Generic Autonomic Management as a Service in a SOA-based Framework for Industry 4.0

Silia Maksuti[1,2], Markus Tauber[1] and Jerker Delsing[2]

[1]*University of Applied Sciences Burgenland* - Eisenstadt, Austria
[2]*Luleå University of Technology* - Luleå, Sweden

***Abstract*—Cyber-physical production systems are engineered systems that are built from, and depend upon, the seamless integration of computational algorithms and physical components. In order to make these system interoperable with each other for addressing Industry 4.0 applications a number of service-oriented architecture frameworks are developed. Such frameworks are composed by a number of services, which are inherently dynamic by nature and thus imply the need for self-adaptation. In this paper we propose generic autonomic management as a service and show how to integrate it in the Arrowhead framework. We propose generic and reusable interfaces for each phase of the autonomic control loop in order to increase the usability of the service for other frameworks and application systems, while reducing the software engineering effort. To show the utility of our approach in the Arrowhead framework we use a climate control application as a representative example.**

I. Introduction

Cyber-Physical Production Systems (CPPS), Internet of Things (IoT), cloud computing are the drivers of Industry 4.0 and enable the creation of high-level services. E.g. a device, even a single sensor, could produce a service. Services open new business opportunities by increasing interoperability, simplifying application development, abstracting the complexity of underlying system, improving the end user experience, etc.

The Service-Oriented Architecture (SOA) is an approach applied to distributed systems that employs loosely coupled services, standard interfaces and protocols to deliver seamless cross platform integration. In a SOA each component in the system provides a service, which has a well defined set of functionalities that does not depend on the state of other services, known as service autonomy. Additionally, only the service provider needs to know the logic of its service, that is why the logic is abstracted through encapsulation, known as service abstraction. These two features, service autonomy and service abstraction, are relevant in the context of this paper as they allow to treat services as autonomic components and allow their self-adaptation. Each service has also a set of communicative meta-data, by which it can be discovered by service consumers. Services comply to communication agreements as defined in standardized service contracts and because they are independent from any particular process they can be reused, composed and decomposed to new configurations. Thus, SOA is based on three main fundamentals: (i) loosely coupled, which allows for dynamic provisioning or deprovisioning of resources to maintain an effective load balancing mechanism, (ii) late binding, which makes possible to use the service any time by connecting to the correct resources and (iii) lookup, which can be used to discover already registered services.

Self-adaptation can be used to deal with rapid changes in the environment and in the system itself, by allowing the system to gather knowledge at runtime and to apply the adaptation policies in specific services. As mentioned above, service autonomy and service abstraction features of SOA allow to treat services as autonomic components. By adapting to system changes with minimal intervention, the advantages of self-adaptation (e.g. increased industrial uptime, more efficient systems, ability to predict errors, etc.) will bring a new impact on interoperable digital manufacturing platforms.

In this paper, we propose generic autonomic management as a service. We build on our previous work on Generic Autonomic Management Framework [1], which is a framework that can be used to develop autonomic managers for any target system without having the (re)implement the generic control mechanisms, and extend it to support also SOA-based frameworks. Given its generic property, the service can be used by different application systems by changing the metrics and adaptation policies, while reducing the software engineering effort needed to implement the generic control mechanisms.

We show how to integrate generic autonomic management service in the Arrowhead framework [2], since it is an ideal support of the approach given its SOA nature. The Arrowhead framework aims to facilitate the creation of local automation clouds, which include devices, application systems and services used to perform the automation tasks. This enables local (on-site) real time performance and security, interoperability, simple and cheap engineering, and scalability through multi-cloud interactions. We consider climate control application as a representative example to explain the utility of the proposed service in the Arrowhead framework.

The remainder of this paper is structured as follows. Section II presents a state of the art analysis of self-adaptation approaches and adaptive SOA-based frameworks. Section III introduces the Arrowhead framework and its core systems and services. Section IV briefly introduces the GAMF framework architecture. In section V we propose generic autonomic management service and show how to integrate it in the Arrowhead framework. We further explain its utility through a representative example, a climate control application. Section VI outlines the findings and the future work.

II. Related Work

In this section we present the state-of-the-art related to self-adaptation approaches and existing solutions on how is self-adaptation applied in SOA-based frameworks.

A. Self-Adaptation Approaches

The complex CPPS are often exposed to various types of uncertainties, e.g., changing environment conditions, unpredictable user behavior, dynamically changing requirements and goals of the system itself, etc. Since predicting such uncertainties in the design phase of CPPS is difficult, they can affect the system qualities, e.g., security, reliability, flexibility, etc. Self-adaptation approach can be used to deal with uncertainties in CPPS by allowing the system to gather knowledge at the runtime. The engineering aspects of self-adaptive systems have emerged over time.

1) Autonomic Computing (AC): The vision of Autonomic Computing [3] has been first introduced by IBM to overcome the issue of anticipating and designing interactions among components in interconnected systems, leaving such issues to be dealt with at runtime. The term autonomic is derived from human biology, in which the autonomic nervous system monitors the heartbeat, checks the blood sugar level and keeps the body temperature constant without any conscious effort on your part. However the main difference is that the decisions made by autonomic capabilities in our body are involuntary, whilst the autonomic capabilities in computer systems depend on adaptation policies. The MAPE-K feedback loop is a closed software component with interfaces to the physical world via sensors and effectors, which consists of four phases (Monitor, Analyze, Plan, Execute) and shared Knowledge.

2) Architecture-driven adaptation: The architecture-driven adaptation mainly addresses the separation between change management and goal management based on a 3-layer model. The model consists of: (i) component control layer, which makes easier to report the current status of components and performs adaptations, (ii) change management layer, which reacts to changes in state of the lower level by executing actions to handle the new situation, (iii) goal management layer, which handles requests from the layer below and generates plans based on high-level goals. Rainbow [4] is a architecture-based self-adaptation framework with reusable infrastructure. Rainbow uses an abstract model to monitor an executing system's runtime properties, evaluates the model for constraint violation, and if a problem occurs performs adaptations on the running system. StarMX [5] is also a architecture-driven adaptation framework for developing self-managing software systems. It is a generic configurable framework based on standards and well-established principles, and provides the required features and facilities for the development of such systems. FORMS [6] is a formal reference model, which provides a formally founded vocabulary for the key architectural constructs comprising self-adaptive systems instead of an implementation framework from which self-adaptive applications can be derived. It is based on three primary aspects, reflectivity, MAPE-K and distributed coordination.

3) Model-driven adaptation: The model-driven adaptation is based on runtime models as key elements to engineer self-adaptive systems. A runtime model is a casually connected self-representation of the system, which enables managing the complexity of large amounts of information associated with runtime aspects. A state of the art framework that supports dynamic adaptation using model-driven approach is Models@Run.Time [7]. FUSION [8] is a framework for engineering self-adaptive systems, which uses a feature-oriented system model to learn the impact of feature selection and feature interactions on the system's competing goals. It then uses this knowledge to efficiently adapt the system to satisfy as many user-defined goals as possible.

4) Goal-driven adaptation: The goal-driven adaptation is mainly concerned with enabling languages and formalisms to specify requirements for feedback loops for self-adaptive systems, e.g., RELAX [9] language, and if feedback loops constitute a solution for adaptation, identifying the requirements this solution intends to solve. The requirements that should be addressed by feedback loops include e.g., runtime success/failure/quality of other requirements.

5) Guarantees under uncertainty: An important concern when dealing with self-adaptive systems is how to guarantee adaptation goals under uncertainty by using e.g., formal techniques at runtime such as ActivFORMS [10]. The formal model is directly executed by a virtual machine to realize adaptation. The approach assures that the adaptation goals that are verified off-line are guaranteed at runtime, and it supports dynamic adaptation of the active model.

6) Control-based adaptation: The control-based adaptation is concerned with applying principles from the control theory to realize self-adaptation, since control theory offers a mathematical foundation to design and analyse self-adaptive systems. E.g. DYNAMICO [11] is a reference model that improves the engineering of self-adaptive systems by addressing the management of adaptation properties as control objectives, the separation of concerns among feedback loops required to address control objectives over time, and the management of dynamic context as an independent control function to preserve context-awareness in the adaptation mechanism.

Self-adaptation is also related with other contributing disciplines, such as: (i) control theory, well-known principles of the control theory, e.g. the optimal closed-loop control problem (feedback control), can be applied to implement adaptive managers, (ii) decision theory and utility, and (iii) artificial intelligence, e.g. one issue in MAPE-K adaptation loop is that static policies cannot deal with emergent situations that have been unknown to engineers at the design time. This can be improved by using deep learning algorithms, such as model-based reinforcement learning (RL) [12]. An RL agent attempts to learn the model of its environment simultaneously and predicts the consequences of actions before they are taken. Thus, it can improve the MAPE-k adaptation loop by bringing policy evolution closer to real-world applications.

B. Adaptive SOA-based Frameworks

Engineering complex and dynamic CPPS requires alternative paradigms in system architectures, which should focus on: (i) services over components, (ii) interoperability and cross platform to deal with the level of diversity between components, (iii) loose coupling to deal with the autonomy of components and (iv) high level of abstraction to deal with the complexity of such systems. SOA paradigm has evolved as a solution to address these issues. The Service-Oriented Solution Stack (S3) [13] provides a detailed architectural definition of a SOA across nine layers, five horizontal and four vertical layers, that aim to reinforce business value. The horizontal layers (operational, service components, services, business process and consumers) are associated with a SOA application structure, whereas the vertical layers (integration, QoS, information architecture and governance) are responsible for cross-application aspects such as integration, security. Even though the S3 model is widely used as a reference model, several extensions to the original model have been done.

MetaSelf [14] is a framework for engineering dependable self-adaptive systems based on a set of requirements, which supports design-time and run-time aspects. The run-time environment is based on a service-oriented architecture, which exploits metadata (data about the running system, e.g. its components, infrastructure and environment) to support decision-making and adaptation based on the dynamic enforcement of explicitly expressed policies. Metadata and policies are managed by appropriate services. SASSY [15] is a model-driven framework for self-architecting service-oriented systems. Throughout the system's life cycle, SASSY aims to maintain an optimal architecture for satisfying functional and quality-of-service (QoS) requirements. MOSES [16], similar to SASSY, is a framework that supports QoS-driven runtime adaptation of service-oriented systems. MUSIC [17] is a framework and methodology for self-adapting applications in ubiquitous computing environments, which supports several adaptation mechanisms and offers a model-driven application development approach supported by a sophisticated middleware. It facilitates the dynamic and automatic adaptation of applications and services based on a clear separation of business logic, context awareness and adaptation concerns.

In this paper, we build on our previous work on self-adaptation [1], [18], [19], [20] and enhance it to also support SOA-based frameworks. Therefore, we propose to develop generic autonomic management as a service, aimed to support the application developers in building self-adaptive applications. In comparison with the aforementioned work the proposed solution is intended to be generic, so that it can be used in different SOA-based frameworks by a number of application systems without requiring a high adjustment effort.

III. Arrowhead Framework

The objective of the Arrowhead framework architecture is to facilitate the creation of local automation clouds and enable local real time performance and security, interoperability, simple and cheap engineering and scalability through multi cloud interaction [2]. The architecture is build based on the SOA fundamentals and addresses the move from large monolithic organisations towards multi-stakeholder co-operations, thus addressing the high level requirements in today's society such as sustainability, flexibility, efficiency and competitiveness. In terms of the Arrowhead framework, a service is an information exchange from a service producer system to a service consumer system. A system provides and/or consumes multiple services and a hardware device is a piece of equipment, machine or hardware with computational, memory and communication capabilities, which can host one or several systems. The Arrowhead framework architecture is composed of a number of systems, including mandatory and automation support core systems and application systems.

A. Arrowhead Local Cloud and Core Systems

The native environment of Arrowhead is the industrial automation domain. A factory is an application example, where a limited number of interconnected sensors, controllers and actuators work together on effectively assembling products. This motivates the local cloud approach. As shown in Figure 1, an Arrowhead local cloud is composed of the three mandatory core systems and at least one application system.

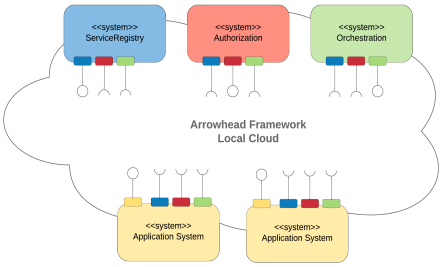

Fig. 1. Arrowhead Local Cloud. The Arrowhead local cloud is composed of three mandatory core systems: (i) ServiceRegistry, (ii) Authorization, (iii) Orchestration and at least one application system

The mandatory core systems include ServiceRegistry, Authorisation and Orchestration systems. The ServiceRegistry system is used to provide storage of all active services registered within a local cloud and enables the discovery of them. The Authorisation system is used to provide authentication, authorisation and optionally accounting of service interactions. The Orchestration system is used to provide a mechanism for distributing orchestration rules and service consumption patterns, thus providing service endpoints to specific requests.

The automation support core systems such as, PlantDescription, EventHandler, Workflow Manager etc., are used to facilitate automation application design, engineering and operation. They should be able to support the implementation of plant automation, housekeeping within the local cloud, inter-cloud service exchange, system and service interoperability, etc. In our previous work [21] we have developed SystemRegistry and DeviceRegistry systems, which are needed for the on-boarding procedure to create a chain of trust within the Arrwohead framework. SystemRegistry is used to provide local cloud

storage for systems registered within Arrwohead, metadata of these systems and their running services. DeviceRegistry is used to provide local cloud storage holding the information about devices registered in the Arrowhead, meta data of these devices and the list of systems deployed on them. These systems are important to create a chain of trust from a hardware device to a hosted software system and its associated services during the on-boarding procedure. In this paper we propose a new support core system for the Arrowhead framework, which will produce a generic autonomic management service, used to provide self-adaptation support within the local cloud.

B. Application Systems

The application systems are used to implement the consumption/production of services aiming to fulfill application requirements. They should be consuming at least the three mandatory core services of the Arrowhead local cloud, thus ServiceDiscovery produced by ServiceRegistry system, AuthorisationControl produced by Authorisation system and OrchestrationStore produced by Orchestration system, in order to be Arrowhead compliant. The Arrowhead framework defines three maturity levels for application systems: *level-3*, the application system implements the consumption/production of services without external components, *level-2*, the application system implements the consumption/production of services by using a software adapter, and *level-1*, the application system uses dedicated hardware with software responsible for wrapping the application system with Arrowhead framework compliant services.

IV. Generic Autonomic Management Framework

The Generic Autonomic Management Framework (GAMF), shown in Figure 2, is a Java-based framework used to develop autonomic managers for any target system without having to (re)implement the generic control mechanisms.

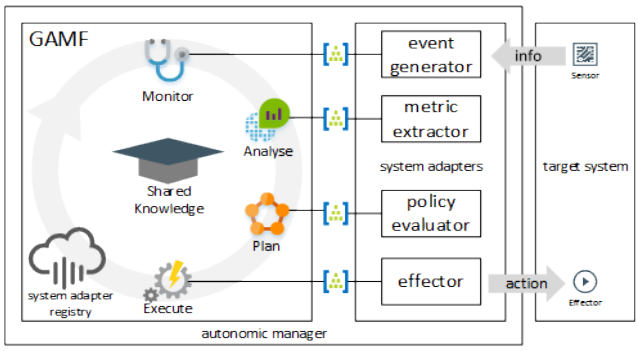

Fig. 2. GAMF Framework Architecture. GAMF provides generic control mechanisms based on the autonomic control loop and a set of interfaces that allow the interaction between the control mechanism and the system adapters

GAMF provides generic control mechanisms based on the autonomic control loop and a set of interfaces to allow the interaction between control mechanism and system-specific management components, the system adapters. System adapters include event generators and effectors, which allow interaction of the control mechanism with the target system, as well as metric extractors and policy evaluators, which provide the means for computing a specific response determined by policies to an observed situation modelled by metrics. The information about how a specific system adapter is triggered is held in the system adapters registry.

We have previously used GAMF: (i) to build an autonomic manager, which autonomously controls the maintenance scheduling of the peer-set in individual Chord nodes, governed by some high level policies [1], (ii) to apply autonomic management in a distributed storage service [18], (iii) to provide a solution towards a flexible and secure end-to-end communication in Industry 4.0 environments [19], and (iv) to implement an autonomic manager, which autonomously applies the most appropriate data transmission configuration (DTC) in a MQTT infrastructure, depending on the application security requirements and current system and environmental conditions, with the aim to improve the trade-off between data transmission security and performance [20]. In this paper we propose to extend GAMF to support also SOA-based frameworks, e.g. Arrowhead framework, by providing generic autonomic management as a service.

V. Generic Autonomic Management Service

In this section we show how we intend to integrate generic autonomic management in Arrowhead by proposing a new support system, the Generic Autonomic Management System (GAMS), which is based on the concept and our experience with the self-adaptation framework presented in Section IV.

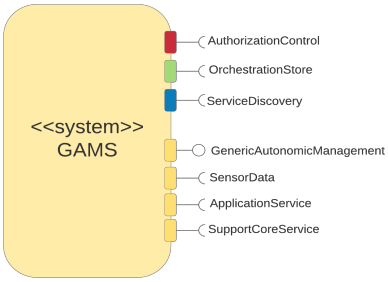

Fig. 3. GAMS system in the Arrowhead framework, which produces the *GenericAutonomicManagement* service

The GAMS system, shown in Figure 3, produces the *GenericAutonomicManagement* service and consumes the three mandatory core services of the Arrowhead framework presented in Section III. In addition, this system can consume other support core services, such as plant description, workflow manager, etc., and application services to build the shared knowledge properly by using semantics and ontologies that already exist in the Arrowhead framework.

A. Climate Control Application

To show the utility of GAMS system within the Arrowhead framework we consider a climate control application, shown in Figure 4, as a representative example.

The climate control application is relevant for smart environments, e.g. smart factories, because having a constant temperature in a production environment may be required to achieve certain quality level of products. Thus, the goal is to ensure optimal temperature conditions in a production environment in an energy-efficient way. To achieve this a temperature control loop is needed, which is used to keep a constant temperature in a specific production environment. To reduce the energy consumption the speed of an air fan can be decreased or increased in case a specific threshold is exceeded. Thus, the goal is to ensure optimal temperature conditions (e.g. between 15°C and 25°C) for a production environment in an energy-efficient way.

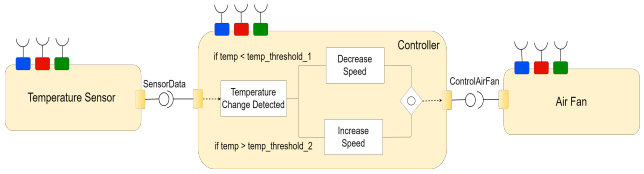

Fig. 4. Climate Control Application

Lets assume we register the services properly in the Arrowhead local cloud to create the temperature control loop. The temperature sensor provides *SensorData* service, which can be consumed by the controller to decrease or increase the speed of an air fan via the *ControlAirFan* service. As mentioned in Section III, each application system should consume at least the three mandatory core systems to be Arrowhead compliant. Thus, the services are registered in the ServiceRegistry system after being authenticated and authorized from the Authorization system and the service exchange is determined through the Orchestration system. Upon instantiation the REST web service *ControlAirFan* is called to activate the air fan. The controller listen for changes from the temperature sensor. If the temperature is below 15°C (**temp_threshold_1**) or above 25°C (**temp_threshold_2**), the next step is triggered, respectively the REST web service will be invoked to decrease or increase the speed of the air fan. The temperature control loop is executed in a loop and continuously controls the temperature once the air fan is turned on.

However, in this application various issues can occur that cannot be detected by the software controlling the air fan, e.g. the air fan might be broken. In this case the controller will try to decrease or increase the speed without reaching the desired threshold, which will increase the energy consumption. To address this issue we propose to use GAMS system, which will monitor sensor data from physical devices and analyze if the defined thresholds are met. In case they are not met an adaption policy will be chosen and executed.

B. GAMS applied to the Climate Control Application

The GAMS system, shown in Figure 5, is designed as a component-based REST service (*GenericAutonomicManagement* service) that can be invoked by different SOA-based frameworks. Additionally, given its generic property, each component of the autonomic control loop has abstract interfaces that can be used by a number of applications systems. Following, we show the utility of our approach through the climate control application.

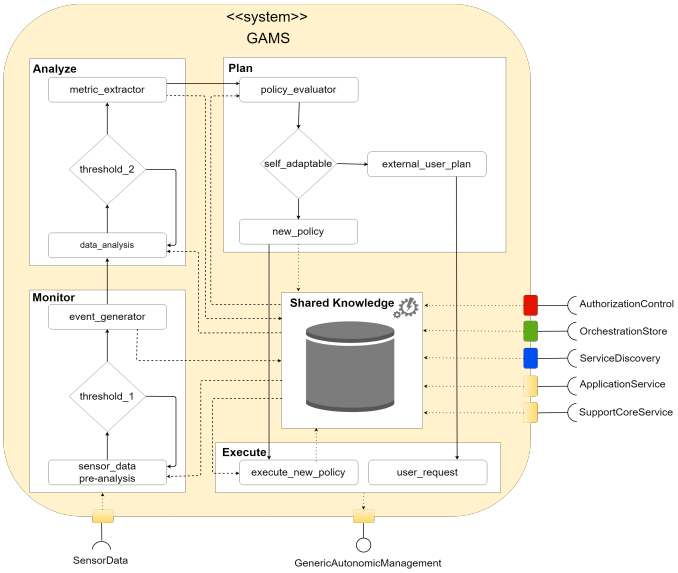

Fig. 5. GAMS as a component-based web service, which produces the *GenericAutonomicManagement* service and consumes the three mandatory core services of the Arrowhead framework. The system can consume other support core services, such as plant description, workflow manager, etc., and application services to build the shared knowledge properly.

a) Monitor: The Monitor component constantly collects monitoring data from the temperature sensor. The component performs a pre-analysis based on the incoming sensor data and context data stored in the SharedKnowledge. In case there is a significant delta an event is generated. Despite the application system that is using GAMS system, the Monitor abstract interface contains the following functions:

```
monitor[serviceID]
        getSensorData(sensorData)
        preAnalysis()
        generateEvent()
        updateSharedKnowledge()
```

For example, if the temperature is below 15°C or above 25°C for more than 3 seconds (**GAMS_threshold_1**), respectively a *Temp_Low_Check_Event* or *Temp_High_Check_Event* is generated. The generated event is fed forward to the Analyze component and stored in the SharedKnowledge.

b) Analyze: The Analyze component evaluates the events received from the Monitor component with respect to the requirements and context data in the SharedKnowledge. If the requirements cannot be satisfied a change request including a description of the metrics is send to the Plan component. The Analyze abstract interface contains the following functions:

```
analyze[serviceID]
        getRequired(required)
        getContext(context)
        getEvent(event)
        extractMetric()
        updateSharedKnowledge()
```

For example, metric *Temp_High* if more than 3 consecutive *Temp_High_Check_Event* are stored in the SharedKnowledge in 5 seconds and metric *Temp_Low* if more than 3 consecutive *Temp_Low_Check_Event* are stored in the SharedKnowledge in 5 seconds (**GAMS_threshold_2**). If the requirements are satisfied the normal execution proceeds.

c) Plan: The Plan component is able to understand the metrics received from the Analyze component and to derive adaptation policies. The Plan abstract interface contains the following functions:

```
plan[serviceID]
        getMetric(metric)
        addResource()
        releaseResource()
        updateSharedKnowledge()
```

For example, if the temperature in a specific production environment has not reached a certain threshold (e.g. *Temp_High*) even though the controller is increasing the speed via the *ControlAirFan* service, an alternative actuator (e.g., air fan) can be activated, which can guarantee the desired temperature in the room.

d) Execute: The Execute component receives the policies from the Plan component and executes the derived action via the *GenericAutonomicManagement* service. The Execute abstract interface contains the following functions:

```
execute[serviceID]
        getPolicy(policy)
        invokeNextAction()
        updateSharedKnowledge()
effectorAdd [serviceID]
effectorRelease[serviceID]
```

In case the system cannot find a suitable adaptation solution (e.g. no additional resource is available) an user intervention is required to handle the issue.

VI. Conclusion

In this paper we have proposed generic autonomic management as a service. We have shown how to integrate it in the Arrowhead framework, since it is an ideal support of the approach given its SOA nature. We have designed GAMS as a component-based service that can be invoked by different SOA-based frameworks without requiring a high adjustment effort. Additionaly given its generic property, each component of the autonomic control loop has abstract interfaces that can be used by a number of application systems. This would reduce the software engineering effort since there is no need to (re)implement the generic control mechanisms for different application systems, only to properly define events, metrics and adaptation policies. To show the utility of our approach we have used climate control application as a representative example. As future work, we will evaluate it in different application systems to show its usability.

Acknowledgment

Research leading to these results has received funding from the EU ECSEL Joint Undertaking under grant agreement n°737459 (project Productive4.0) and from the partners' national programmes/funding authorities.

References

[1] M. Tauber, G. Kirby, and A. Dearle, "Self-adaptation applied to peer-set maintenance in chord via a generic autonomic management framework," in *Self-Adaptive and Self-Organizing Systems Workshop (SASOW), 2010 Fourth IEEE International Conference on*. IEEE, 2010, pp. 9–16.

[2] J. Delsing, *Iot automation: Arrowhead framework*. CRC Press, 2017.

[3] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[4] 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.

[5] R. Asadollahi, "Starmx: A framework for developing self-managing software systems," Master's thesis, University of Waterloo, 2009.

[6] D. Weyns, S. Malek, and J. Andersson, "Forms: a formal reference model for self-adaptation," in *Proceedings of the 7th international conference on Autonomic computing*. ACM, 2010, pp. 205–214.

[7] B. Morin, O. Barais, J.-M. Jézéquel, F. Fleurey, and A. Solberg, "Models@ run. time to support dynamic adaptation," *Computer*, vol. 42, no. 10, pp. 44–51, 2009.

[8] A. Elkhodary, N. Esfahani, and S. Malek, "Fusion: a framework for engineering self-tuning self-adaptive software systems," in *Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering*. ACM, 2010, pp. 7–16.

[9] J. Whittle, P. Sawyer, N. Bencomo, B. H. Cheng, and J.-M. Bruel, "Relax: Incorporating uncertainty into the specification of self-adaptive systems," in *2009 17th IEEE International Requirements Engineering Conference*. IEEE, 2009, pp. 79–88.

[10] M. U. Iftikhar and D. Weyns, "Activforms: Active formal models for self-adaptation," in *Proceedings of the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. ACM, 2014, pp. 125–134.

[11] N. M. Villegas, G. Tamura, H. A. Müller, L. Duchien, and R. Casallas, "Dynamico: A reference model for governing control objectives and context relevance in self-adaptive software systems," in *Software Engineering for Self-Adaptive Systems II*. Springer, 2013, pp. 265–293.

[12] H. N. Ho and E. Lee, "Model-based reinforcement learning approach for planning in self-adaptive software system," in *Proceedings of the 9th International Conference on Ubiquitous Information Management and Communication*. ACM, 2015, p. 103.

[13] A. Arsanjani, L.-J. Zhang, M. Ellis, A. Allam, and K. Channabasavaiah, "S3: A service-oriented reference architecture," *IT professional*, vol. 9, no. 3, 2007.

[14] G. Di Marzo Serugendo, J. Fitzgerald, A. Romanovsky, and N. Guelfi, "Metaself-a framework for designing and controlling self-adaptive and self-organising systems," 2008.

[15] D. Menasce, H. Gomaa, J. Sousa *et al.*, "Sassy: A framework for self-architecting service-oriented systems," *IEEE software*, vol. 28, 2011.

[16] V. Cardellini, E. Casalicchio, V. Grassi, S. Iannucci, F. L. Presti, and R. Mirandola, "Moses: A framework for qos driven runtime adaptation of service-oriented systems," *IEEE Transactions on Software Engineering*, vol. 38, no. 5, pp. 1138–1159, 2012.

[17] S. Hallsteinsen, K. Geihs, N. Paspallis, F. Eliassen, G. Horn, J. Lorenzo, A. Mamelli, and G. A. Papadopoulos, "A development framework and methodology for self-adapting applications in ubiquitous computing environments," *Journal of Systems and Software*, vol. 85, no. 12, pp. 2840–2859, 2012.

[18] M.Tauber, G. Kirby, and A. Dearle, "Autonomic management of client concurrency in a distributed storage service," in *Integrated Network Management (IM), 2011 IFIP/IEEE International Symposium on*. IEEE, 2011, pp. 1109–1115.

[19] S. Maksuti, A. Bicaku, M. Tauber, S. Palkovits-Rauter, S. Haas, and J. Delsing, "Towards flexible and secure end-to-endcommunication in industry 4.0," in *IEEE 15TH INTERNATIONAL CONFERENCE OF INDUSTRIAL INFORMATICS INDIN'2017*, 2017.

[20] S. Maksuti, O. Schluga, M. Tauber, and J. Delsing, "Self-adaptation applied to mqtt via a generic autonomic management framework." in *2019 IEEE 20th International Conference on Industrial Technology (ICIT 2019)*. IEEE, 2019.

[21] A. Bicaku, S. Maksuti, C. Hegedűs, M. Tauber, J. Delsing, and J. Eliasson, "Interacting with the arrowhead local cloud: On-boarding procedure," in *2018 IEEE Industrial Cyber-Physical Systems (ICPS)*. IEEE, 2018, pp. 743–748.