Received 6 December 2021; revised 31 January 2022; accepted 1 February 2022. Date of publication 7 February 2022; date of current version 24 February 2022. The review of this paper was arranged by Associate Editor Stamatis Karnouskos.

Digital Object Identifier 10.1109/OJIES.2022.3149093

Dynamical Orchestration and Configuration Services in Industrial IoT Systems: An Autonomic Approach

AN NGOC LAM [1,2] (Student Member, IEEE), ØYSTEIN HAUGEN [1] (Member, IEEE), AND JERKER DELSING [2] (Member, IEEE)

[1]Faculty of Computer Science, Engineering and Economics, Østfold University College, 1757 Halden, Norway
[2]Department of Computer Science, Electrical and Space Engineering, Luleå University of Technology, 971 87 Luleå, Sweden

CORRESPONDING AUTHOR: AN NGOC LAM (e-mail: an.ngoc.lam@ltu.se)

This work was supported in part by the EU ECSEL Joint Undertaking under Grant Agreement Numbers 737459 Productive4.0 Project and 826452 Arrowhead Tools Project, and in part by the Research Council of Norway under Project 282904.

ABSTRACT The Industrial Internet of Things (IIoT) enables the integration of physical devices such as sensors and actuators into the virtual world of automation application systems via different communication protocols. Interoperability among the "things" appears to be one of the biggest conceptual and technological challenges in developing the IIoT framework. Typically, collaboration at the field device level is very limited. Instead, the decision-making process is usually propagated to higher levels with substantial computational resources. This centralized architecture has been widely deployed based on global cloud infrastructure. However, sending data over the cloud for analysis may bring about privacy and security threats. Besides, network latency could be another factor that reduces adaptability. In this article, we propose a decentralized approach that applies the concepts of local automation cloud. By using semantic technologies to achieve autonomicity, the approach enables real-time monitoring of the control systems within one local cloud and automates orchestration and configuration locally through adaptation based on semantic policies. The approach is deployed and tested on a chemical production use case in which business-level policies have been used for dynamical planning for suppliers and automatic detection of malfunctioning sensors with subsequent adaptation to continuing supply planning and production as smooth as possible.

INDEX TERMS Arrowhead framework, autonomic computing, industrial IoT, self-adaptation, semantic interoperability, semantic web.

I. INTRODUCTION

Industry 4.0 promotes a new generation of manufacturing systems with integrated sensors and actuators that are used for process control at the factory floor [1]. These physical devices, together with industrial production machines, are managed by software systems interconnected through industrial networks and protocols. The integration at the factory floor plays a critical part in the optimization of manufacturing and business processes as well as the generation of new business values for the manufacturers. Furthermore, in order to quickly adapt to the contemporary demand of the market or to meet new requirements from the customers, the industrial systems and manufacturing processes need to be monitored and supervised continuously [2]. To achieve these goals, recent information technologies such as Industrial Internet of Things (IIoT), Cyber-physical Systems (CPS), Cloud-based technologies, and Service-oriented Architecture (SoA) have been introduced into modern manufacturing systems.

Typically, recent research in the industrial automation domain presents system architecture in which the management and the location of the knowledge are away from the shop floor physical machines [3]. Specifically, data from different components of the production line are collected and sent over a scalable Cloud infrastructure to analyze. Nevertheless, employing such global cloud architecture may introduce problems related to security, privacy, and latency into the manufacturing systems. Furthermore, cloud-based technologies may bring challenges such as management of the big and heterogeneous data generated by a substantial number of devices at the shop floor. The computing resources of the devices at the edge (e.g., gateway, embedded systems) are often not leveraged. In this context, decentralized architecture has been proposed as a potential approach for the integration of the systems and for the process data analysis by clustering the assembly shop into subsystems and modules[4]. In this architecture, the decision-making process is delegated to the individual agents in the factory shop-floor instead of the central manufacturing system [5]. The approaches employing this decentralized architecture are designed based on distributed control in which individual systems react to their local condition in real-time [6]. These systems interact with their neighbors to form a network that displays the desired self-organizing behavior [7]. The decision is made by the systems individually based on the local condition and the collaboration with other systems.

Arrowhead Framework [8] is a cloud-based framework that supports such decentralized solution. Arrowhead Framework employs SoA to facilitate the secure communication of the devices and systems in a distributed manner. In Arrowhead Framework, the systems defined locally can exchange their data by producing and consuming services. However, there are heterogeneity issues originating from data generated by a wide variety of devices and machines. These issues are still not fully addressed in the decentralized approaches, and may therefore limit the collaboration between the entities in the industrial chain. Consequently, there is a growing demand for interoperability among the systems, processes, devices - down from the factory floor up to the enterprise and business levels [2]. Interoperability is the crucial element for optimizing manufacturing and improving productivity, reliability, and availability of the production systems in a heterogeneous environment.

Self-adaptation emerges as a promising solution to enable interoperable systems [9]. Self-adaptation enables the systems to exchange data with each other, analyze the data, and make appropriate decisions automatically. For example, this approach can be applied to condition monitoring tasks to determine the correctness of the operating states of the physical devices or manufacturing processes. Within traditional production systems, these tasks are usually performed by the highly experienced operator to prevent outage situations in case of machinery failure. However, due to the evolution of system architecture and functionality, these domain professionals may require more effort to adapt and may not be able to provide response precisely and timely to a machinery fault or failure [10]. For this reason, there is a need for a self-adaptation solution for IIoT systems that is capable of "harmonizing" heterogeneous data and providing reactive adaptation. This solution should be applied to any application domain while also enabling the integration of domain expert knowledge. However, As indicated in [9] such a full-fledged architecture for self-adaptation that combines SoA and IoT is still missing.

To address the challenges above, our work proposes an approach that introduces Autonomic Computing into decentralized control architecture. Autonomic Computing [11] is the concept introduced by IBM describing software systems that automatically observe and react to their operating environment. In our article, the MAPE-K (Monitor, Analyse, Plan, Execute, Knowledge) model introduced within Autonomic Computing is employed as a fundamental model to enable a generic self-adaptation solution. The adaptations will be provided as services to the application systems in order to orchestrate the communication among the systems or to configure their internal functionality. Furthermore, our approach enables the domain expert to integrate their knowledge into the process of generating adaptation under the form of high-level adaptation policies. To unify the heterogeneous data and infer valuable information for generating adaptations, Semantic Web Technologies have been exploited for transforming, analyzing, and reasoning over the data.

We make the following contributions:

1) We illustrate the use of Autonomic Computing in a service-oriented Industrial IoT framework that deals with heterogeneous data from various services and application systems and makes operating decisions automatically.
2) We propose a software engineering approach for enabling autonomic adaptation for configuration and orchestration of service-oriented industrial IoT application systems.
3) We lay out our vision that exploits the local automation cloud concept to migrate from traditional centralized architecture to decentralized architecture in order to achieve dynamic adaptation of manufacturing processes.

II. RELATED WORK AND TECHNOLOGIES

The concept of employing Autonomic Computing to achieve self-adaptation for different IoT domains has been proposed in multiple approaches. In [12], an approach that applied the MAPE-K model to perform automatic management of IoT architecture was proposed. In [13], the RoCoSys framework, which applied the MAPE-K model for coordination of Mobile IoT Devices, was used to make self-driving Mindstorms robots. The components of the MAPE-K model in those works are designed for the designated applications, thereby limiting their reusability for other domains. There were also Model-Driven Engineering (MDE) based approaches were presented in [14]–[17] where the adaptation is carried out by applying the concepts of models@runtime into the MAPE-K loop. Specifically, these approaches used the design model of the system as the semantic knowledge for generating automatic adaptations, which could make their solutions limited to specific applications since not every system has a complete design model. Furthermore, they also experienced the heterogeneity issue for transforming models from different Domain-Specific Modelling Languages (DSMLs).

Works providing autonomicity for Service-oriented environments have also been proposed. The authors of [18] proposed an approach for decentralized autonomic operations in microservices architectures (e.g., Kubernetes, Docker Swarm). In [19], Baylov *et al.* proposed a reference architecture for service-oriented systems based on the concept of an adaptation registry. Khazaei *et al.* [20] propose another solution providing adaptation as a service for microservice systems. Similarly, Maksuti *et al.* proposed a GAMS System (Generic Autonomic Management Service) that provided adaptation service within Arrowhead Framework [21]. The main focus was establishing the chain of trust and secure on-boarding procedure for the devices and systems within Arrowhead local cloud. Furthermore, Machine Learning (ML) algorithms were also employed to realize the MAPE-K model. For example, a self-healing microservice architecture that used ML to detect anomalies in docker containers was proposed in [22]. The authors in [23] introduce a distributed framework for autonomic resource management in the context of fog computing by applying support vector regression and reinforcement learning techniques for the analysis and planning phases. Other than ML approaches, rule-based solutions were also introduced. For example, the EXCLAIM framework [24] employed Semantic Web Technologies to monitor and process real-time data streams from cloud platforms and propose reconfiguration to address the scalability of the cloud. Cao *et al.* [10] also exploited Semantic Web for condition monitoring and failure detection in manufacturing processes. These solutions again targeted application-specific problems where the technologies were designed to fit the particular applications. As indicated in [9] a full-fledged framework or reference architecture for service-oriented IoT systems is still missing.

Distinguished from the works reported above, we propose a generic approach to achieve self-adaptation for service-oriented IoT systems. Specifically, we apply decentralized SoA as the primary concept to provide adaptation services generated by the MAPE-K feedback loop. To realize a generic MAPE-K solution that can be applied to any given domain, the Semantic Web Technologies and semantic modeling approach have been exploited for the implementation of the MAPE-K components. Furthermore, the Arrowhead Framework is used as the fundamental framework to realize our approach.

III. MOTIVATING SCENARIO

At Unger Fabrikker,[1] one of our partners in the Productive 4.0 project,[2] we developed a supply chain monitoring system that observed the consumption of the raw materials during the production. The monitoring system was also integrated with the ordering of the raw materials from the suppliers, and notifying the suppliers of upcoming deliveries. The solution consists of several gateways and sensors deployed to measure the level of the tanks that store the raw material.

Unger Fabrikker manufactures and exports a wide variety of surfactants, which are the main ingredients to produce household detergents, personal care, institutional products, and functional additives for industrial applications. Lye and Sulphur are two of the main ingredients for the production of surfactants at Unger Fabrikker. Typically, to make the orders for the raw materials, the operator at Unger Fabriker needs to keep monitoring the level of the tanks as well the production plan and must communicate with the suppliers to agree on the delivery plan (e.g., type and amount of the materials, delivery date). Furthermore, whenever there are changes to the production plan, the deliveries need to be updated accordingly within a reasonable time. The manual supervision and communication in the supply chain may result in longer time and inefficiencies in the production. Therefore, a system that constantly monitors the production status is useful in order to understand how these conditions might affect the consumption of raw materials, make or update the orders for the suppliers, notify or take any other actions automatically. A straightforward solution is to hard-code a set of rules for corresponding actions. For example, if the level of the raw material is running low according to the current production plan, a new delivery needs to be made before the tank becomes empty. As another example, whenever the sensor delivers malicious measurement values, the system should use the other sensors as an alternative or notify the operator for the replacement of a new sensor. However, rules and actions vary for different kinds of materials, suppliers, or even periods of a year. As a result, whenever new requirements arise, the system needs to be reprogrammed. Therefore, hard-coding does not scale well and is not efficient for new problems or new business rules.

From this scenario, it is clear that the operator wants a solution that provides him more flexibility to address new situations that were not envisioned during the initial setup of the system. Therefore, it needs to be dynamic, flexible and should not involve any reprogramming, redeployment, or hard reset of the devices. Also, it should be usable even by non-IT experts. In this work, our proposed solution supports the implementation of a system that does not require reprogramming whenever new requirements or new business rules are introduced. Specifically, the solution contains the following characteristics:

1) Expressing the conditions, requirements, and corresponding actions of the business rules explicitly and precisely.
2) Efficient deployment strategies of these business rules.
3) Efficient execution of these business rules.
4) Portable on different platforms (e.g., gateway, edge, fog node, Raspberry Pi, etc.).

[1] [Online]. Available: http://www.unger.no
[2] [Online]. Available: https://productive40.eu

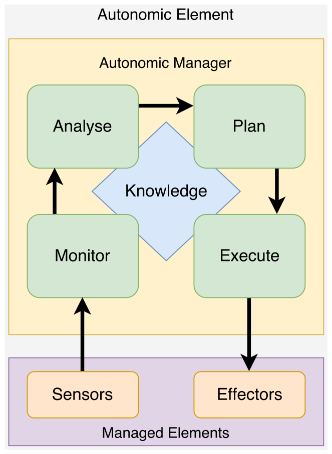


FIGURE 1. IBM's MAPE-K reference model for autonomic control loops.

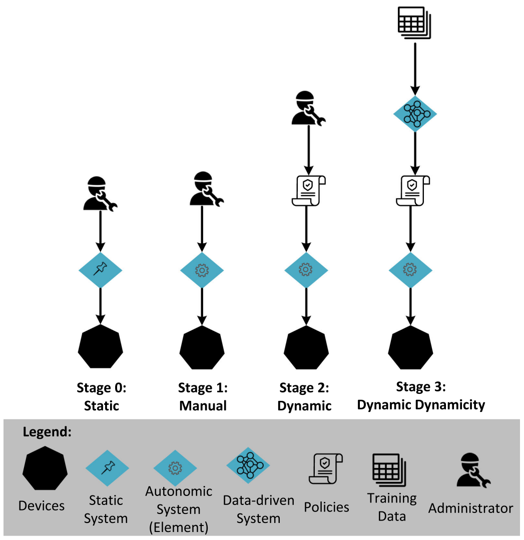


FIGURE 2. Four stages of autonomic adaptation.

5) Independent of physical devices type (e.g., sensors, actuators), communication protocols, and application domain.

IV. BACKGROUND

A. AUTONOMIC COMPUTING

Autonomic Computing [11] was first introduced by IBM based on the human autonomic nervous system. Its overarching goal is to realize software systems that can manage themselves in accordance with high-level guidance from humans [25]. In our work, Autonomic Computing is employed as the enabling technology for the implementation of our adaptation solution.

In order to achieve Autonomic Computing, IBM also proposed a reference model for autonomic control loop [26], which is also referred as MAPE-K (Monitor, Analyse, Plan, Execute, Knowledge) loop and is depicted in Fig. 1. In the MAPE-K autonomic loop, the *Managed Element* represents any software or hardware resource that is provided with autonomic behaviors by coupling it with an *Autonomic Manager*. *Sensors*, also called probes or gauges, collect information about the Managed Element whereas *Effectors* carry out changes to the Managed Element [27] . The Autonomic Manager is the core software system that provides autonomicity to the Managed Elements. Particularly, the Autonomic Manager keeps monitoring the status of the Managed Element and generates adaptation whenever needed. The behavior of the Autonomic Manager is modeled with five core components: the *Monitor* component collects the data about the Managed Element via Sensors, the *Analyze* component extracts critical information from the collected data, the *Plan* component generates guiding actions for the Managed Element based on the inferred critical conditions, and the *Execute* component carries out the actions with the support of Effectors and the *Knowledge base*, the central component for storing all generated information.

B. FOUR STAGES OF AUTONOMIC ADAPTATION

An autonomic system can be designed by employing the MAPE-K model. According to Fig. 1, the autonomic system can be modeled as the Autonomic Element which consist of two components: the Autonomic Manager and the Managed Element. Additionally, the autonomic system is also provided with high-level adaptation policies as the guidance for adaptation goals. In this section, we depict different approaches to autonomic adaptation for software systems and summarize them into four stages as illustrated in Fig. 2.

1) **Stage 0 - Static**: This is the early software system that was developed with static configuration (e.g., hard-coded setpoints). In the case of unexpected events, the systems may require reprogramming and redeployment in order to adapt to the current condition.
2) **Stage 1 - Manual**: The systems are enhanced with configurable capabilities (e.g., via a configuration file or an interactive user interface to update the set points). The configurations are set manually by an administrator. At this stage, the systems are designed to adapt to particular situations anticipated in those predefined configurations. This stage could be considered as the most common solution as it introduces flexibility at a certain level without introducing complexity to the implementation of the systems. At this stage, human intervention is still required when system calibration is needed. Also, the systems may need to be redesigned in order to adapt to the changes that cannot be done via reconfiguration.
3) **Stage 2 - Dynamic**: The systems at this stage are designed to react to particular conditions that are specified by a set of policies. This stage can be considered as the first stage where the systems are self-managed. The term 'policy' refers to a structured set of rules and actions that govern the behaviors of the system during run-time. Policies can be defined either implicitly (e.g,

[28]) or explicitly (e.g., [29], [30]) in the design of the systems, and be specified either statically (e.g, [31], [32]) at design time or dynamically during run-time (e.g, [29], [30]). These systems are designed to cope with environmental changes that cannot be anticipated completely during design time. Although such designs may increase the complexity of the systems (e.g., by introducing adaptation logic into the original application logic of the system), there has been increasing interest in self-adaptive approaches because of the increasing cost related to managing systems in dynamic environments [33]. The common aspect of the systems at this stage is the reliance on the user for defining and maintaining the policies.

4) **Stage 3 - Dynamic Dynamicity**: At this stage, the adaptation solution is enhanced with the capability to self-managing its policies. Specifically, these approaches exploit a 'learning component' to disable inefficient or invalid policies or to add new policies that handle unforeseen situations. Therefore, these solutions may introduce more complexity to the adaptation logic of the system. Also, the 'learning component' of these systems is implemented based on machine learning or artificial intelligence algorithms with the learning data from either the design models of the system or the run-time data (i.e., [34]–[36]). As a result, these approaches usually address domain-specific problems or handle particular situations derived from the training data.

C. ARROWHEAD FRAMEWORK

Arrowhead Framework [8] is a service-oriented IIoT framework that exploits the design of private cloud architecture to facilitate the development of local automation clouds. The concept "local automation cloud" is based on the idea that certain automation tasks should be encapsulated and protected based on their geographical organizations [37]. Therefore, the local automation cloud basically involves devices and systems needed to perform the automation tasks, thus forming a local "room" that protects against unwanted external influence. These local clouds may have secure interaction with other local clouds, thus constituting a System of Systems (SoS) architecture. Within Arrowhead Framework,

1) A *Service* contains the information about the interface (e.g., communication protocol, syntax, and semantics applied, authentication and authorization mechanism) that defines how the service provider and consumer should interact.
2) A *System* is a software artifact that either provides or consumes services. There are two types of systems: core systems which provide various administration of the local louds, and application systems which implement the automation logic.
3) A *Device* could be any equipment, machine, or computer that can deploy one or more systems.

Fig. 12 illustrates an Arrowhead automation local cloud that consists of the three mandatory core systems—Service

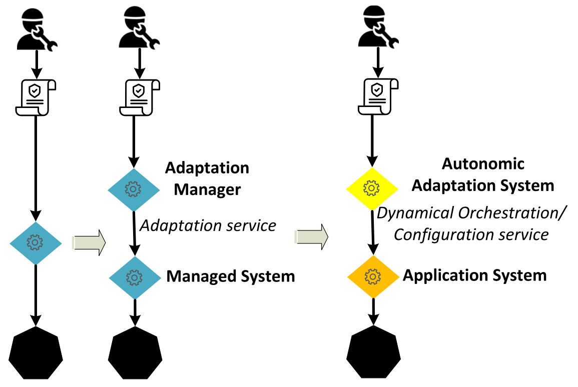

Adaptation Manager
Adaptation service
Managed System
Autonomic Adaptation System
Dynamical Orchestration/ Configuration service
Application System

FIGURE 3. Autonomic Adaptation Solution within SoA.

Registry, Orchestration, and Authorization Systems (color-coded as blue, green, and red), the support core system (color-coded as yellow)—Autonomic Adaptation System, and the application systems (color-coded as orange). The three core mandatory core systems are required to construct a minimal Arrowhead local automation cloud.

1) The Service Registry is the central record of the published services. It supports the service producer to register its services and the service consumer to discover the published services.
2) The Authorization System provides mechanisms for authenticating the application systems and authorizing service consumption.
3) The Orchestration System establishes and maintains the interactions among the service producers and service consumers.

In addition to those mandatory core systems, Arrowhead Framework also consists of various support core systems that facilitate both design-time and run-time development of automation systems with either intra- or inter-cloud interaction. As examples of such support systems we have the GateKeeper and Gateway [38] for inter-cloud service interaction, the Configuration system to store configurations of the systems, and the Data Manager for short-time or long-time application data storage. These core systems together enable a framework with *lookup*, *loose-coupling* and *late-binding* properties for engineering IIoT SoS satisfying the requirements for real-time performances, robust security, and interoperability. [37].

V. AUTONOMIC APPROACH

A. AUTONOMIC ADAPTATION SYSTEM

In this section, we discuss our proposed architecture that supports the development of IIoT applications and addressing the *dynamic adaptation* requirements mentioned above.

Definition 1: **Dynamic Adaptation** indicates the ability of an application system to modify its behaviors during its execution as a reaction to the changes of its operating environment.

In order to achieve dynamic adaptation, the model in stage 2 of autonomic adaptation is employed as the fundamental design of our approach. Fig. 3 illustrates our conceptual design

```
{
  "response" : [ {
    "service" : {
      "serviceDefinition" : "Service1_Tank1",
      "interfaces" : [ "JSON" ],
      "serviceMetadata" : {
        "unit": "Tons",
        "location": "Tank1",
        "feature": "Lye Level"
      }
    },
    "provider" : {
      "systemName" : "Tank1_Service1",
      "address" : "192.168.1.45",
      "port" : 8460
    },
    "serviceURI" : "level",
  } ]
}
```

FIGURE 4. Example payload of orchestration service.

```
{
  "systemName": {
    "systemName": "SupplierManagementApp",
    "contentType": "text/plain",
    "data":
    "CreateOrder:
            Supplier1-Lye-1602510142027-40"
  }
}
```

FIGURE 5. Example payload of configuration service.

within SoA. Accordingly, by employing SoA, we separate the application system in stage 2 into two sub-systems: the **Adaptation System** that interprets the user policies and provide corresponding adaptation as a service, and the **Managed System** that implements the core functionality and reacts to the changes under the coordination of the adaptation service. As a result, this design reduces the complexity of the original application system by separating the implementation of the adaptation support from the core functionality while also introducing flexibility to the deployment of the systems by employing SoA.

To realize the conceptual design in Fig. 3, we employ the Arrowhead Framework as the fundamental architecture for our solution. We propose one support core system for the Arrowhead framework, namely **Autonomic Adaptation System**. This system plays the role of the Adaptation Manager in our conceptual design in Fig. 3 and enables the application systems (a.k.a. Managed Systems) with dynamic adaptation under the form of *dynamical orchestration and configuration services*. As shown later, these two services are implemented by applying the core services of Orchestration and Configuration systems of Arrowhead Framework. From this section, for simplicity, we will use the term "adaptation" to refer to dynamical orchestration and configuration.

Definition 2: **An Orchestration** is a message containing the information about the service and its corresponding provider system which is used by the consumer system to establish a connection to that provider system.

Fig. 4 shows an example of an orchestration of a RESTful service which consists of the service unique id (*serviceDefinition*), exchanging formats (*interfaces*), metadata as well as endpoint of the server (*provider*).

Definition 3: **A Configuration** is a message containing the actual parameter values for the application system to adjust its internal functionality.

An example of such configuration is showed in Fig. 5 which instructs the application system (identified by the *systemName*) to create a new order with details provided in the *data* field.

Definition 4: **Orchestration Service** is the service that coordinates the communication among application systems by providing appropriate Orchestrations to the requesting consumer systems.

Definition 5: **Configuration Service** is the service that provides Configurations (e.g., parameter property) to the application systems.

Orchestration service is used to control how systems are deployed and interconnected, while configuration service enables the systems to change their internal functionality during run-time. Within Arrowhead Framework, these two services are provided by the Orchestration and Configuration systems. Specifically, the service consumer can query information about its service of interest by providing the exact name of that service to the Orchestration System. Subsequently, the Orchestration System will retrieve the corresponding information from the Service Registry and the Authorization System, and compose a list of authorized services and their corresponding service providers. This mechanism is implemented by string matchmaking and filtering over the service namespace. If there is more than one provider in the return list of orchestrations, the consumer can decide which one to consume the service. Alternatively, the Orchestration System also maintains an *Orchestration Store* that contains static orchestrations for the consumer system to connect (e.g., specified manually by the administrator of the system). This static approach can be used to initialize or orchestrate the system whenever the name of the service is unknown during the design time. Similarly, the Configuration System also maintains a *Configuration Store* for configuring the applications statically.

In this work, we move one step further by providing dynamical orchestration and configuration mechanisms to Arrowhead Framework. Particularly, the contextual information of the corresponding services and application systems will be considered to generate their orchestrations and configurations automatically. This mechanism is realized with Orchestration and Configuration policies.

Definition 6: **Orchestration or Configuration policy** is the rule (if-then-else statement) specifying how an application system should be orchestrated or configured under a specific condition.

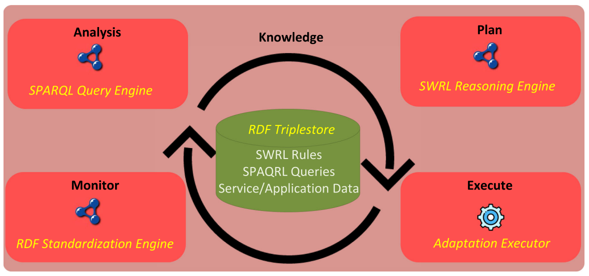


FIGURE 6. **The Service-oriented Autonomic IoT (SAI) platform. Adapted from [39].**

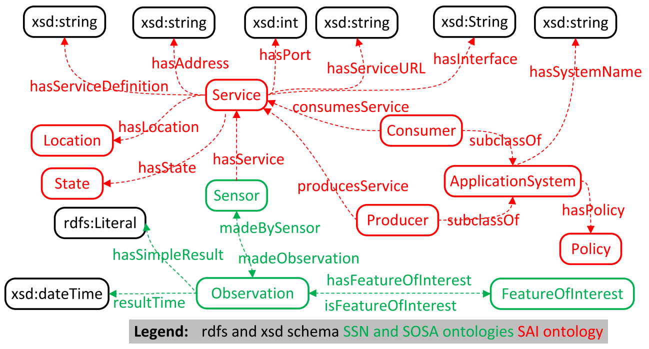


FIGURE 7. **Service, System and Sensor models in the SAI ontology.**

The Autonomic Adaptation System will exploit these policies to generate orchestration and configurations and push them to the Orchestration and Configuration Stores accordingly. Accordingly, the Autonomic Adaptation System enables the application systems to automatically re-orchestrate their service consumption or reconfigure their settings in reaction to the changes of other systems and services within one Arrowhead local cloud.

The core functionality of the Autonomic Adaptation System is designed based on the concept introduced in Autonomic Computing (discussed in Section IV-A). Specifically, our proposed approach uses the MAPE-K model as the fundamental model for the implementation of the Autonomic Adaptation System. In order to realize the MAPE-K model, we employ the SAI (Service-oriented Autonomic IoT) platform - our previous work presented in [39], [40]. The SAI platform exploited different tools and technologies from the *Semantic Web* to implement the core functionalities of the MAPE-K model. Fig. 6 illustrates the usage of Semantic Web Technologies to implement the SAI platform. Specifically, we employed the Jena RDF (Resource Description Framework) Triplestore[3] as the central Knowledge Base (KB) to store the monitoring data and policies specified with SWRL (Semantic Web Rule Language) and SPARQL queries (SPARQL Protocol and RDF Query Language). The SAI ontology [40] is used as the common schema for semantic data representation and validation. This ontology was developed based on different well-known IoT ontologies such as IoT-O [41], SOSA [42], SSN [43]) to specify concepts of devices, sensors, observation, and actuators. Fig. 7 shows an extract of the SAI ontology containing the models of Sensor, Arrowhead Service, and Application System. Furthermore, the Monitor component collects the data from the services and application systems, transforms them into RDF triples, and updates to the KB. Fig. 8 show an extract of the KB representing the data from the sensor that measures the level of a Lye Tank. The SAI ontology is used as the common vocabulary to annotate and integrate the sensor value into its respective service in the Arrowhead local cloud. The Analysis component exploits the SPARQL Query Engine to filter the data in the KB for

[3][Online]. Available: https://jena.apache.org/

```
:Tank1_Service1
  a                      :Service ;
  :hasServiceDefinition "Tank1_Service1" ;
  :hasState              :OnlineState ;
  :hasLocation           :Tank1 .

:Tank1_Producer1
  a                  :Producer ;
  :producesService   :Tank1_Service1 .

:Tank1_Sensor1
  a                  :Sensor ;
  :hasService        :Tank1_Service1 .

:Observation1_Tank1_Level
  a                          sosa:Observation ;
  :hasUnit                   "Tons" ;
  sosa:hasFeatureOfInterest :"Lye Level" ;
  sosa:hasSimpleResult       "36.09"^^xsd:double ;
  sosa:madeBySensor          :Tank1_Sensor1 ;
  sosa:resultTime            :1602602232000 .
```

FIGURE 8. **A snapshot of the KB that annotates data from the sensor of a Lye Tank.**

only critical conditions about the monitoring services or applications and updates the querying results to the KB. Fig. 9 shows the SPARQL query that periodically checks the timestamp of the sensor observation and updates the status of the corresponding sensor service to *OfflineState* if the value is not updated within 20 seconds. As can be seen from this figure, the SAI ontology is also used as the language to describe the query. Furthermore, the Plan component will reason over the KB for any relevant adaptations which will then be carried out by the Execute component. Specifically, the Plan component exploits the adaptation policies as the instruction for generating adaptations. In this work, we use SWRL to specify those adaptation policies. Fig. 10 illustrates a SWRL rule instructing the application to re-orchestrate (*substituteService*) to an “Online” service if its consuming service is “Offline”. Similar to the SPARQL queries, the SWRL rules also exploit the SAI ontology to specify the policies. These queries and policies are also stored inside the KB as shown in Fig. 11.

The SAI platform was discussed in our previous work [39]. Here, we move one step further by adapting the SAI platform

```
DELETE {
 ?service :hasState :OnlineState .
}
INSERT {
 ?service :hasState :OfflineState .
}
WHERE {
 ?sensor rdf:type :Sensor .
 ?service rdf:type :Service .
 ?observation rdf:type sosa:Observation ;
              sosa:madeBySensor ?sensor .
 ?sensor :hasService ?service .
 ?observation sosa:resultTime ?time .
 ?service :hasState :OfflineState .
 :DateTimeNow :hasValue ?now
 FILTER (?time > ( ?now - 20000 ))
}
```

FIGURE 9. A SPARQL query to update state of the service.

```
(?c rdf:type :Consumer)
 ?c :consumesService ?s1)

(?p1 rdf:type :Producer)
(?p1 :producesService ?s1)

(?p2 rdf:type :Producer)
(?p2 :producesService ?s2)

(?d1 :hasService ?s1)
(?d1 sosa:hasLocation ?l)
(?obs1: soasa:madeBySensor ?d1)
(?obs1: soasa:hasFeatureOfInterest ?f)

(?s1 :hasState :OfflineState)


(?d2 :hasService ?s2)
(?d2 sosa:hasLocation ?l)
(?obs2: soasa:madeBySensor ?d2)
(?obs2: soasa:hasFeatureOfInterest ?f)

(?s2 :hasState :OnlineState)
->
substituteService(?c, ?s1, ?p1, ?s2, ?p2)
```

FIGURE 10. Examples of an orchestration policy to substitute from one Offline sensor service to an Online one at the same location and feature of interest.

```
:SupplierManagementApp
  a                      :Consumer ;
  :consumesService       :ServiceTank1 ;
  :hasPolicy             :SensorDisconnect ;

:SensorDisconnect  :hasBody "<The_SWRL_Rule>" .
```

FIGURE 11. A snapshot of the KB that annotates the consumer and its policies.

to implement the Autonomic Adaptation System in order to achieve dynamical adaptation within Arrowhead local cloud. Fig. 12 illustrates the interaction of the Autonomic Adaptation System with the application systems via Arrowhead services.

As can be seen from Fig. 12, the SAI platform, applying the MAPE-K model, is the core component of the Autonomic Adaptation System to implement self-adaptation. In addition, the Autonomic Adaptation System is also equipped with different *Application Service Adapters* to support the Monitor component to parse and transform the messages from the monitoring services. At the current implementation of the Autonomic Adaptation System, the adapters only accept service interfaces employing SenML[4] specification with either XML or JSON format. The numbers in Fig. 12 represent the order of information flow of the adaptation control loop. To be monitored by the Autonomic Adaptation System, the Application Systems have to send their data (conforming to the SenML format) to the Data Manager system which acts as the central storage for real/time monitoring data (1). The Autonomic Adaptation System will use these data from the Data Manager (2), transform, and update the KB as presented above. The Autonomic Adaptation System will also generate Orchestrations and Configurations based on the specified policies and push them to the Orchestration and Configuration Stores respectively (3). Furthermore, those Orchestrations or Configurations will be sent to the corresponding application systems for re-orchestration or reconfiguration (4). To carry out the Orchestration or Configuration, these application systems need to implement an `Adaptation Executor` to analyze the Orchestrations or Configurations and update their functionalities accordingly. The Arrowhead Framework already provides libraries to parse the Orchestration and Configuration showed in Figs. 4 and 5. The application system can exploit those libraries and implement the methods to handle the information accordingly. We do not define any format for the configuration value (e.g., the `data` field of the configuration). It is left to the application layer to design syntax or semantics for this field.

B. AUTONOMIC ADAPTATION SYSTEM AUTOMATES RECONFIGURATION OF MANUFACTURING PROCESS

Manufacturing Execution System (MES) acts as an intermediate layer between the information systems of high-level enterprise applications and the tightly coupled systems on the shop floor which manage the low-level data generated by field devices. Through the control loops between the ongoing manufacturing processes and their respective control instances (i.e., Programmable Logic Controller), MES improves the overall process efficiency in terms of quality, timeliness, and maintenance aspects. Fig. 13 illustrates such organizational approach in current automation systems. Traditionally, the field devices (i.e., robots, sensors, actuators) on the shop floor are controlled by the automation systems built with the Programmable Logic Controller (PLC). These systems contain all necessary programs, sequences, and logical correlations for the manufacturing steps performed by the field devices. Furthermore, certain data generated by these field devices might be collected, aggregated, and shown within supervisory control and data acquisition systems (SCADA) for diagnostics or analytical purposes. SCADA usually collect information

[4][Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-core-senml

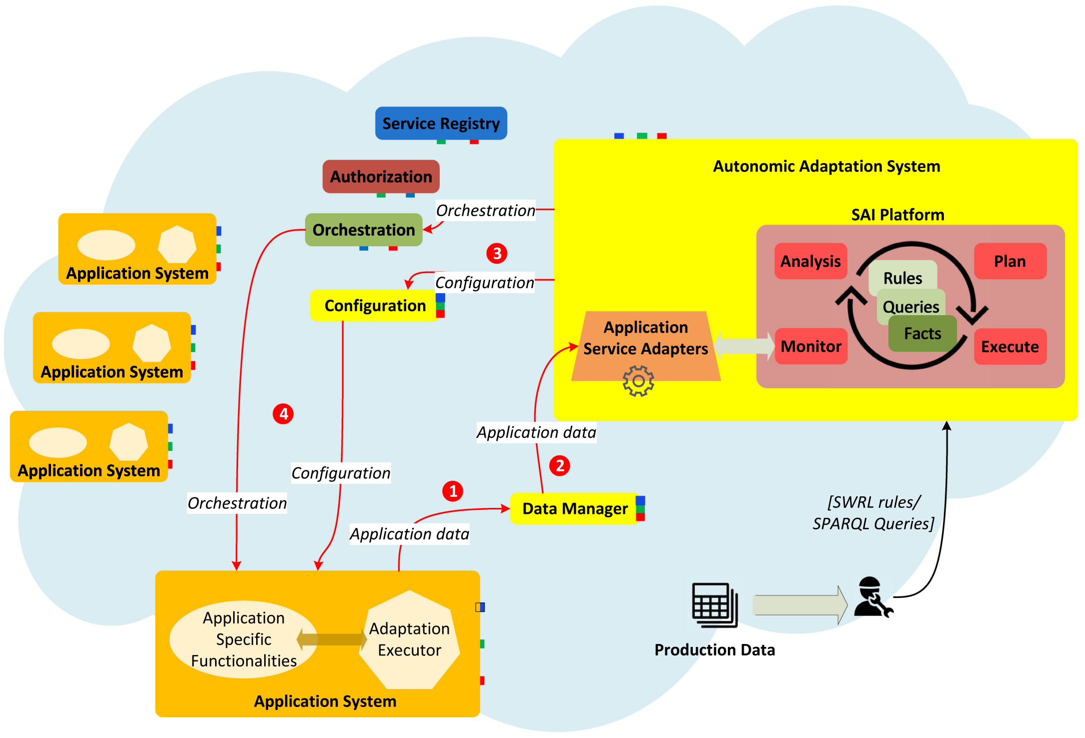


FIGURE 12. Interaction of the Autonomic Adaptation System with other systems within Arrowhead local cloud. The numbers represent the order of the communication flow.

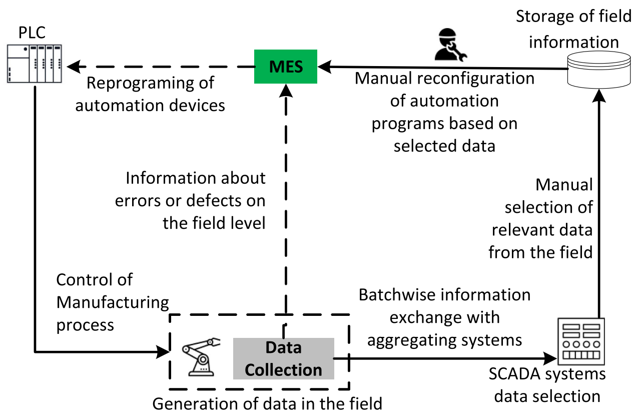

PLC
Reprograming of automation devices
MES
Storage of field information
Manual reconfiguration of automation programs based on selected data
Information about errors or defects on the field level
Manual selection of relevant data from the field
Control of Manufacturing process
Batchwise information exchange with aggregating systems
Data Collection
Generation of data in the field
SCADA systems data selection

FIGURE 13. Production organization in current automation systems (Adapted from [44]).

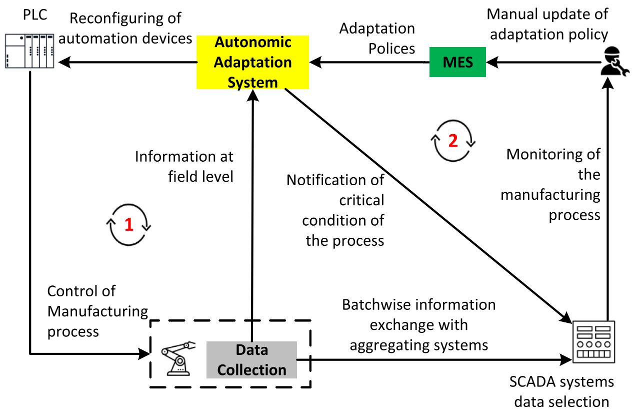

PLC
Reconfiguring of automation devices
Autonomic Adaptation System
Adaptation Polices
MES
Manual update of adaptation policy
Information at field level
Notification of critical condition of the process
Monitoring of the manufacturing process
1
2
Control of Manufacturing process
Data Collection
Batchwise information exchange with aggregating systems
SCADA systems data selection

FIGURE 14. Production organization with Autonomic Adaptation System.

about the ongoing process such as average time of the production steps, deviations of manufacturing tolerances, or similar KPI (Key Performance Indicator) and forward such information to higher systems (i.e., database server) for later tactical or strategic decisions regarding the production planning and control. Moreover, this information could be used as reference data for diagnosing manufacturing errors and making decisions accordingly. Typically, whenever errors occur (e.g., caused by operating environment, material faults, etc.), the MES may stop the process or re-initialize the production in accordance with human interaction with the system configuration [44]. However, due to human-based interaction, the cycle time of these feedback loops is too long [44]. Furthermore, in most cases, the exchanged data lacks context and other descriptive information and does not fit the need for automatic decision-making. As a result, the processes of extracting relevant information for utilizing the manufacturing processes are usually manual and human-based. Therefore, the aforementioned feedback loops could not be considered for short-term planning and optimization scenarios.

The Autonomic Adaptation System facilitates the feedback loop between the field and the managing instances of the automation system in a semi-automated manner. Fig. 14 illustrates such production organization with the support of the Autonomic Adaptation System. It automates the processes of reconfiguration of the automation systems in order to achieve self-healing or self-optimization. Particularly, by collecting, transforming, and integrating the field data with necessary contextual information, the Autonomic Adaptation System monitors the status of manufacturing processes and makes decisions automatically whenever anomalies occur or other actions need to be taken. These decisions could be either short-term goals such as prioritizing certain products or optimization of a single process or even mid-term and long-term goals at the management level. However, human interaction cannot be completely eliminated from the feedback loop. In particular, the Autonomic Adaptation System needs to be supervised by the process operators under the form of policies for when and how to make adaptations. This second feedback loop between the Autonomic Adaptation System and high-level systems is rather slow but will not occur frequently. Once the policies are updated in the Autonomic Adaptation System, and subsequently triggered, reconfiguration will be executed immediately without any human intervention.

C. PROPERTIES OF ARROWHEAD LOCAL CLOUD WITH AUTONOMIC ADAPTATION SYSTEM SUPPORT

In the traditional approach, knowledge localization and manipulation are implemented far away from where the data is generated (i.e., physical devices), whereas the private local cloud approach brings the knowledge representation to the device level, thereby decentralizing the knowledge closer to the knowledge-driven system [45]. In our research, we develop this private cloud approach further by introducing autonomicity into the local cloud.

The approach discussed in this article employs the local automation cloud concept within Arrowhead Framework but goes one step further by leveraging the knowledge to detect anomalies of the systems in the cloud and generating re-orchestration or re-configuration accordingly. Table 1 shows the different aspects of this approach in comparison with the traditional approach and the Arrowhead local cloud approach. The categories of this table are obtained based on the benchmark made in [45] and supplemented with the approaches discussed in our study.

As can be seen from Table 1, the first direct benefit of the new system is *autonomic adaptation*. While the original system permits only one static configuration (e.g., consuming only one set of sensor units). Traditionally, the connection between the automation systems and the field devices is usually decided at design time and hard-coded during the development of the systems. This limits the complexity of the systems, but it also limits their flexibility during execution time. As a result, the static configuration requires reprogramming and redeploying the system whenever there are updates to the design or configuration of the systems. In contrast, within Arrowhead local cloud, the application systems can be re-orchestrated or reconfigured during their run-time based on multiple preferences stored in the Orchestration and Configuration systems. Therefore, application systems become loosely coupled with the devices at the lower layer. However,

TABLE 1. Properties of Arrowhead Local Cloud With Autonomic Orchestration System Support (Extended from [45])

	Traditional	Arrowhead local cloud	Arrowhead local cloud with Autonomic Adaptation System
System Adaptation	Static	Manual	**Autonomic**
Communication protocols	Open and Proprietary	Open	Open
Plug and play	No	Yes	Yes
IT infrastructure	Poorly reusable	Highly reusable	Highly reusable
Control architecture	Centralized	Decentralized	**Decentralized Autonomy**
Control model	Scan based	Scan based/Event based	Scan based/Event based
Addressability	Hierarchical	Direct	Direct
Data processing	At global central unit	At local cloud level	At local cloud level
Number of computational nodes	1 (centralized architecture)	Many (decentralized architecture, at multiple local clouds)	Many (decentralized architecture, at multiple local clouds)
Exchange data	Signal data	Information (annotated data)	Information
Message readability by humans	No	Yes	Yes
Message overhead	No	Yes	Yes
Data representation (at computational nodes)	Heterogeneous	Heterogeneous	**Autonomic alignment of heterogeneous data representation**
Communication protocol overhead	No	Yes	Yes
Application implementation complexity	Low	High	**Reduced**
Horizontal and vertical integration	Limited	Yes	Yes
Monitoring and Adaptation	Human based	Human or specific monitoring and analysis tools	**Autonomic monitoring and adaptation**
Cyber security	Obscurity	At device, system, service, cloud levels	At device, system, service, cloud levels

adaptation within Arrowhead local cloud is still manual since those preferences of orchestration and configuration are added by the operator of the systems and need to be updated with the evolution of the whole system. The Autonomic Adaptation System addresses this issue by taking into account the context in which the systems are executing and generating adaptations accordingly. This is achieved by enhancing manual adaptation with adaptation policies. By satisfying these policies, the Autonomic Adaptation System will decide the optimal adaptation that fits the most recent status of the systems.

The *communication protocols* in the traditional approach are basically proprietary and made for particular device types. As the systems evolve, the devices may need to be replaced or new devices added. Thus, these systems have to be reconfigured (or even reprogrammed) manually in order to adapt to these changes. Within Arrowhead local cloud, by exploiting the SoA concept, the communication is abstracted into services that typically employ open and standardized protocols. Specifically, Arrowhead Framework provides multiple core services that support the communication with physical devices via various industrial protocols (e.g., Modbus TCP, OPC-UA). These services act as adapters between the devices and the application systems and can be re-used for new devices. Although unsupported protocols require new adapter implementations, the application systems do not need to be reprogrammed but instead re-orchestrated. In our approach, with the support of dynamic adaptation of the Autonomic Adaptation System, changing physical devices can be described as adaptation policies and carried out automatically.

Within the traditional approach, due to the diversity of communication protocols between the automation system and devices, *Plug and Play* feature is not supported. Within Arrowhead local cloud, by employing the SoA paradigm, the connection between the systems and devices becomes loosely coupled. As mentioned earlier, the application systems need to be re-orchestrated to work with newly added devices. This could be done manually, and the application system may need to be restarted to update the new orchestration. With the support of the Autonomic Adaptation System, once the policies regarding new devices are registered, new orchestration can be generated and pushed to the appropriate application system automatically.

IT infrastructure regarding both hardware and software is poorly reusable in the traditional approach as the systems are usually designed and implemented to work with designated devices. Therefore, replacing or adding new devices requires re-configuring and re-programming the systems. In contrast, the Arrowhead framework aims at loosely coupled systems. As a result, the software components (e.g., services, application systems) could be exploited to compose a new system of systems. Also, as discussed earlier, physical devices could be easily integrated and interact with the systems unambiguously and automatically. Therefore, the reusability of both systems and devices is highly supported during the development of new systems.

The *control architecture* in traditional approach is purely centralized. Typically, the automation systems collect and send data to one central server which performs processing and analyzing of the data. Based on the analysis, the decision will be made and sent back to those systems. This architecture is simple and easy to deploy and maintain as all necessary operations are managed in one place. However, the computing resources of modern devices at the edge are not fully leveraged. Also, security and privacy risks could be introduced because the data could be sent over the network (e.g., to a global cloud) for processing. Additionally, response time could be longer due to the network latency or high workload at the central system. These issues are addressed in the Arrowhead framework which employs decentralized architecture. Particularly, each Arrowhead local cloud encapsulates all the application systems and devices that are deployed within one local area (e.g, one factory floor, one room of a building). In each local cloud, there could be one central system that collects data from the others and analyzes the data locally. The data can also be sent to other local clouds if necessary for making the final decisions. With the support of the Authorization and Certificate Authority services, data privacy and security are ensured for both intra- and inter-cloud communication [46]. Within one Arrowhead local cloud, any system could be implemented as the central node which collects and analyzes the data. However, as new systems and services are introduced to the local cloud, this central node has to be updated by reprogramming or re-configuring manually. In our autonomic approach, by employing the MAPE-K model, the decentralized architecture is enhanced with autonomic control. The Autonomic Adaptation System is responsible for monitoring and generating the decisions, whereas the application systems specify how decisions will be executed in the form of adaptations. With semantic technologies, the data will be annotated and standardized to support automatic decision-making. As a result, the actions based on the generated decision will be carried out automatically without any interruptions or manual operations.

The *control model* in traditional automation systems is usually scan-based. In principle, these systems have to poll the central node periodically to get updated with the latest control decision. This method may waste time and resources spent on communication and computation when no significant change in the system has occurred. In contrast to this scan-based model, in the event-based model, the systems will be triggered whenever there are changes that need to be updated. Arrowhead framework supports both models. Specifically, depending on their designs, the systems could be implemented to keep polling the information periodically, or the information will be pushed to those systems once it is generated. In our approach, the communication between the Autonomic Adaptation System and the monitored system is designed based on an event-driven model. Therefore, the application systems have to implement corresponding callback functions to handle those triggered events.

Regarding the *addressability* of the resources (e.g., sensor data), within the traditional approach, it is designed as a hierarchical model. The data from the field devices have to go through the systems through multiple levels (e.g., PLC, SCADA, MES) in order to be analyzed. Accordingly, the instructions from the systems at higher levels will also go through the intermediate level before arriving at the devices. Within Arrowhead local cloud, the data can go directly to the designated systems with the support of service discovery. As all operations are abstracted into services, the application can query over the Service Discovery to look for the desired service providers and start the communication immediately.

Within the traditional approach, *data processing* will be performed at a global central unit (e.g., global cloud which provides storage and analysis services). In contrast, within Arrowhead local cloud, the data will be collected and analyzed locally by any authorized system. Therefore, the *the number of computation nodes* will increase with the number of systems and local clouds. As a result, the approach can not only utilize the computational capacities of the local machines on which those systems are deployed but also mitigate the security and latency issues related to the global network communication.

Furthermore, the *exchange data* within the local cloud is annotated with semantic description which becomes information instead of pure signal data as in the traditional approach. Since the operations of the field devices are now abstracted into services and communication between the systems becomes producing and consuming services, the raw signal data would be annotated with metadata at the service level. Therefore, messages become more *human readable*, thus reducing the effort related to diagnostic and maintenance of the systems. However, the approach increases *message overhead* as the exchange messages will contain additional information related to syntactic and semantic annotation.

In both traditional approach and Arrowhead local cloud, the *data representation* at the computational nodes that collect and process the data vary relative to the domain of the application. Therefore, the data are exchanged and stored with different syntax and semantics from different systems. This heterogeneity problem is addressed in our approach. By employing semantic technologies, the data collected from heterogeneous service interfaces are transformed, annotated based on the proposed semantic ontology, and stored in RDF triplestore. Thus, the application systems may use the ontology as a uniform vocabulary to describe their policies and communicate with the Autonomic Adaptation System.

Arrowhead Framework may introduce *communication protocol overhead* as the application systems need to interact with the core systems for authorization, orchestration, or other management purposes. By employing Arrowhead architecture, our approach also requires the application systems to follow the same protocols as described in Fig. 12 in order to achieve dynamic adaptation, thereby introducing more interaction among the application systems at the service level. However, as the local cloud is designed to coordinate the application systems and devices in close proximity, the latency introduced by protocol overhead is expected to be insignificant in comparison with the global cloud architecture.

By separating the monitoring and reasoning tasks from the main application functionalities, our approach with the support of the Autonomic Adaptation System helps to reduce the *application implementation complexity*. In traditional automation systems or usual Arrowhead application systems, adaptation features could be achieved either manually or by integrating the solutions into the implementation of the application. The former requires manual effort and may result in a longer adaptation cycle, while the latter introduces more complexity to the application logic and therefore requires development and maintenance effort. The Autonomic Adaptation System automates the process of monitoring and adaptation reasoning and provides adaptation as a service. Therefore, the application systems only need to execute the adaptations accordingly. Also, these applications have to follow the communication protocol as introduced earlier. However, the overhead related to the implementation of the communication and processing adaptations is expected to be smaller than the monolithic solution.

Regarding the *horizontal and vertical integration* aspect, by employing the SoA concept with unified service management operations, Arrowhead Framework not only enables the easy integration of the devices and systems in one layer but also improves the interaction of the systems from different layers of the ISA-95 model. Furthermore, both the traditional approach and Arrowhead Framework have limited support of *monitoring and adaptation* whereas this feature is the core functionality of the Autonomic Adaptation System. Finally, by employing Arrowhead core systems, authentication and authorization are implemented at several levels: physical device level, service level, system level, and cloud level, and thereby improving the *cyber security* of the system.

VI. CASE STUDY

In this section, we will demonstrate our proposed approach by a use case aiming to improve the supply chain management system at Unger Fabrikker. We will introduce high-level policies to automate the process of monitoring and planning for supplying the raw materials during the production at Unger Fabrikker.

Unger Fabrikker has several suppliers of raw materials (e.g., lye, sulfur). The materials are stored in multiple tanks. These chemical supplies require extensive planning by both suppliers and Unger Fabrikker. To ease the entire management of the supply chain, the Autonomic Adaptation System, together with Arrowhead Framework, has been used for real-time monitoring of storage tanks, predicting the consumption of the raw materials, and planning for future supplies.

Fig. 15 illustrates the software architecture of this case study. Specifically, the solution is made available through the *SupplyManagementApp* - a web-based application to manage the supply chain at Unger Fabrikker internally (e.g., create or modify the supply orders, visualize real-time values of the storage tanks and production) and the *SupplierApp* - a web-based application to provide the external suppliers with their corresponding supply data. For example, the SupplierApp provides the suppliers with detailed information about the level of the storage tanks that they supply and their corresponding future orders and deliveries. The suppliers can use

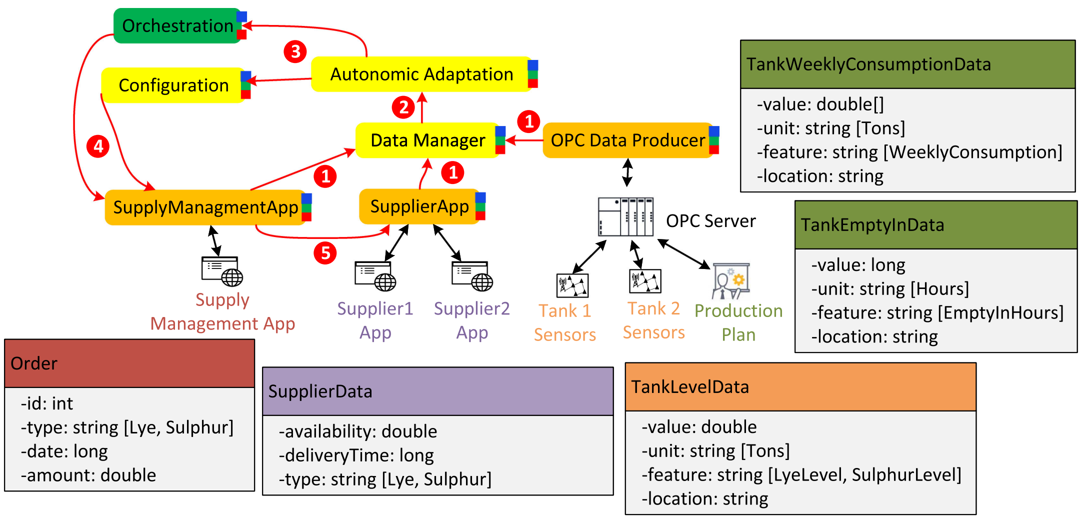


FIGURE 15. Information model and the communication of the application systems in the case study. The classes represent the information models of the components with the same color.

the information to continuously plan and assess the need for a new delivery in principle without involving Unger personnel.

To enable automatic monitoring of the supply chain, different sensors are installed to measure the level of storage tanks. These sensor values are stored in an OPC Server. Also, production information necessary for supply planning is made available at the OPC server. We develop an Arrowhead application system, *OPC Data Producer*, to enrich the raw data with more contextual information and make them accessible via services. These data are corresponding to *TankLevelData*, *TankWeeklyConsumptionData* and *TankEmptyInData* showed in Fig. 15. Specifically, the TankLevelData represents the level of the corresponding lye or sulfur tanks (in tons). The TankWeeklyConsumptionData provides the estimated usage of a storage tank for the following seven days. TankEmptyInData represents the number of hours until a particular tank is empty. These data will be pushed to the Data Manager (1) and subsequently transformed to semantic knowledge according to the Sensor model of the ontology shown in Fig. 7 (2). Furthermore, the information of the *Order* (e.g., date, amount, type of substance) and the *SupplierData* (e.g., available amount, average delivery time, type of substance) will also be captured by the Autonomic Adaptation System. Here we use the location value to identify the tank related to these data. In order to represent these data, the proposed ontology is extended with the Supplier model as shown in Fig. 16. In this case study, the SupplyManagementApp also consumes all the aforementioned data services and visualizes the data in a dashboard to support the management staff to monitor the supply chain.

Once the data is available, the Autonomic Adaptation System will automate the real-time monitoring of the availability of the supplies as well as the production plan and generate appropriate plans for new orders or deliveries of these supplies according to the high-level policies input from the

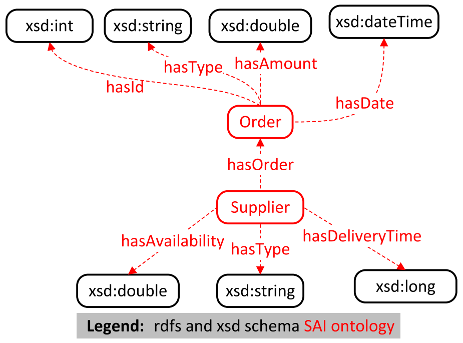


FIGURE 16. Extract of the SAI ontology representing the related concepts of the case study.

management staff at Unger Fabrikker. Table 2 shows the high-level policies that are used for monitoring and planning of the supply chain in this case study.These policies are described by SWRL according to the ontologies presented in Figs. 7 and 16. For example, Fig. 10 show the SWRL representation of policy #1 which instructs the SupplyManagmentApp to re-orchestrate to a new online tank sensor if the old one is offline. Policy #2 and #3 check if the value of tank level is below a threshold and send a notification to the SupplyManagmentApp accordingly. For notification purpose, we encapsulate the notification message into the *data* field of the configuration (example showed in Fig. 5) and push it to Configuration Store (3). The same mechanism is employed for creating or modifying new order. Therefore, the SupplyManagmentApp has to extract the value from the configuration (4) and decode it into respective message types (e.g., notification, create order and modify order). This method is implemented in the Adaptation Executor of the application as discussed in Section V-A. Policy #4 and #5 check the value of TankEmplyInData

TABLE 2. Policies for Monitoring and Planning of Supply Chain at Unger Fabrikker

Number	Description	Estimated time for manual handling
1	Re-orchestrate an Offline sensor service with an Online one located at the same tank.	5-30 minutes
2-3	Notify the management staff and corresponding suppliers whenever the level of the tank is low (one policy for each type of substance).	5-30 minutes
4-5	Create new order based on the TankEmptyInData value and notify the management staff and supplier accordingly (one policy for each type of substance).	several hours
6-7	Modify existing orders if the TankEmptyInData value changes and notify the management staff and supplier accordingly (one policy for each type of substance).	several hours
8-9	Notify the corresponding supplier whenever the deliver date is coming (one for each type of substance).	several hours

and instruct the SupplyManagmentApp to create a new order before the corresponding tank becomes empty. Similarly, Policy #6 and #7 monitor the changes of TankEmplyInData values and update the delivery date of the created orders accordingly. Finally, Policy #8 and #9 send a notification to the SupplyManagmentApp whenever the delivery date is coming.

Once the new orchestrations and configurations (e.g., notification, create and modify orders) are pushed to the SupplyManagementApp, this application will either update its dashboard or forward the notifications to the corresponding suppliers via the SupplierApp accordingly (5).

Table 2 also shows the estimated time for the management staff to execute those policies manually. Normally, it will take from a few minutes to hours to monitor the data, update or create the orders and communicate with the suppliers regarding those orders via emails or phone calls. In contrast, our solution facilitates the supply chain management at Unger Fabrikker by reducing internal monitoring and supervision regarding the planning and delivery of the supplies as well as the external communication with their suppliers. Once the policies are loaded into the Autonomic Adaptation System, the supply chain management will be executed automatically within a few seconds. The development and maintenance of policies may take long time and require domain knowledge, but will not occur frequently compared to continuous monitoring and communication with the suppliers.

In this case study, we introduced only a set of simple policies to illustrate the feasibility of our approach within a particular industrial domain. However, more conditions (e.g., arithmetic computations of the values) and relationship concepts can be added to the policies in order to describe more complex planning strategies. As long as these strategies can be modeled with the proposed ontologies, the Autonomic Adaptation System can make inferences with the SWRL and generate adaptations accordingly.

This case study was conducted as part of the collaboration during Productive4.0 project. Particularly, the Autonomic Adaptation System monitored the actual real-time production data provided by Unger Fabrikke. However, the generated adaptations were only provided as the recommendations to the management staff as well as to evaluate the feasibility and efficiency of the proposed approach.

VII. EVALUATION

In this section, we discuss the performance evaluation of the proposed approach based on the case study presented in the previous section. The following perspectives will be considered in our conducted experiments:

1) Reliability regarding the generation of adaptations properly and timely.
2) Efficiency in term of reduction in the manual effort related to monitoring and planning.
3) Scalability concerning the increase of services and systems.

The Arrowhead Framework version 4.0.0, Jena TDB triplestore and its API [5] are used as the core technologies for the implementation of the prototype. The Arrowhead core systems and application systems (depicted in Fig. 15) are deployed on a laptop with a 2.20 GHz Intel i7-8750H CPU and 16 GB RAM. In order to trigger the adaptations described in Table 2, the simulated events corresponding to the conditions of the policies are injected into the KB, and different metrics are measured accordingly.

A. EXPERIMENTAL RESULTS

Based on the perspectives mentioned above, the following measurements are recorded in order to evaluate the performance of our approach:

1) The maximum memory consumption of the Autonomic Adaptation System during its execution time.
2) Reasoning Time ($t_{reasoning}$): is the duration from the time when the simulated event is injected into the KB to the time when the respective adaptation is ready to send by the Execute component.
3) Adaptation Time ($t_{adaptation}$): is the duration from the time when the simulated event is injected into the KB to the time when the respective adaptation is executed by the corresponding application system.

Generally, $t_{adaptation} = t_{reasoning} + t_{communication}$, where $t_{communication}$ is the time for the adaptation to be sent from the Autonomic Adaptation System to the application system. The Reasoning Time and Adaptation Time are measured for each rule listed in Table 2. In order to ensure the reliability of the results, the experiments are performed ten times.

To evaluate the reliability of the approach to generate the expected adaptations correctly and timely, we modified the value of service monitoring data to meet the conditions of

[5][Online]. Available: https://jena.apache.org

TABLE 3. Average Reasoning Time, Communication Time and Adaptation Time (In Seconds)

	Policy 1	Policy 2	Policy 3	Policy 4	Policy 5	Policy 6	Policy 7	Policy 8	Policy 9
$t_{reasoning}$	0.745	0.741	0.759	1.995	1.951	2.428	2.273	0.778	0.795
$t_{adaptation}$	2.228	2.052	2.132	3.644	3.627	3.807	3.577	2.440	2.322

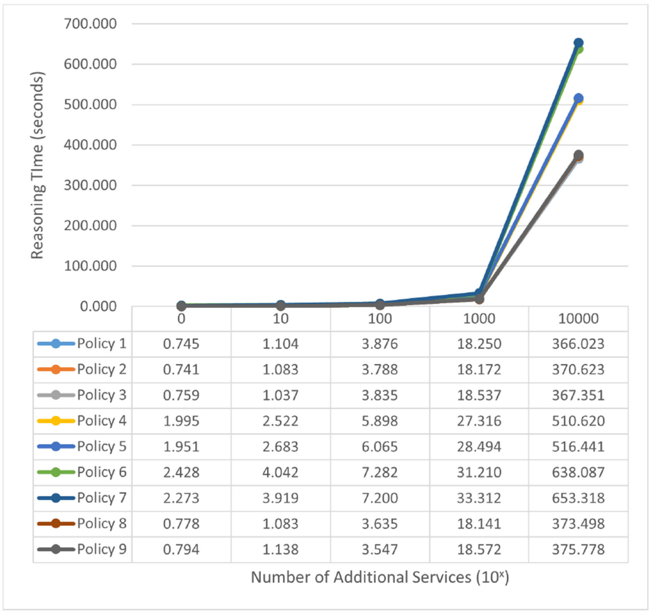
each policy in Table 2, captured and validated the respective adaptation generated by the Autonomic Adaptation System. The estimated time for manually handling the policies (as shown in Table 2) was used as the expected timeout for the Autonomic Adaptation System to reason over the KB and trigger an adaptation. Once the expected is generated, the Adaptation Time ($t_{adaptation}$) will also be measured for each scenario. Table 3 shows the average Reasoning Time of the nine policies. The Autonomic Adaptation System was able to generate all of the expected adaptations within 3 seconds after we injected the triggering events of policies. The Reasoning Time may vary according to the complexity of the policy. As can be seen from the table, policies #6 and #7 takes the longest time for reasoning as the Autonomic Adaptation System may need to compare the delivery dates of the orders with the date indicated in TankEmplyInData values in order to compute the new delivery dates, whereas simple policies such as #1, 2 or 3 may take a shorter time to reason as the Autonomic Adaptation System needs to compare only single data value. However, in general, within this prototype with only 2 Tanks and 2 Suppliers, the KB is relatively small (about 625 triples). Therefore, the measuring times do not differ significantly among the policies.

Furthermore, we also captured the time when the SupplyManagementApp finished executing the generated adaptations. As shown in Table 3, the Adaptation Time $t_{adaptation}$ represents the total adaptation process from simulated events were injected to adaptations has been completely carried out. In this prototype, because the SupplyManagementApp simply updates its internal data structure and prints out a message to the console whenever receiving the adaptation, executing the adaptation is also done within a few seconds. It is also worth mentioning that since we run all of the applications on the same machine, the network latency is also very small. However, in this approach, as adaptations are encapsulated into orchestration or configuration messages (as shown in Fig. 5), communicated among the systems within one local cloud, and can be handled automatically by the application systems, the overhead related to network latency or processing the adaptations is expected to be smaller than manual operations. As can be seen from Table 3, our approach shows orders of magnitude greater operational efficiency in term of executing the policies compared to the estimated manually handling time of the management staff (as shown in the third column of Table 2).

Finally, regarding the scalability issue, we repeated the same experiment as described above with a gradually increasing number of sensor services. We added arbitrary sensor services which are of a different type (e.g., having different FeatureOfInterest) from the ones used in this use case in order to avoid overlapping with the use case data as well as to simplify the process of validating the generated adaptations. Table 4 shows the number of triples of the KB with the increasing number of services as well as the maximum memory consumption of the Autonomic Adaptation System during its run-time. Specifically, the memory allocation tends to increase slower than the number of added services. This figure approximately doubles when the number of devices increases ten times and raises up to 1.5GB with 10000 services. As we apply in-memory data process and reasoning, this figure can be critical to the feasibly of our approach when deployed on resource-limited devices. Furthermore, we also measured the Reasoning Time of the Autonomic Adaptation System as shown in Fig. 17. Particularly, the system faces considerable challenges, which stem from the volume and variety of the KB. As can be seen from this figure, the numbers have the same increasing tendency that is exponential to the number of services. Most of the policies were triggered within 8 seconds when there were 100 more services. The number increased

TABLE 4. Number of Triples and Maximum Memory Consumption of the Autonomic Adaptation System With Increasing Number of Services

Number of additional services	0	10	100	1000	10000
Number of Triples	625	775	2275	17275	32275
Maximum RAM Consumption(MB)	238	295	373	944	1507



	0	10	100	1000	10000
Policy 1	0.745	1.104	3.876	18.250	366.023
Policy 2	0.741	1.083	3.788	18.172	370.623
Policy 3	0.759	1.037	3.835	18.537	367.351
Policy 4	1.995	2.522	5.898	27.316	510.620
Policy 5	1.951	2.683	6.065	28.494	516.441
Policy 6	2.428	4.042	7.282	31.210	638.087
Policy 7	2.273	3.919	7.200	33.312	653.318
Policy 8	0.778	1.083	3.635	18.141	373.498
Policy 9	0.794	1.138	3.547	18.572	375.778

FIGURE 17. Average Reasoning Time of the Autonomic Adaptation System with increasing number of services.

to 34 seconds with 1000 services more and went up to 11 minutes (660 seconds) when 10000 services were added.

B. DISCUSSION

In this use case, we have demonstrated the feasibility of the Autonomic Adaptation System to automate the processes of monitoring the production status and making supplying orders. These processes are described with orchestration and configuration policies that can be updated and governed by the Autonomic Adaptation System during run-time. The experiment results present the *reliability* of the approach by generating the expected outputs properly and timely. The Autonomic Adaptation System was able to reason for the policies and generate corresponding adaptations within a few seconds, while these processes may take hours to handle manually. Although there are only nine policies introduced, these policies are sufficient to cover most representative scenarios from the supply chain at Unger Fabrikker. Moreover, additional policies can also be added to the system without significant effect on the performance of the system because these policies can be handled individually and parallel by the Autonomic Adaptation System. As our approach models the adaptation policies as orchestration and configuration policies, it is developed to support real-time monitoring and autonomous decision-making such as anomaly detection or run-time optimization and configuration of the production systems based on the collected data. For more complex orchestration scenarios that involve other entities in the manufacturing (e.g., machines, robot, products), more support systems in the Arrowhead Framework can be added to our proposed approach to accomplish more advanced features. For example, The Workflow Choreographer [47] that exploits BPMN (Business Process Model and Notation) and CPN (Colored Petri Net) languages to model the production "recipe" and its sequences of tasks, which will be generated into executable models. These models are used to orchestrate the tasks, services, resources, machines, etc. in accordance with the specified workflow of the production. Our proposed Autonomic Adaptation System, instead, can be exploited to enhance this Workflow Choreographer solution by providing run-time critical events of the systems to achieve more flexible production. However, as showed in the evaluation result, the Reasoning Time of our approach may be longer for more complex policies and larger KB. The reason is that the Autonomic Adaptation System has to match multiple conditions of the policies over the whole KB. Nevertheless, experiments showed that the approach is still able to provide expected results within the timeout whenever the KB grows larger with 10000 additional sensor services. Furthermore, as shown in Fig. 10, the policy consists of different conditions (represented by the triples) which can be evaluated parallel. Therefore, one complex policy may be split into smaller sub-policies and handled simultaneously in order to reduce the reasoning time and diagnostic effort as well as to avoid error or editing mistakes.

Regarding the *efficiency* aspect compared with the current manual process, the experiments also present a significant improvement in the total operation time of each scenario. Accordingly, our autonomic approach can execute processes of monitoring and generating supply orders much faster than the estimated manual time of the operator. As everything is handled automatically once policies are available in the system, our approach can provide expected output within a few seconds in comparison to minutes or hours of manual monitoring and communication among the partners. However, there is still one recognized issue related to the creation and validation of the policies. In this prototype, we assume that the policies are available beforehand for the experiments. In fact, these policies manifest the knowledge of the domain experts and may take more time and experience to infer. Nevertheless, the Autonomic Adaptation System can be used as the database that persists the knowledge which can be used as the base resource for making further optimization, diagnostic or training purposes. Furthermore, although making new policies can take a long time, this does not happen very frequently. Instead, executing those policies may require more effort throughout the production of the factory. As a result, the proposed approach can demonstrate its efficiency in term of human effort reduction, especially for use cases involving a large number of field devices and applications. Furthermore, the process of validating the generated policies and their corresponding adaptations is performed manually in the presented use case. With a small number of policies, this may not take too much time. Also, compared to the great effort related to manual operation, it can be much more efficient to just evaluate the adaptations proposed by the system. For future direction, machine learning or data-driven methods can be applied as an external module to the MAPE-K component to support such processes of creating and validating the policies and adaptations.

Furthermore, the *scalability* evaluation also shows the potential of our approach for use cases that involve a large number of physical devices. Accordingly, by exploiting the current SAI ontologies, there are not many additional triples for each added physical device. As a result, our approach does not require too much RAM memory for reasoning tasks with the increasing number of devices. However, the reasoning task may suffer from scalability for larger KB, which could introduce challenge to the deployment of Autonomic Adaptation System to resource-limited devices. Nevertheless, benefit from the modular characteristic of the proposed decentralized SoA, different components of the system can be deployed on multiple resource-constraint devices. For further improvement, solutions for lightweight triplestore such as the approach in [48] can be considered for the implementation of the SAI platform. Furthermore, to improve the reasoning performance, mechanisms such as parallelizing the evaluation of policies or splitting policies can be applied to complement the reasoning performance. Moreover, as our approach applied the decentralized SoA where each local cloud has its own Autonomic Adaptation System to support dynamic adaptation, the total number of devices in each local cloud is expected not to be substantial. Therefore, as shown in the last experiment, the Autonomic Adaptation System can also handle a practical number of monitoring services with reasonable response time. Furthermore, as each use case may have its separate set of related physical devices or services, the approach can also be designed to have different KBs for different use cases, thereby addressing the scalability issue while facilitating analysis or diagnostic purposes.

VIII. CONCLUSION

Industry 4.0 envisions a new generation of automation systems where modern Information and Communication Technologies such as IIoT, CPS, and Cloud-based Technologies are applied to facilitate the vertical integration of field devices from the shop floor to enterprise software systems to establish an agile manufacturing environment. However, applying these technologies also brings new challenges such as security, heterogeneity, interoperability, and scalability. To address these challenges, this work proposes a decentralized service-oriented architecture for self-adaptive IIoT systems. Specifically, our approach provides self-adaptation as a service within one local automation cloud where software systems and devices are orchestrated and configured securely and automatically. Self-adaptation is achieved by employing the MAPE-K model proposed for Autonomic Computing and implemented with different tools from Semantic Web Technologies. Furthermore, this approach enables the integration of domain expert knowledge in the form of high-level policies in order to support the generation of adaptations. This is a generic approach that can be applied to different application domains. The approach is implemented as the Autonomic Adaptation System, a support core system within Arrowhead Framework, to support dynamic orchestration and configuration for the application systems during run-time. As a proof-of-concept, this solution was used to demonstrate dynamic planning for the supply chain of the production at the chemical factory Unger Fabrikker.

Our solution of dynamic adaptation can be applied to systems at different levels. For example, it can be used to deal with uncertainties and anomalies during the production at the manufacturing control level, or to facilitate the communication and planning for the supply chain at the manufacturing management level as shown in the proposed case study. Therefore, employing dynamic adaptation can reduce the engineering and management effort related to manual maintenance, diagnostic and configuration with regards to changes and evolution of the production systems as well as improve the efficiency and sustainability of production in term of time, waste, or other similar objectives. This approach does not eliminate humans from the operation management. Instead, it assists management staff with situation evaluation and planning so that they can focus on supervision and establishing self-management policies.

However, there is one issue that needs to be addressed in our future research. As we use SWRL to describe the high-level adaptation policies, the approach may be error-prone and difficult for domain experts to specify and verify these policies. To deal with this issue, an approach to facilitate the creation of policies is needed. One solution is to develop a tool to assist the user in creating, editing, and validating the adaptation policies. Furthermore, to achieve the higher level of adaptation as shown in Fig. 2, data-driven algorithms could be applied to enhance the MAPE-K with automatic generation of policies. The integration of ML and reinforced learning into MAPE-K model as suggested in [49] should be taken into consideration for this direction.

REFERENCES

[1] U. D. Atmojo, Z. Salcic, I. Kevin, K. Wang, and V. Vyatkin, “A service-oriented programming approach for dynamic distributed manufacturing systems,” *IEEE Trans. Ind. Inf.*, vol. 16, no. 1, pp. 151–160, Jan. 2019.

[2] O. Givehchi, K. Landsdorf, P. Simoens, and A. W. Colombo, “Interoperability for industrial cyber-physical systems: An approach for legacy systems,” *IEEE Trans. Ind. Informat.*, vol. 13, no. 6, pp. 3370–3378, Dec. 2017.

[3] H. Derhamy, J. Eliasson, and J. Delsing, “IoT interoperability—on-demand and low latency transparent multiprotocol translator,” *IEEE Internet Things J.*, vol. 4, no. 5, pp. 1754–1763, Oct. 2017.

[4] B. Scholz-Reiter and M. Freitag, “Autonomous processes in assembly systems,” *Confinement Inf. Retrieval Process. Ann.*, vol. 56, no. 2, pp. 712–729, 2007.

[5] J. Váncza *et al.*, “Cooperative and responsive manufacturing enterprises,” *Confinement Inf. Retrieval Process. Ann.*, vol. 60, no. 2, pp. 797–820, 2011.

[6] M. E. Leusin, E. M. Frazzon, M. Uriona Maldonado, M. Kück, and M. Freitag, “Solving the job-shop scheduling problem in the industry 4.0 era,” *Technologies*, vol. 6, no. 4, 2018, Art. no. 107.

[7] B. Katti, “Ontology-based approach to decentralized production control in the context of cloud manufacturing execution systems,” Ph.D. thesis, Technische Universitát Kaiserslautern, 2020. [Online]. Available: http://nbn-resolving.de/urn:nbn:de:hbz:386-kluedo-59873

[8] J. Delsing, *IoT Automation: Arrowhead Framework*. Boca Raton, FL, USA: CRC Press, 2017.

[9] M. De Sanctis, H. Muccini, and K. Vaidhyanathan, “Data-driven adaptation in microservice-based iot architectures,” in *Proc. IEEE Int. Conf. Softw. Architecture Companion*. 2020, pp. 59–62.

[10] Q. Cao, F. Giustozzi, C. Zanni-Merk, F. de Bertrand de Beuvron, and C. Reich, “Smart condition monitoring for industry 4.0 manufacturing processes: An ontology-based approach,” *Cybern. Syst.*, vol. 50, no. 2, pp. 82–96, 2019.

[11] P. Horn, “Autonomic computing: IBM’s perspective on the state of information technology,” IBM, Tech. Rep. 2001. [Online]. Available: http://www.ibm.com/research/autonomic

[12] D. Weyns, M. U. Iftikhar, D. Hughes, and N. Matthys, “Applying architecture-based adaptation to automate the management of internet-of-things,” in *Proc. Eur. Conf. Softw. Architecture*., 2018, pp. 49–67.

[13] C. Krupitzer, M. Breitbach, J. Saal, C. Becker, M. Segata, and R. L. Cigno, “RoCoSys: A framework for coordination of mobile IoT devices,” in *Proc. IEEE Int. Conf. Pervasive Comput. Commun.*, 2017, pp. 485–490.

[14] C. Cetina, P. Giner, J. Fons, and V. Pelechano, “Autonomic computing through reuse of variability models at runtime: The case of smart homes,” *Computer*, vol. 42, no. 10, pp. 37–43, 2009.

[15] E. Mezghani, E. Exposito, and K. Drira, “A model-driven methodology for the design of autonomic and cognitive IoT-based systems: Application to healthcare,” *IEEE Trans. Emerg. Top. Comput. Intell.*, vol. 1, no. 3, pp. 224–234, Jun. 2017.

[16] F. Ciccozzi and R. Spalazzese, “MDE4IoT: Supporting the Internet of Things with model-driven engineering,” in *Proc. Int. Symp. Intell. Distrib. Comput.*, 2016, pp. 67–76.

[17] F. J. Acosta Padilla, “Self-adaptation for internet of things applications,” Ph.D. dissertation, *Université Rennes* 1, 2016. [Online]. Available: https://tel.archives-ouvertes.fr/tel-01426219

[18] L. Florio and E. Di Nitto, “Gru: An approach to introduce decentralized autonomic behavior in microservices architectures,” in *Proc. IEEE Int. Conf. Autonomic Comput.*, 2016, pp. 357–362.

[19] K. Baylov and A. Dimov, “Reference architecture for self-adaptive microservice systems,” in *Proc. Int. Symp. Intell. Distrib. Comput.*, 2017, pp. 297–303.

[20] H. Khazaei, A. Ghanbari, and M. Litoiu, “Adaptation as a service,” in *Proc. Int. Conf. Softw. Syst.*, 2018, pp. 282–288.
[21] S. Maksuti *et al.*, “Establishing a chain of trust in a sporadically connected cyber-physical system,” in *Proc. IFIP/IEEE Int. Symp. Integr. Netw. Manage.* 2021, pp. 890–895.
[22] B. Magableh and M. Almiani, “A self healing microservices architecture: A case study in docker swarm cluster,” in *Proc. Int. Conf. Adv. Inf. Netw. Appl.*, 2019, pp. 846–858.
[23] M. Faraji Mehmandar, S. Jabbehdari, and H. Haj Seyyed Javadi, “A dynamic fog service provisioning approach for IoT applications,” *Int. J. Commun. Syst.*, vol. 33, no. 14, 2020, Art. no. e4541.
[24] R. Dautov, “Exclaim framework: A monitoring and analysis framework to support self-governance in cloud application platforms,” Ph.D. dissertation, Univ. Sheffield, 2015.
[25] M. Parashar and S. Hariri, “Autonomic computing: An overview,” in *Proc. Int. Workshop Unconventional Program. Paradigms*. 2004, pp. 257–269.
[26] A. Computing *et al.*, “An architectural blueprint for autonomic computing,” *IBM White Paper*, vol. 31, no. 2006, pp. 1–6, 2006.
[27] M. C. Huebscher and J. A. McCann, “A survey of autonomic computing—degrees, models, and applications,” *ACM Comput. Surv.*, vol. 40, no. 3, pp. 1–28, 2008.
[28] A. Murguzur, R. Capilla, S. Trujillo, Ó. Ortiz, and R. E. Lopez-Herrejon, “Context variability modeling for runtime configuration of service-based dynamic software product lines,” in *Proc. 18th Int. Softw. Product Line Conf.: Companion Volume Workshops*, Demonstrations and Tools-Volume 2, 2014, pp. 2–9.
[29] J. C. Georgas and R. N. Taylor, “Policy-based self-adaptive architectures: A feasibility study in the robotics domain,” in *Proc. Int. Workshop Softw. Eng. Adaptive Self- Manag. Syst.*, 2008, pp. 105–112.
[30] A. Rafique, D. Van Landuyt, E. Truyen, V. Reniers, and W. Joosen, “SCOPE: Self-adaptive and policy-based data management middleware for federated clouds,” *J. Internet Serv. Appl.*, vol. 10, no. 1, pp. 1–19, 2019.
[31] K. S. Phanse, L. A. DaSilva, and S. F. Midkiff, “Design and demonstration of policy-based management in a multi-hop ad hoc network,” *Ad. Hoc Netw.*, vol. 3, no. 3, pp. 389–401, 2005.
[32] R. Basra, K. Lü, G. Rzevski, and P. Skobelev, “Resolving scheduling issues of the london underground using a multi-agent system,” in *Proc. Int. Conf. Ind. Appl. Holonic Multi-Agent Syst.*, 2005, pp. 188–196.
[33] D. Garlan, S.-W. Cheng, A.-C. Huang, B. Schmerl, and P. Steenkiste, “Rainbow: Architecture-based self-adaptation with reusable infrastructure,” *Computer*, vol. 37, no. 10, pp. 46–54, 2004.
[34] F. Quin, D. Weyns, T. Bamelis, S. S. Buttar, and S. Michiels, “Efficient analysis of large adaptation spaces in self-adaptive systems using machine learning,” in *Proc. IEEE/ACM 14th Int. Symp. Softw. Eng. Adaptive Self- Manag. Syst.*, 2019, pp. 1–12.
[35] V. Klös, T. Göthel, and S. Glesner, “Adaptive knowledge bases in self-adaptive system design,” in *Proc. 41st Euromicro Conf. Softw. Eng. Adv. Appl.*, 2015, pp. 472–478.
[36] P. Jamshidi, A. Sharifloo, C. Pahl, H. Arabnejad, A. Metzger, and G. Estrada, “Fuzzy self-learning controllers for elasticity management in dynamic cloud architectures,” in *Proc. 12th Int. ACM SIGSOFT Conf. Qual. Softw. Architectures*, 2016, pp. 70–79.
[37] I. Marcu, G. Suciu, C. Bălăceanu, A. Vulpe, and A.-M. Drăgulinescu, “Arrowhead technology for digitalization and automation solution: Smart cities and smart agriculture,” *Sensors*, vol. 20, no. 5, 2020, Art. no. 1464.
[38] C. Hegedus, P. Varga, and A. Frankó, “Secure and Trusted Inter-Cloud Communications in the Arrowhead Framework,” in *Proc. IEEE Ind. Cyber-Physical Syst.*, 2018, pp. 755–760.
[39] A. N. Lam and Ø. Haugen, “Supporting IoT semantic interoperability with autonomic computing,” in *Proc. IEEE Ind. Cyber-Physical Syst.*, 2018, pp. 761–767.
[40] A. N. Lam, “Sai: A service oriented autonomic IoT platform,” Master’s thesis, 2017.
[41] N. Seydoux, K. Drira, N. Hernandez, and T. Monteil, “IoT-o, a core-domain iot ontology to represent connected devices networks,” in *Proc. Eur. Knowl. Acquisition Workshop*, 2016, pp. 561–576.
[42] K. Janowicz, A. Haller, S. J. Cox, D. Le Phuoc, and M. Lefrançois, “SOSA: A lightweight ontology for sensors, observations, samples, and actuators,” *J. Web Semantics*, vol. 56, pp. 1–10, 2019.
[43] A. Haller, K. Janowicz, S. Cox, D. Le Phuoc, K. Taylor, and M. Lefrançois, “Semantic sensor network ontology,” 2017. [Online]. Available: https://www.w3.org/TR/vocab-ssn/
[44] M. Hoffmann, *Smart Agents for the Industry 4.0: Enabling Machine Learning in Industrial Production*. Springer Vieweg, Wiesbaden, 2019.
[45] B. R. Ferrer and J. L. M. Lastra, “Private local automation clouds built by cps: Potential and challenges for distributed reasoning,” *Adv. Eng. Informat.*, vol. 32, pp. 113–125, 2017.
[46] S. Plósz, C. Hegedűs, and P. Varga, “Advanced security considerations in the arrowhead framework,” in *Proc. Int. Conf. Comput. Safety, Reliability, Secur.*, 2016, pp. 234–245.
[47] D. Kozma, P. Varga, and F. Larrinaga, “Dynamic multilevel workflow management concept for industrial IoT systems,” *IEEE Trans. Automat. Sci. Eng.*, vol. 18, no. 3, pp. 1354–1366, Jul. 2021.
[48] R. Verborgh *et al.*, “Low-cost queryable linked data through triple pattern fragments.” in *Proc. Int. Semantic Web Conf.*, 2014, pp. 13–16.
[49] A. Palm, A. Metzger, and K. Pohl, “Online reinforcement learning for self-adaptive information systems,” in *Proc. Int. Conf. Adv. Inf. Syst. Eng.*, 2020, pp. 169–184.

AN NGOC LAM (Student Member, IEEE) received the B.Sc. degree in computer engineering from the Ho Chi Minh City University of Technology, Ho Chi Minh, Vietnam, in 2012, and the M.Sc degree in computer science from Østfold University College, Halden, Norway, in 2017. Since 2017, he has been working toward the Ph.D. degree with the Luelå University of Technology, Luelå, Sweden. He was a Researcher with Cyber-physical Systems Group, Østfold University College. He has been a part of the PRODUCTIVE4.0 and ArrowheadTools EU projects.

ØYSTEIN HAUGEN (Member, IEEE) is currently a Professor of computer science with Østfold University College, Halden, Norway, and leads the research group on Cyber-physical systems. He has participated in the evolution of the Arrowhead framework for several years. His main research interests include the development and usage of modeling languages especially in the field of real-time systems. Since 1990, he has been involved with the standardization of modeling languages. Before becoming a Professor, he was with large industry companies like Ericsson and ABB, and research institutes SINTEF and Norwegian Computing Center, Oslo, Norway.

JERKER DELSING (Member, IEEE) received the M.Sc. degree in engineering physics from the Lund Institute of Technology, Lund, Sweden, in 1982, and the Ph.D. degree in electrical measurement from Lund University, Lund, Sweden, in 1988. From 1985 to 1988, he worked part time with Alfa-Lava - SattControl (now, ABB) in development of sensors and measurement technology. In 1994, he was promoted to an Associate Professor in heat and power engineering with Lund University. In 1995, he was appointed as a Full Professor in industrial electronics with Luleå University of Technology, Luelå, Sweden, where he is currently the Scientific Head of EISLAB. He and his EISLAB Group is partner of several large EU projects in the field, such as Socrades, IMC-AESOP, Arrowhead, FAR-EDGE, Productive4.0, and Arrowhead Tools. His research interests include IoT and SoS automation, with applications to automation in large and complex industry, and society systems. He is a Board Member of the ARTEMIS, ProcessIT.EU, and ProcessIT Innovations.