **API Call:** makes an API Call to determine the value for the next phase.
- **Transform:** transforms the incoming value using a mathematical function.
- **None:** forwards the incoming value to the next phase without changing it.

The SysML activity diagram of Plan component is shown in Figure 6.

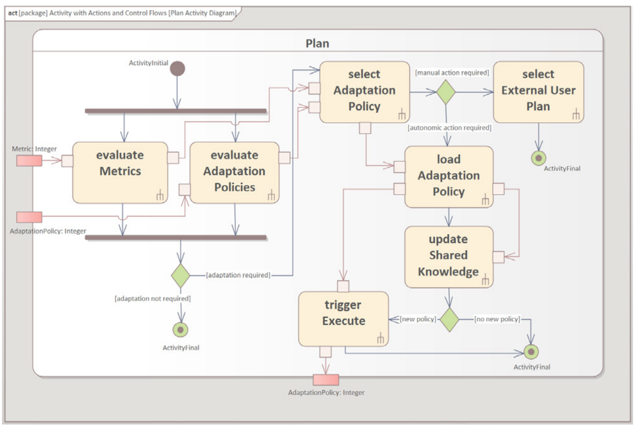


Fig. 6: SysML activity diagram showing the sequence of actions that are called as invocations of activities in the Plan component of GAMS.

d) Execute: The Execute component receives the policies from the Plan component and executes the derived action through the *GenericAutonomicManagement* service. The implemented functions are the following:

- **Composite Action:** allows the execution of multiple actions either in parallel or one after another.
- **API Call:** executes an API call as corrective action for the autonomic element.
- **Generate Event:** creates a new event to feed into the MAPE-K loop. This allows the re-evaluation of the sensor values with updated information from previous loops.
- **Logging Action:** logs the outcome of the MAPE-K loop.

The SysML activity diagram of Execute component is shown in Figure 7.

B. Service Interface Design Description

This section describes the HTTP/TLS/JSON *GenericAutonomicManagement* service interface. When a client request is made through a REST API, a representation of the state of the resource is transmitted to the GAMS endpoint. This information is transmitted in JSON format over HTTPs. Compared to other formats, JSON is language agnostic and can be read by both humans and machines [25].

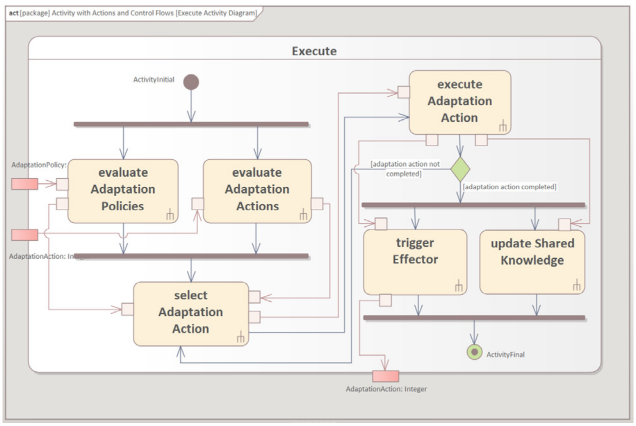
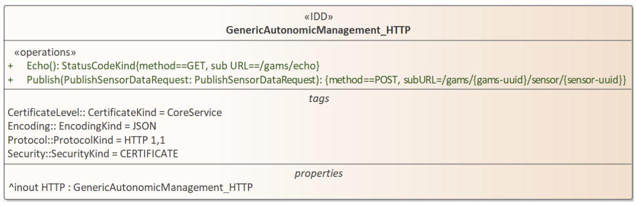

act [package] Activity with Actions and Control Flows [Execute Activity Diagram]
Execute
ActivityInitial
AdaptationPolicy:
AdaptationAction: Integer
evaluate Adaptation Policies
evaluate Adaptation Actions
select Adaptation Action
execute Adaptation Action
[adaptation action not completed]
[adaptation action completed]
trigger Effector
update Shared Knowledge
ActivityFinal
AdaptationAction: Integer

Fig. 7: SysML activity diagram showing the sequence of actions that are called as invocations of activities in the Execute component of GAMS.

1) Service Operations: In the following, an overview of the interface of the *GenericAutonomicManagement* service, its operations, data models and implementation is given and shown in Figure8. Both operations are described by their name and by an input type (between parenthesis) and an output type (at the end, preceded by a colon). Input and output types are only specified if they are accepted or returned by the interface in question.


«IDD»
GenericAutonomicManagement_HTTP

«operations»
+ Echo(): StatusCodeKind{method==GET, sub URL==/gams/echo}
+ Publish(PublishSensorDataRequest: PublishSensorDataRequest): {method==POST, subURL=/gams/{gams-uuid}/sensor/{sensor-uuid}}

tags
CertificateLevel:: CertificateKind = CoreService
Encoding:: EncodingKind = JSON
Protocol::ProtocolKind = HTTP 1,1
Security::SecurityKind = CERTIFICATE

properties
^inout HTTP : GenericAutonomicManagement_HTTP

Fig. 8: SysML block description diagram of the *GenericAutonomicManagement* service interface and its operations.

Publish(PublishSensorDataRequest) The Publish operation is used to send new sensor readings to the service, as exemplified in Listing 1. The sensor readings could be either numeric (integer or floating point number) or textual (event based), depending on the configuration. The sensor inputs feed the MAPE-K control loop of GAMS and eventually trigger a change on an actuator. The specific REST operation associated with this is: `POST/gams/{gams-uuid}/sensor/{sensor-uuid}`

```

POST /gams/f8c3de3d-1fea-4d7c-a8b0-29f63c4c3454/sensor/123
    e4567-e89b-42d3-a456-556642440000 HTTP/1.1

{
  "timestamp": "2021-07-04 12:00:00",
  "data": 1.2
}
```

Listing 1: An example of the Publish invocation for a floating point number.

Echo():StatusCodeKind The Echo operation returns an "is alive" response from the *GenericAutonomicManagement* service, as exemplified in Listing 2, which can be used to test the availability of the core service. The specific REST operation associated with this is: `GET/gams/echo`

```

GET /gams/echo HTTP/1.1

Got it!
```

Listing 2: An Echo invocation response.

Both operations respond with the HTTP status code `201 Created` when successfully invoked. The error codes are: `400 Bad Request` if request is incorrect, `401 Unauthorized` if an improper client certificate was provided, and `500 Internal Server Error` if *GenericAutonomicManagement* service cannot process the request due to an internal problem.

2) Information Model: The Publish operation has as input type the **PublishSensorDataRequest** structure, which is used to publish new sensor readings. The identification of the sensor is possible in two ways:

- Uniform Resource Identifier (URI) path parameters denoting the GAMS instance and the sensor identification,
- URI path parameter denoting the GAMS instance and using the source IP address to determine the sensor.

The POST request contains the parameters (data types) described in Table I.

Field	Type	Description
timestamp	DateTime	The date and time of the sensor reading. Pinpoints a specific moment in time.
data	SensorData	The value of the sensor reading. May be configured as integer number, floating-point number, or text string.

TABLE I: POST request parameters of the Publish operation.

The activity diagram shown in Figure 9 describes the process of publishing sensor data in GAMS.

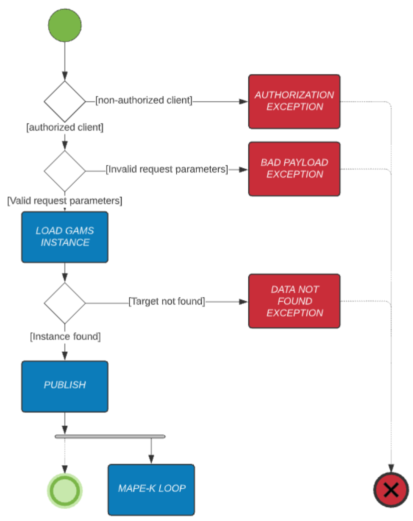


Fig. 9: Information model as an activity diagram that describes the process of publishing sensor data in GAMS.

To interact with the Arrowhead local cloud, each client must be authenticated and authorized through the onboarding procedure and register its device, systems and services respectively in the DeviceRegistry, SystemRegistry, and ServiceRegistry systems. When the client is properly authenticated and authorized, it receives the GAMS endpoint from the Orchestrator system. The client sends a POST request to the provided GAMS endpoint. If the POST request parameters are valid, the GAMS instance is loaded and the Publish operation is invoked.

V. GAMS INTEGRATED IN THE SMART AGRICULTURE SOS

In the smart agriculture use case described above, despite mutual authentication through the Arrowhead automated onboarding procedure, attackers can gain physical access to a sensor node and replicate many clones that have the same identity as the compromised node. The malicious node can spoof the media access control (MAC) address of the legitimate node to bypass possible security measures at the drone access point (e.g., MAC address whitelist). In addition, the malicious node can spoof the IP address of the legitimate node to bypass the IP address restriction on the client certificate, allowing the malicious node to send additional sensor data to the drone. Another problem can be caused by a real malfunction of a sensor node, such as a low battery. We propose to integrate GAMS into the smart agriculture use case so that the SoS can adapt itself in the face of such uncertainties.

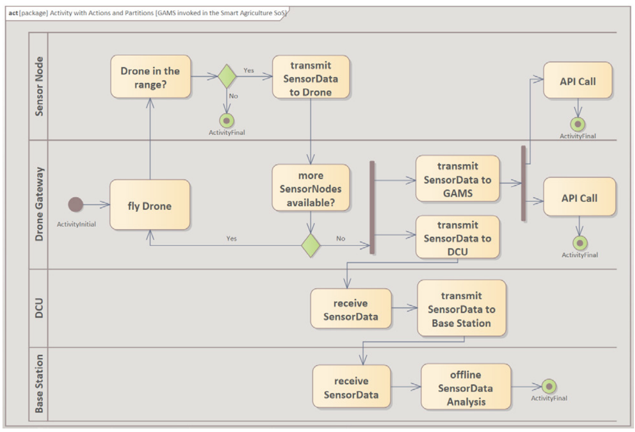

act [package] Activity with Actions and Partitions [GAMS invoked in the Smart Agriculture SoS]
Sensor Node
Drone Gateway
DCU
Base Station
Drone in the range?
Yes
No
ActivityFinal
transmit SensorData to Drone
API Call
ActivityFinal
ActivityInitial
fly Drone
more SensorNodes available?
Yes
No
transmit SensorData to GAMS
transmit SensorData to DCU
API Call
ActivityFinal
receive SensorData
transmit SensorData to Base Station
receive SensorData
offline SensorData Analysis
ActivityFinal

Fig. 10: SysML activity diagram that describes the process of invoking GAMS in the smart agriculture SoS use case.

The invocation of GAMS takes place in the drone gateway, as shown in Figure 10. The SharedKnowledge of GAMS contains information about traffic profiles, e.g., the expected number of sensor data, which serve as a baseline for normal traffic behavior. Such profiles may be provided by the industrial partners or generated from historical data. GAMS uses the sensor data as input for MAPE-K loop and compares the actual traffic to the baseline. It sends API calls when suspicious behaviors are detected that may indicate the following: (i) possible malicious attacks or (ii) sensor node malfunctions. An application-specific effector (or actuator) is then used to execute the adaptation action decided by GAMS (management system) in the drone and sensor nodes (managed system).

A. GAMS Configuration

GAMS is developed as a generic system; therefore, it should be reusable and extensible so that it can be applied to a variety of use cases without requiring major changes to the solution. As described in Section IV-A, multiple functions are implemented in each phase to cover a range of use cases. Configuring GAMS for a particular use case means that one or more functions from each phase will be selected and extended as needed, based on adaptation requirements.

a) Monitor: We have used the **Count** function implemented in the Monitor component. Each sensor node uses the Publish operation of *GenericAutonomicManagement* service interface to send sensor readings to GAMS. After receiving the sensor data, GAMS counts the number of sensor data for a configured time frame, in our use case it counts the number of sensor data per day.

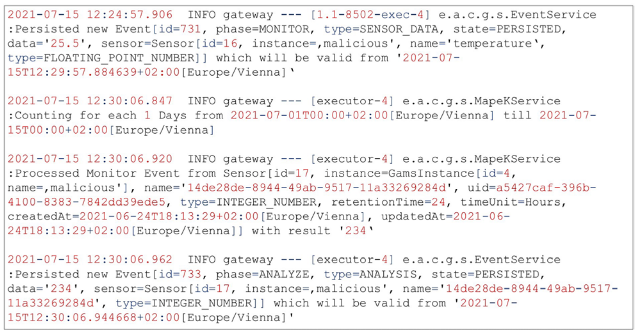

```
2021-07-15 12:24:57.906  INFO gateway --- [1.1-8502-exec-4] e.a.c.g.s.EventService
:Persisted new Event[id=731, phase=MONITOR, type=SENSOR_DATA, state=PERSISTED,
data='25.5', sensor=Sensor[id=16, instance=,malicious', name='temperature‛,
type=FLOATING_POINT_NUMBER]] which will be valid from '2021-07-
15T12:29:57.884639+02:00[Europe/Vienna]‛

2021-07-15 12:30:06.847  INFO gateway --- [executor-4] e.a.c.g.s.MapeKService
:Counting for each 1 Days from 2021-07-01T00:00+02:00[Europe/Vienna] till 2021-07-
15T00:00+02:00[Europe/Vienna]

2021-07-15 12:30:06.920  INFO gateway --- [executor-4] e.a.c.g.s.MapeKService
:Processed Monitor Event from Sensor[id=17, instance=GamsInstance[id=4,
name=,malicious'], name='14de28de-8944-49ab-9517-11a33269284d', uid=a5427caf-396b-
4100-8383-7842dd39ede5, type=INTEGER_NUMBER, retentionTime=24, timeUnit=Hours,
createdAt=2021-06-24T18:13:29+02:00[Europe/Vienna], updatedAt=2021-06-
24T18:13:29+02:00[Europe/Vienna]] with result '234‛

2021-07-15 12:30:06.962  INFO gateway --- [executor-4] e.a.c.g.s.EventService
:Persisted new Event[id=733, phase=ANALYZE, type=ANALYSIS, state=PERSISTED,
data='234', sensor=Sensor[id=17, instance=,malicious', name='14de28de-8944-49ab-9517-
11a33269284d', type=INTEGER_NUMBER]] which will be valid from '2021-07-
15T12:30:06.944668+02:00[Europe/Vienna]'
```

Fig. 11: The log file of the Monitor component.

The log file of the Monitor component is shown in Figure 11. In this example, it counts 234 SensorData/day and stores this as an event in the SharedKnowledge.

b) Analyze: We have used the **Set-Point** function implemented in the Analyze component. A Set-Point controller has two set points to which it switches the output. For our use case, the accepted range is between 180 and 220 SensorData/day. The Analyze component returns a positive number when the input exceeds the upper set point (POSITIVE_METRIC) and a negative number when the input is below the lower set point (NEGATIVE_METRIC). In all other cases, zero is returned (ZERO_METRIC). These metrics are forwarded to the Plan component.

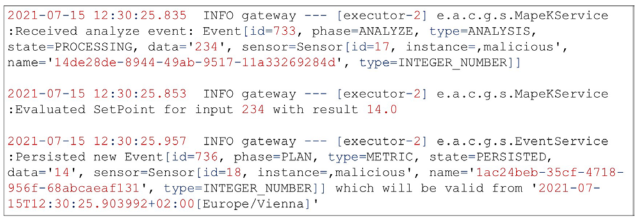
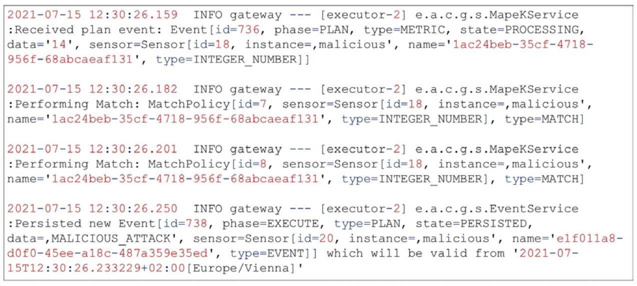

```
2021-07-15 12:30:25.835  INFO gateway --- [executor-2] e.a.c.g.s.MapeKService
:Received analyze event: Event[id=733, phase=ANALYZE, type=ANALYSIS,
state=PROCESSING, data='234', sensor=Sensor[id=17, instance=,malicious',
name='14de28de-8944-49ab-9517-11a33269284d', type=INTEGER_NUMBER]]

2021-07-15 12:30:25.853  INFO gateway --- [executor-2] e.a.c.g.s.MapeKService
:Evaluated SetPoint for input 234 with result 14.0

2021-07-15 12:30:25.957  INFO gateway --- [executor-2] e.a.c.g.s.EventService
:Persisted new Event[id=736, phase=PLAN, type=METRIC, state=PERSISTED,
data='14', sensor=Sensor[id=18, instance=,malicious', name='1ac24beb-35cf-4718-
956f-68abcaeaf131', type=INTEGER_NUMBER]] which will be valid from '2021-07-
15T12:30:25.903992+02:00[Europe/Vienna]'
```

Fig. 12: The log file of the Analyze component.

The log file of the Analyze component is shown in Figure 12. In this example it returns a POSITIVE_METRIC, since the input exceeds the upper set point with 14 SensorData/day.

c) Plan: We have used the **Match** function implemented in the Plan component. If an adaptation is required (POSITIVE_METRIC or NEGATIVE_METRIC), the Plan component selects a matching adaptation policy. If no adaptation is required (ZERO_METRIC), the operation is stopped.

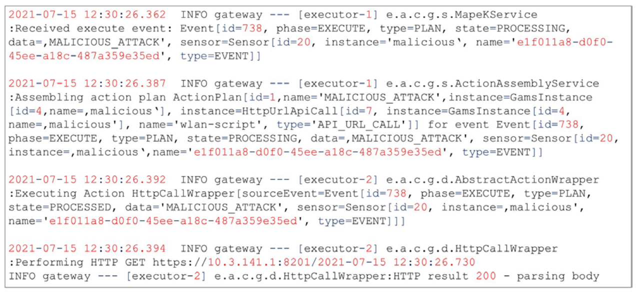


```
2021-07-15 12:30:26.159  INFO gateway --- [executor-2] e.a.c.g.s.MapeKService
:Received plan event: Event[id=736, phase=PLAN, type=METRIC, state=PROCESSING,
data='14', sensor=Sensor[id=18, instance=,malicious', name='1ac24beb-35cf-4718-
956f-68abcaeaf131', type=INTEGER_NUMBER]]

2021-07-15 12:30:26.182  INFO gateway --- [executor-2] e.a.c.g.s.MapeKService
:Performing Match: MatchPolicy[id=7, sensor=Sensor[id=18, instance=,malicious',
name='1ac24beb-35cf-4718-956f-68abcaeaf131', type=INTEGER_NUMBER], type=MATCH]

2021-07-15 12:30:26.201  INFO gateway --- [executor-2] e.a.c.g.s.MapeKService
:Performing Match: MatchPolicy[id=8, sensor=Sensor[id=18, instance=,malicious',
name='1ac24beb-35cf-4718-956f-68abcaeaf131', type=INTEGER_NUMBER], type=MATCH]

2021-07-15 12:30:26.250  INFO gateway --- [executor-2] e.a.c.g.s.EventService
:Persisted new Event[id=738, phase=EXECUTE, type=PLAN, state=PERSISTED,
data=,MALICIOUS_ATTACK', sensor=Sensor[id=20, instance=,malicious', name='e1f011a8-
d0f0-45ee-a18c-487a359e35ed', type=EVENT]] which will be valid from '2021-07-
15T12:30:26.233229+02:00[Europe/Vienna]'
```

Fig. 13: The log file of the Plan component.

The log file of the Plan component is shown in Figure 13. In this example, it returns MALICIOUS_ATTACK adaptation policy since it matches with the POSITIVE_METRIC.

d) Execute: We have used two functions implemented in the Execute component. The **API Call** function is used to trigger an effector, in our use case to invoke a service for changing the WLAN password in the drone and all sensor nodes, except the compromised node. The **Logging Action** function is used to create a log entry when the number of sensor data is below the lower set point.



```
2021-07-15 12:30:26.362  INFO gateway --- [executor-1] e.a.c.g.s.MapeKService
:Received execute event: Event[id=738, phase=EXECUTE, type=PLAN, state=PROCESSING,
data=,MALICIOUS_ATTACK', sensor=Sensor[id=20, instance='malicious‛, name='e1f011a8-d0f0-
45ee-a18c-487a359e35ed', type=EVENT]]

2021-07-15 12:30:26.387  INFO gateway --- [executor-1] e.a.c.g.s.ActionAssemblyService
:Assembling action plan ActionPlan[id=1,name='MALICIOUS_ATTACK',instance=GamsInstance
[id=4,name=,malicious‛], instance=HttpUrlApiCall[id=7, instance=GamsInstance[id=4,
name=,malicious'], name='wlan-script', type='API_URL_CALL']] for event Event[id=738,
phase=EXECUTE, type=PLAN, state=PROCESSING, data=,MALICIOUS_ATTACK', sensor=Sensor[id=20,
instance=,malicious‛,name='e1f011a8-d0f0-45ee-a18c-487a359e35ed', type=EVENT]]

2021-07-15 12:30:26.392  INFO gateway --- [executor-2] e.a.c.g.d.AbstractActionWrapper
:Executing Action HttpCallWrapper[sourceEvent=Event[id=738, phase=EXECUTE, type=PLAN,
state=PROCESSED, data='MALICIOUS_ATTACK', sensor=Sensor[id=20, instance=,malicious',
name='e1f011a8-d0f0-45ee-a18c-487a359e35ed', type=EVENT]]]

2021-07-15 12:30:26.394  INFO gateway --- [executor-2] e.a.c.g.d.HttpCallWrapper
:Performing HTTP GET https://10.3.141.1:8201/2021-07-15 12:30:26.730
INFO gateway --- [executor-2] e.a.c.g.d.HttpCallWrapper:HTTP result 200 - parsing body
```

Fig. 14: The log file of the Execute component.

The log file of the Execute component is shown in Figure 14. In this example, it executes a HttpCallWrapper adaptation action, which is the API call associated with MALICIOUS_ATTACK adaptation policy.

The log files presented in this section illustrate only one example of suspicious behavior that can be detected by GAMS, namely a possible malicious attack due to an increased number of SensorData/day. Another example is a possible malfunction of a sensor node due to a decreased number of SensorData/day. In this case, GAMS would return a NEGATIVE_METRIC corresponding to the MALFUNCTION_SENSOR adaptation policy and create a log entry as an adaptation action.

B. Results

The experimental environment consists of a *testDataset* generator script written in *python3*, a legitimate node containing valid sensor data readings (generated by the *testDataset* script), and a clone containing invalid sensor data readings (generated by the *testDataset* script). The invalid sensor data readings typically shows a higher temperature (+10°C) and contains more sensor data entries than the upper set point (220 SensorData/day) stored in the SharedKnowledge. The rest of the environment uses the actual scripts from the use case as described in Section III-A.

The measurements are simulated for 10 days. On the first and second day, the legitimate node sends valid data. From the third day, the clone with the same credentials as the legitimate node sends invalid data. The drone flies every other day. When GAMS is invoked, it receives the sensor data collected by the drone and detects an increased number of sensor data exceeding the upper limit defined in the SharedKnowledge on the third and fourth day. According to the configuration described in SectionV-A, GAMS sends an API call to the drone and the legitimate node to change the WLAN password. From this point on, the clone can no longer connect to the drone, so only valid data is sent. For testing purposes, we used only two nodes. However, in a real scenario, both the clone and the compromised node do not receive the new password to connect. The results are shown in Figure 15.

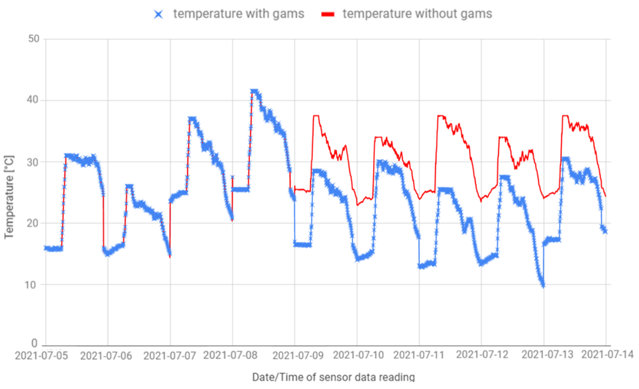


Fig. 15: The use case execution with/without GAMS invoked.

The results indicate that GAMS is able to detect unexpected changes in a SoS and take adaptation actions without human intervention, which can help maintain required security levels for the use case even in the presence of uncertainty.

VI. Conclusion

SoSs evolve over time and exhibit new properties, such that security controls introduced in the design phase to mitigate potential attack vectors may not be appropriate or sufficient later in operation. In this manuscript, we justify the need to provide SoS with self-adaptive capabilities to address security issues that may arise from uncertainties that are difficult to predict before the system is deployed. Such uncertainties may stem from factors internal or external to the SoS.

To address this challenge, we proposed a generic autonomic management system (GAMS) that automatically tracks runtime uncertainties and adapts SoS settings without human intervention. The internal building blocks of GAMS (Monitor, Analyze, Plan, Execute, and Shared-Knowledge) are designed and implemented in such a way that they can be reused and extended for a variety of use cases without requiring major changes. This reduces the software engineering effort. We integrated GAMS into a smart agriculture use case to demonstrate its functionality. The results showed that GAMS is able to detect a change in the environment and successfully send an API call to the drone and sensor nodes to change system settings to mitigate a potential malicious attack or detect a sensor node malfunction.

As future work, we plan to improve the codebase of GAMS to increase performance for resource-constrained systems, and evaluate it in various use cases to show its generic property.

ACKNOWLEDGMENT

This work has received funding from ECSEL Joint Undertaking (JU) under grant agreement No 826610 Comp4Drones project. The JU receives support from EU's Horizon 2020 research and innovation programme and Spain, Austria, Belgium, Czech Republic, France, Italy, Latvia, Netherlands.

REFERENCES

[1] Mark W Maier. Architecting principles for systems-of-systems. *Systems Engineering: The Journal of the International Council on Systems Engineering*, 1(4):267– 284, 1998. DOI: 10.1002/(SICI)1520-6858(1998)1:4<267::AID-SYS3>3.0.CO;2-D.

[2] Silia Maksuti, Michael Pickem, Mario Zsilak, Anna Stummer, Markus Tauber, Marcus Wieschhoff, Dominic Pirker, Christoph Schmittner, and Jerker Delsing. Establishing a chain of trust in a sporadically connected cyber-physical system. In *2021 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pages 890–895. IEEE, 2021.

[3] Sina Sontowski, Maanak Gupta, Sai Sree Laya Chukkapalli, Mahmoud Abdelsalam, Sudip Mittal, Anupam Joshi, and Ravi Sandhu. Cyber attacks on smart farming infrastructure. In *2020 IEEE 6th International Conference on Collaboration and Internet Computing (CIC)*, pages 135–143. IEEE, 2020. DOI: 10.1109/CIC50333.2020.00025.

[4] Jerker Delsing. *Iot automation: Arrowhead framework*. Crc Press, 2017.

[5] Ani Bicaku, Silia Maksuti, Csaba Hegedűs, Markus Tauber, Jerker Delsing, and Jens Eliasson. Interacting with the arrowhead local cloud: On-boarding procedure. In *2018 IEEE industrial cyber-physical systems (ICPS)*, pages 743–748. IEEE, 2018. DOI: 10.1109/ICPHYS.2018.8390800.

[6] Markus Tauber. Autonomic management in a distributed storage system. arXiv preprint arXiv:1007.0328, 2010.

[7] Silia Maksuti, Ani Bicaku, Markus Tauber, Silke Palkovits-Rauter, Sarah Haas, and Jerker Delsing. Towards flexible and secure end-to-end communication in industry 4.0. In *2017 IEEE 15th International Conference on Industrial Informatics (INDIN)*, pages 883–888. IEEE, 2017. DOI: 10.1109/INDIN.2017.8104888.

[8] Silia Maksuti, Oliver Schluga, Giuseppe Settanni, Markus Tauber, and Jerker Delsing. Self-adaptation applied to mqtt via a generic autonomic management framework. In *2019 IEEE International Conference on Industrial Technology (ICIT)*, pages 1179–1185. IEEE, 2019. DOI: 10.1109/ICIT.2019.8754937.

[9] Silia Maksuti, Markus Tauber, and Jerker Delsing. Generic autonomic management as a service in a soa-based framework for industry 4.0. In *IECON 2019 – 45th Annual Conference of the IEEE Industrial Electronics Society*, volume 1, pages 5480–5485. IEEE, 2019. DOI: 10.1109/IECON.2019.8927245.

[10] Daniel Menasce, Hassan Gomaa, Joao Sousa, et al. Sassy: A framework for self-architecting service-oriented systems. *IEEE software*, 28(6):78–85, 2011. DOI: 10.1109/MS.2011.22.

[11] Valeria Cardellini, Emiliano Casalicchio, Vincenzo Grassi, Stefano Iannucci, Francesco Lo Presti, and Raffaela Mirandola. Moses: A framework for qos driven runtime adaptation of service-oriented systems. *IEEE Transactions on Software Engineering*, 38(5):1138–1159, 2011. DOI: 10.1109/TSE.2011.68.

[12] Cristian Ruz, Françoise Baude, and Bastien Sauvan. Flexible adaptation loop for component-based soa applications. In *Seven International Conference on Autonomic and Autonomous Systems*, 2011.

[13] Sylvain Frey, Ada Diaconescu, David Menga, and Isabelle Demeure. Towards a generic architecture and methodology for multi-goal, highly-distributed and dynamic autonomic systems. In *10th International Conference on Autonomic Computing ({ICAC} 13)*, pages 201–212, 2013.

[14] Mahdi Ben Alaya and Thierry Monteil. Frameself: an ontology-based framework for the self-management of machine-to-machine systems. *Concurrency and Computation: Practice and Experience*, 27(6):1412–1426, 2015. DOI: 10.1002/cpe.3168.

[15] Svein Hallsteinsen, Kurt Geihs, Nearchos Paspallis, Frank Eliassen, Geir Horn, Jorge Lorenzo, Alessandro Mamelli, and George Angelos Papadopoulos. A development framework and methodology for self-adapting applications in ubiquitous computing environments. *Journal of Systems and Software*, 85(12):2840–2859, 2012.

[16] Vishwa T Alaparthy, Amar Amouri, and Salvatore D Morgera. A study on the adaptability of immune models for wireless sensor network security. *Procedia computer science*, 145:13–19, 2018. DOI: 10.1016/j.procs. 2018.11.003.

[17] Jose Fran Ruiz, Carsten Rudolph, Antonio Maña, and Marcos Arjona. A security engineering process for systems of systems using security patterns. In *2014 IEEE International Systems Conference Proceedings*, pages 8–11. IEEE, 2014. DOI: 10.1109/SysCon.2014.6819228.

[18] J Dahmann, George Rebovich, Michael McEvilley, and G Turner. Security engineering in a system of systems environment. In *2013 IEEE International Systems Conference (SysCon)*, pages 364–369. IEEE, 2013. DOI: 10.1109/SysCon.2013.6549907.

[19] Larry B Rainey and Andreas Tolk. Modeling and simulation support for system of systems engineering applications. 2015. DOI: 10.1002/9781118501757.

[20] Maanak Gupta, Mahmoud Abdelsalam, Sajad Khorsandroo, and Sudip Mittal. Security and privacy in smart farming: Challenges and opportunities. *IEEE Access*, 8:34564–34584, 2020. DOI: 10.1109/ACCESS.2020.2975142.

[21] Dominic Pirker, Thomas Fischer, Christian Lesjak, and Christian Steger. Global and secured uav authentication system based on hardware-security. In *2020 8th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud)*, pages 84–89. IEEE, 2020. DOI: 10.1109/MobileCloud48802.2020.00020.

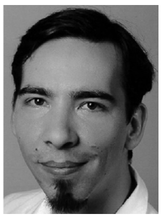
[22] Mark Masse. *REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces*. " O'Reilly Media, Inc.", 2011.

[23] Snehal Mumbaikar, Puja Padiya, et al. Web services based on soap and rest principles. I*nternational Journal of Scientific and Research Publications*, 3(5):1–4, 2013.

[24] Jeffrey O Kephart and David M Chess. The vision of autonomic computing. *Computer,* 36(1):41–50, 2003. DOI: 10.1109/MC.2003.1160055.

[25] Nurzhan Nurseitov, Michael Paulson, Randall Reynolds, and Clemente Izurieta. Comparison of json and xml data interchange formats: a case study. *Caine*, 9:157–162, 2009.

DI Silia Maksuti is a PhD student at Luleå University of Technology, Sweden, and works as a researcher at the University of Applied Sciences Burgenland, Austria, in the research center ”Cloud and Cyber Physical Systems Security”. Recently, she was working at the Austrian Institute of Technology (AIT) in the AIT’s ICT-Security Program. She received the Dipl-Ing. degree in Communication Engineering from the Carinthia University of Applied Sciences, Klagenfurt, Austria, and her B.Sc. degree in Telecommunication Engineering from the Polytechnic University of Tirana, Albania. She has been part of several EU projects, e.g., SECCRIT, SEMI40, PRODUCTIVE4.0, ArrowheadTools and Comp4Drones.



Mario Zsilak works as software engineer at the Center for Cloud and CPS Security at the Forschung Burgenland GmbH. He has contributed to Arrowhead in a number of projects. He is currently completing his Master in the MSc Program Business Process Management and Engineering at the University of Applied Sciences Burgenland. He completed his BSc in Information and Communication Systems and Services in 2017, at the University of Applied Sciences Technikum Wien.

Prof. (FH) Dr. Markus Tauber works as Chief Scientific Officer at Research Studios Austria Forschungsgesellschaft. Between 2015 until 2021, he worked as FH-Professor for the University of Applied Sciences Burgenland, where he held the position: director of the MSc program “Cloud Computing Engineering” and led the research center “Cloud and Cyber-Physical Systems Security”. From 2012 until 2015, he coordinated the “High Assurance Cloud” research topic at the Austrian Institute of Technology (AIT) part of AIT’s ICT-Security Program. Amongst other activities, he was the coordinator of the FP7 Project “Secure Cloud computing for CRitical infrastructure IT” - (www.seccrit.eu) and involved in the ARTEMIS Project Arrowhead. From 2004 to 2012, he was working at the University of St Andrews (UK), where he worked as a researcher on various topics in the area of network and distributed systems and was awarded a PhD in Computer Science for which he was working on “Autonomic Management in Distributed Storage Systems”.

Prof. Jerker Delsing received the M.Sc. in Engineering Physics at Lund Institute of Technology, Sweden 1982. In 1988 he received the PhD. degree in Electrical Measurement at the Lund University. During 1985 - 1988 he worked part time at Alfa-Laval - SattControl (now ABB) with development of sensors and measurement technology. In 1994 he was promoted to associate professor in Heat and Power Engineering at Lund University. Early 1995 he was appointed full professor in Industrial Electronics at Lulea University of Technology where he currently is the scientific head of EISLAB, http://www.ltu.se/eislab. His present research profile can be entitled IoT and SoS Automation, with applications to automation in large and complex industry and society systems. Prof. Delsing and his EISLAB group has been a partner of several large EU projects in the field, e.g. Socrades, IMC-AESOP, Arrowhead, FAR-EDGE, Productive4.0 and Arrowhead Tools. Delsing is a board member of ARTEMIS, ProcessIT.EU and ProcessIT Innovations.