Implementing autonomic Internet of Things ecosystems – practical considerations

Kumar Nalinaksh[1 [0000−0001−9656−1248]],
Piotr Lewandowski[2 [0000−0003−4035−9846]],
Maria Ganzha[2 [0000−0001−7714−4844]],
Marcin Paprzycki[2 [0000−0002−8069−2152]],
Wiesław Pawłowski[3 [0000−0002−5105−8873]], and
Katarzyna Wasielewska-Michniewska[2 [0000−0002−3763−2373]]

[1] Warsaw Management University, Kawęczyńska 36, 03-772 Warsaw, Poland
kumarnalinaksh21@gmail.com
[2] System Research Institute, Polish Academy of Sciences, Warsaw, Poland
{firstname.lastname}@ibspan.waw.pl
[3] University of Gdańsk, Wita Stwosza 57, 80-308 Gdańsk, Poland,
wieslaw.pawlowski@ug.edu.pl

Abstract. Development of next generation Internet of Things ecosystems will require bringing in (semi-)autonomic behaviors. While the research on autonomic systems has a long tradition, the question arises, are there any "off-the-shelf" tools that can be used directly to implement autonomic solutions/components for IoT deployments. The objective of this contribution is to compare real-world-based, autonomy-related requirements derived from ASSIST-IoT project pilots with existing tools.

Keywords: Internet of Things · Autonomic systems · Self-* mechanisms.

1 Introduction

The idea of autonomic systems can be traced back to early works in the discipline known as cybernetics [1]. However, the modern understanding of the concept arose from seminal work performed by IBM, within the scope of the autonomic computing initiative (ACI) [2]. Here, (and in later work [3]) IBM proposed four categories of, so called, "Self-*" properties, which were to capture main aspects for development of autonomic systems: (1) *Self-configuration*: automatic component configuration; (2) *Self-healing*: automatic fault discovery and correction; (3) *Self-optimization*: automatic resource monitoring and control to ensure optimal performance in accordance with specified requirements; (4) *Self-protection*: diligent detection and protection from random attacks. Later, seven Self-* properties have been proposed [4, 5]. Let us leave aside the number and scope of Self-* properties and come back to them later.

To realize the Self-* mechanisms, the MAPE-K (Monitor, Analyze, Plan, Execute, Knowledge) loop was proposed [3]. In the MAPE-K autonomic loop

sensors gather data about the managed element, while actuators make modifications to it. Specifically, a manager tracks the state of an element and makes adjustments using the data gathered by the sensors. As part of its development work on the Autonomic Computing Toolkit, IBM created a prototype version of the MAPE-K loop, called the Autonomic Management Engine.

Independently, recent years are characterized by rapid developments in the area of the Internet of Things. Here, the main idea is to deploy sensors and actuators, connected using heterogeneous networking infrastructures (wireless and wired), to deliver novel services for the users. With the size of IoT ecosystem deployments reaching thousands of elements, it becomes clear that it is not going to be possible to "hand manage" them. In this context, recently, the European Commission requested research in the area of Self-adaptive, Self-aware and semi-autonomous IoT systems[4]. One of the projects that was funded as a result of this call is ASSIST-IoT [5]. This project is grounded in four pilots, and each one of them has specific needs for Self-* mechanisms. This leads to the question: can these needs be satisfied using existing solutions/tools? The aim of this work is to answer this question.

In this context we proceed as follows. In Section 2 identified Self-* needs of the ASSIST-IoT pilots are discussed. Next, in Section 3 we present known to us tools that can be used in context of implementation of Self-* mechanisms. We follow, in Section 4, with discussion on how the existing solutions address the identified needs. Section 5, summarizes our findings.

2 Autonomic computing for the real-world IoT

Results of the ASSIST-IoT project will be validated in four pilots: (1) port automation, (2) smart worker protection, and (3) cohesive vehicle monitoring and diagnostics. The latter one is divided into sub-pilots dealing with (3a) car engine monitoring, and (3b) car exterior monitoring. Let us now discuss which Self-* mechanisms have been identified in each pilot, during the requirements analysis phase of the project.

2.1 Port automation pilot

Owing to the high volume of TEUs (an inexact measure of cargo capacity that is frequently employed by port authorities) handled and the growing number of stopovers, the Malta Freeport Terminal (MFTL) is nearly at capacity, with almost constant congestion in the terminal area and sporadic disruptions having a significant effect on business operations. As a consequence, four main problems can occur: (1) longer vessel dwell periods; (2) increased berthing-wait-time; (3) vessels being moved to other terminals; and (4) increased wait and turn-around times of land-side vehicles, all of which contribute to increased environmental

[4] https://ec.europa.eu/info/funding-tenders/opportunities/portal/screen/opportunities/topic-details/ict-56-2020

[5] https://assist-iot.eu/

load, transportation inefficiency, and cost of efficient movements. To deal with the existing threats, using solutions provided by the ASSIST-IoT project, three business scenarios have been identified: (1) asset monitoring in the terminal yard, (2) automated container handling equipment cooperation, and (3) rubber-tired gantry remote control, with augmented reality assistance. All of those scenarios will need Self-* capabilities, that will work seamlessly in heterogeneous IoT environment.

Scenarios (1), (2) and (3) will need Self-inspection (sometimes called Self-awareness) to understand where particular assets are located and what is the current state of those assets. Self-healing and Self-diagnosis will also be impactful as they will allow to automatically detect issues, autonomously fix some of them or call for human operator as a last resort. (1) will also require Self-configuration capabilities so that new devices can easily connect and acquire required configuration (i.e. map of the port). (2) will additionally need Self-organization and Self-adaptation to autonomously carry on container handling via organizing work efficiently and by adapting to a changing port environment.

2.2 Smart safety of workers pilot

Construction companies and relevant administration agencies, such as the European Agency for Safety and Health at Work, place a high emphasis on compliance with workplace safety and health standards and risk management at small or large, private or public construction projects. A vast number of people with varying degrees of knowledge and experience collaborate with each other, control equipment, and interface with heavy machinery on each building site, which is occupied by many subcontracted firms. Their experience, best practises, and risk management culture offer a layer of security for construction workers, but it does not ensure that all accidents could be avoided. Accidents will happen in a split second with no indications. Furthermore, unless appropriate monitoring mechanisms are in place, a potentially life-saving immediate intervention to an accident couldn't be feasible.

ASSIST-IoT solution will enable this pilot to collect accurate and appropriate data in order to produce intelligent insights for the protection of all people involved at every work site within a vast construction site. Such data and observations, along with the clear implementation of data security policies, will advance understanding and increase awareness about workplace safety, as well as lead to the digital transformation of construction processes that retains the employee at the leading edge. In this application area, the main goal of ASSIST-IoT is to prevent and detect common Occupational Safety and Health (OSH) hazards such as stress, exhaustion, overexposure to heat and ultraviolet rays, slips, trips, falls from heights, suspension injuries, lack of mobility due to loss of consciousness, collision with heavy equipment, entrapment and PPE misuse. The success of implementing this pilot test-bed would result in two key outcomes: better working conditions for thousands of workers and a clear return on investment (ROI) for the facility. This pilot has been divided into four business

scenarios, (1) occupation safety and health monitoring; (2) fall arrest monitoring; (3) safe navigation; and (4) health and safety inspection support.

(1), (2), (3) and (4) will need Self-inspection to understand where particular events are taking place or if (and where) someone is accessing dangerous zone. Self-diagnosis is required so that whole system can autonomously detect any potential issues. (1) and (3) will also need Self-configuration to automatically connect upcoming devices and ensure that up-to-date configuration in dynamic construction site environment is available; (3) will need Self-adaptation so that in case of dynamically occurring risks, the safest route can be selected.

2.3 Cohesive vehicle monitoring and diagnostics pilot

Currently, ICT penetration in the automobile industry is just a fraction of what it should be, and it is mostly dominated by car manufactures. Because of high costs and bandwidth problems, communication between vehicle fleets and original equipment manufacturers is also restricted. Due to safety and security concerns, most IoT integration in vehicles programs struggle to incorporate data from various sources (e.g. industry data, environmental data, data from inside the car, historical vehicle maintenance data) and to obtain access to vehicle data. Although real-time operation of a moving vehicle creates safety risks and therefore prohibits full unrestricted access to the information and control firmware, there is no theoretical obstacle to trustworthy third parties having access to onboard sensor measurements for diagnostics and monitoring. Furthermore, no existing application or implementation incorporates and delivers automotive details to a customer in an immersive friendly atmosphere based on their position and relationship with the vehicle, avoiding recalls.

The use of the ASSIST-IoT reference architecture in this pilot will improve the automotive OEMs' ability to track the pollution standards of vehicles that are currently on the road in order to ensure that the fleet maintains certification limits over its lifespans. Monitoring fleet pollution levels allows for the prompt execution of corrective measures, if necessary, to return them to acceptable levels. There are two independent sections of this pilot: (1) a Ford initiative and (2) a TwoTronic initiative. The Ford initiative is divided into two business scenarios: (1) fleet in-service conformity verification; and (2) vehicle diagnostics; while the TwoTronic initiative deals with vehicle exterior condition inspection and documentation.

Scenarios (1) and (2) will need Self-learning to constantly improve their capabilities and Self-diagnosability to ensure that all components of the system provide realistic measurements. We also assumed that Self-configuration will be required to always be up-to-date with current requirements.

3 State-of-the-art in tools for autonomic computing

Let us now summarize the state-of-the-art in the area of tools that can be used to implement Self-* mechanisms. Most important factor taken into account was

their out-of-the-box Self-* capability. Moreover, tools were selected on having publicly available (open source) code repositories that have recent updates indicting that these tools are currently under active development. Another factor was the potential to generalize particular tool to solve novel problems.

AMELIA: Analysable Models Inference In [9] authors report on tracking IoT system trajectories, i.e. a series of latitude and longitude coordinate points mapped with respect to time [10], in a complex spatial context. This is combined with accessible space landmarks, to create graph-based spatial models. These are, in turn, analysed by the MAPE-K loop's Analyse feature, to search for goal and requirement violations, during system runtime. The project is available as a virtual environment, in which it can be run and the findings replicated as published. The authors run the simulations[6] using actual data sets derived from Taxis (IoT devices) and used city's landmarks as the graph's nodes. The project is primarily built on Python with MongoDB as it can resolve geo-spatial queries. Shell scripts are used to interface between the project and the operating system. If required, the project can be built and run locally with different parameters.

OCCI-compliant sensor management The Open Cloud Computing Interface (OCCI) specifies an API for managing large and diverse cloud services that is independent of the service provider. Various tools offer interfaces for identifying, initiating, and implementing modifications to complex cloud environments. The authors built an OCCI monitoring extension[7] in JAVA that offered managing the implementation and setup of monitoring sensors in the cloud [11]. In an OCCI-compliant runtime model, sensors and their monitoring results are described. This extension transforms the OCCI runtime model into a knowledge base that, when coupled with the other objects in the OCCI ecosystem, facilitates full control loops for Self-adaptation into cloud systems. The authors integrated the project with a real-world cloud infrastructure and included two sample scenario implementations for other researchers using the test environment to validate the project outcomes. A Hadoop cluster was implemented and dynamically scaled in both instances.

PiStarGODA-MDP: A Goal-Oriented Framework to Support Assurances Provision A Self-adaptive system often works in a complex and partly unknown context, which introduces uncertainty that must be addressed in order for it to accomplish its objectives. Furthermore, several other types of uncertainties exist, and the causes of these uncertainties are not consistently resolved by current approaches in the Self Adaptive System (SAS) life cycle. This begets the question of how can the goals of a system that is subject to continuous uncertainties be guaranteed? Here, the authors proposed and implemented a goal-oriented assurance method that allows monitoring sources of uncertainty

[6] https://dsg.tuwien.ac.at/team/ctsigkanos/amelia/
[7] https://gitlab.gwdg.de/rwm/de.ugoe.cs.rwm.mocci

that arise during the design phase, or during runtime execution of a system [12]. The SAS is designed with the goals in mind, and the Self-adaptation occurs during the runtime. GORE (Goal-Oriented Specifications Engineering) is used for separating technological and non-technical criteria into clearly specified goals and justifications for how to accomplish them. These goal models are converted into reliability and cost parametric formulae using symbolic model checking, which are then used as runtime models to express the likelihood of SAS goals being reached. Based on the principle of feedback control, the controller continuously monitors the managed system's costs and reliability statuses, as well as contextual constraints, at runtime to address parameterized uncertainties. The runtime models are then used to assess (i) system's reliability and cost, and (ii) policy measures that should be activated to accomplish the goals, influencing SAS adaptation decisions. The authors evaluated their project's[8] approach using the Body Sensor Network (BSN) implemented in OpenDaVINCI[9] and were able to effectively provide guarantees for Self-adaptive systems' goals. JavaScript and Java were used in the project's development. Heroku hosts the pistarGODA modelling and analysis environment.

TRAPP: Traffic Reconfiguration through Adaptive Participatory Planning Traffic management is a difficult challenge from the standpoint of Self-adaptation because it is hard to prepare ahead with all potential scenarios and behaviours. Here, authors present a method for autonomous agents to collaborate in the absence of a centralised data collection and arbitrator [13]. TRAPP integrates the SUMO [14] and EPOS [15] frameworks. EPOS is a decentralised combinatorial optimization approach for multi-agent networks, while SUMO is a simulation environment for traffic dynamics. SUMO sends EPOS a list of potential routes for each vehicle, and EPOS generates the designated plan for each vehicle, which SUMO picks up and executes. The mechanism described above occurs on a regular basis. Periodical adaptation cycles are operated by the managing system, which, in accordance with the MAPE-K loop, monitor data, evaluate it for traffic issues or anomalies, schedule subsequent activities to adjust the way participatory preparation occurs, and eventually perform the adaptation actions by configuring EPOS accordingly. The revised configuration is used the next time EPOS is invoked. The authors run simulations[10] by deploying 600 cars in the city of Eichstatt, which has 1131 roads. Python and Jupiter notebook were used to create the project.

mRUBiS: Model-Based Architectural Self-Healing and Self-Optimization Self-adaptive software is a restricted system that uses a feedback mechanism to adjust to changes in the real world. This mechanism is implemented by the adaptation engine, while the domain logic is realised by adaptable software

[8] https://github.com/lesunb/pistarGODA-MDP
[9] https://github.com/se-research/OpenDaVINCI
[10] https://github.com/iliasger/TRAPP

and controlled by the engine. The authors came to the conclusion that there is no off-the-shelf product for designing, testing, and comparing model-based architectural Self-adaptation and hence they developed mRUBIs [17]. It simulates adaptable software and allows for "issues" to be injected into runtime models. This helps developers to test and compare different adaptation engine variants as well as validate the Self-adaptation and healing properties of the adaptation engine. The authors developed a generic modelling language called "CompArch" to interact with the project[11], while the project itself has been implemented in JAVA.

Lotus@Runtime: Tool for Runtime Monitoring and Verification of Self-adaptive Systems Lotus@Runtime tracks execution traces provided by a Self Adaptive System and annotates the probability of occurrence of each system operation using a Labelled Transition System model [18]. In addition, the probabilistic model is used at runtime to check adaptability properties. A warning function built into the tool notifies the Self-adaptive device if a property is violated. These notifications are handled by ViolationHandler module that the user implements during planning phase. The project[12] is based over the existing LoTuS[13] project built in JAVA. The authors used Tele Assistance System (TAS) and Travel Planner Application (TPA) [19] for validating the project[14].

Intelligent Ensembles Autonomous components are deployed in a physical world in smart cyber-physical systems (CPS) like smart cities, where they are supposed to collaborate with each other and also with humans. They must be capable of working together and adapt as a group to deal with unexpected circumstances. To address this problem, the authors applied Intelligent Ensembles. They're dynamic groups of components that are generated at runtime depending on the components' current state. Components are not capable of communicating with one another; rather, the ensemble is responsible for communication. The Intelligent Ensembles framework uses a declarative language called "EDL" for describing dynamic collaboration groups [20]. The project[15] is built over the Eclipse Modelling Framework and the Z3 SMT solver.

CrowdNav and RTX The authors look at the issue of a crowdsourced navigation system (CrowdNav). It's a city traffic control system that gathers data from a variety of sources, such as cars and traffic signals, and then optimises traffic guidance. The authors solve this problem by interpreting and adapting the stream of data from the distributed system using Real-Time Experimentation

[11] https://github.com/thomas-vogel/mRUBiS
[12] https://github.com/davimonteiro/lotus-runtime
[13] https://github.com/lotus-tool/lotus-tool
[14] https://drops.dagstuhl.de/opus/volltexte/2017/7145/
[15] https://drops.dagstuhl.de/opus/volltexte/2017/7144/

(RTX) tool [21]. The project[16] is written in Python, configures Kafka and Spark and links them together. Its architecture is straightforward and restricted to the most relevant input and output parameters, with Big Data analytics guiding Self-adaptation based on a continuous stream of operational data from CrowdNav. To help in the assessment of different Self-adaptation strategies for dynamic large-scale distributed systems, the authors built a concrete model problem using CrowdNav and SUMO in this exemplar [16, 14].

DeltaIoT: Self-Adaptive Internet of Things Wireless connectivity absorbs the majority of energy in a standard IoT unit, so developing reliable IoT systems is critical. Finding the correct network configurations, on the other hand, is difficult because IoT implementations are subject to a multitude of uncertainties, such as traffic load fluctuations and connectivity interruption. Self-adaptation enables hand-tuning or over-provisioning of network settings to be automated.A feedback loop is installed on top of the network to track and measure the motes and the environment, allowing the IoT system to adapt autonomously. The DeltaIoT project[17] consists of an offline simulator and a physical setup of 25 mobile nodes which can be remotely controlled for field testing. The IoT system is installed on the KU Leuven Computer Science Department's property. DeltaIoT [22] is the very first Self-adaptation research project to have both a simulator and a physical system for testing. DeltaIoT is used in Self-adaptation studies. It allows researchers to test and compare emerging Self-adaptation approaches, techniques, and resources in the IoT. The WebService Engine is a user interface for inspecting and controlling the Internet of Things system. A WSDL file is used to describe this interface. Just one person may do Self-adaptation at a time, hence accessibility to the web service is restricted.

TAS: Tele Assistance System TAS [23] was created with the help of the Research Service Platform (ReSeP)[18]. ReSeP is built upon the Service-Oriented Architecture (SOA) principles using JAVA. The tool is an example of a service-based system (SBS). It gives preventive care to chronic patients in their own homes. TAS makes use of sensors mounted in a wearable interface, and remote services from healthcare, pharmacy, and emergency response providers. Periodic samples of a patient's critical parameters are taken and exchanged with a medical service for study. The service may invoke a pharmacy service based on the review to distribute new medication to the patient or to change and upgrade the medication dose. Using ReSeP the authors defined two different adaptation policies and validated it with TAS. The first policy was to retry twice in case of service failure whereas the second policy selects an alternate service with similar cost and invokes it. The experiment found that the first policy kept the costs low but failed the reliability constraint while the second one passed successfully albeit with high costs.

[16] https://drops.dagstuhl.de/opus/volltexte/2017/7143/
[17] https://people.cs.kuleuven.be/ danny.weyns/software/DeltaIoT/
[18] https://github.com/davimonteiro/resep

DEECo: Dependable Emergent Ensembles of Components The authors conclude that developing complex Self-adaptive smart CPS is a difficult challenge that current software engineering models and methods can only partially solve. The appropriate architecture of a smart CPS adopts a holistic view that considers the overall system goals, operating models that include system and climate uncertainties, and the communication models that are being used. To answer these issues the authors used DEECo [25]. It's a model and framework for creating sophisticated smart CPS. It also provides precise information about the consequences of adaptation techniques in complex smart CPS. The Java and C++ are included with the DEECo component model [24]. The C++ architecture is used for real-world deployment on embedded devices, like the STM32F4 MCU. Java i.e. JDEECo, on the other hand, is used for adaptation and autonomous components simulation. JDEECo simulates implementations of hybrid network environments, mixing IP networks and mobile/vehicle ad-hoc networks (MANETS/VANETS), as seen in current smart-* systems, by using the OMNeT extensions INET and MIXIM. The project[19] was created specifically for the purpose of developing and simulating dynamic Self-adaptive smart CPS. Authors used a smart parking scenario to validate its usage.

Znn.com Rainbow [26] is a framework for designing a system with Self-adaptive, run-time capabilities for monitoring, detecting, deciding, and acting on system optimization opportunities. Znn.com is an N-tier-style web-based client-server system. Rainbow uses the following guidelines to handle Znn.com's Self-adaptation at peak periods: (i) Changing the server pool size and (ii) switching between textual and multimedia response [27]. The project[20] has been built using several different languages such as PHP, Shell, Brainfuck, Awk, HTML and Perl etc.

Dragonfly The authors [28] noted that when designing cyber-physical Systems, we often encounter defiant systems that can evolve and collaborate to achieve individual goals but struggle to achieve global goals when combined with other individual systems. They suggest an integration strategy for converting these defiant systems to also achieve the overall objectives. Dragonfly[21] is a drone simulator that allows users to simulate up to 400 drones at once. Simulations may be performed in both regular and unusual conditions. The wrappers implement the drones' adaptive behavior and enable runtime adaptation. The simulator is built using JAVA, AspectJ, HTML and Docker.

DARTSim Cyber-Physical systems make use of Self-adaptive capabilities to autonomously manage uncertainties at the intersection of the cyber and physical worlds. Self-adaptation-based approaches face several challenges such as:

[19] https://github.com/d3scomp/JDEECo
[20] https://github.com/cmu-able/znn
[21] https://github.com/DragonflyDrone/Dragonfly

(i) sensing errors while monitoring environment, (ii) not being able to adapt in time due to physical constraints etc, (iii) objectives that cannot be coupled together in a single utility matrix such as providing good service vs avoiding an accident. To evaluate and compare various Self-adaptation approaches aiming to address these unique challenges of smart CPS, DARTSim was created in 2019 [29]. DARTSim is a simulation of an autonomous team of unmanned aerial vehicles (UAVs) conducting a reconnaissance mission in a hostile and unfamiliar area. The squad must follow a predetermined path and pace when attempting to locate the targets. The lower it goes, the more likely it is to find targets, but also the more likely it is to be killed by threats. The high-level Self-adaptation decisions that the machine must make to achieve mission success are the subject of DARTSim. This sCPS has the "smartness" needed to conduct the task autonomously thanks to the adaptation manager who makes these decisions. When a mission detects at least half of the threats, it is considered effective. The project[22] outcome is available as a C++ library or via a TCP interface[23].

4 Needs vs available tools – critical analysis

Existing tools/platforms vary in application and abstraction level. Their range of capabilities varies from solving specific problem using specific type of IoT Device (TRAPP) to a high-level generic ones that need non-trivial amount of additional work to solve concrete tasks (e.g. OCCI-compliant, fully causal-connected runtime models supporting sensor management). In this context, let us consider existing tools and evaluate their potential to deliver Self-* mechanisms identified within the ASSIST-IoT pilots. In total, to satisfy all pilot requirements the following Self-capabilities were identified: (1) Self-inspection (or Self-awareness), (2) Self-diagnosis, (3) Self-healing, (4) Self-configuration, (5) Self-organization, (6) Self-adaptation, and (7) Self-learning.

When considering available solutions, we verified whether their public source code repositories were available and then focused on the fact whether the Self-* capabilities were available out of the box or with minimal additional work required. Following is a list of considered solutions with Self-* capabilities that they support:

AMELIA (1) Self-inspection, (6) Self-adaptation, (7) Self-learning

OCCI (1) Self-inspection, (2) Self-diagnosis, (3) Self-healing, (6) Self-adaptation, (7) Self-learning

PiStarGODA-MDP (1) Self-inspection, (6) Self-adaptation, (7) Self-learning

TRAPP (6) Self-adaptation, (7) Self-learning.

mRUBiS (1) Self-inspection, (2) Self-diagnosis, (3) Self-healing, (5) Self-organization, (6) Self-adaptation, (7) Self-learning

[22] https://github.com/cps-sei/dartsim
[23] https://hub.docker.com/r/gabrielmoreno/dartsim/

Lotus@Runtime: (1) Self-inspection, (2) Self-diagnosis, (3) Self-healing, (6) Self-adaptation

Intelligent Ensembles (6) Self-adaptation

CrowdNav and RTX (6) Self-adaptation, (7) Self-learning

DeltaIoT (1) Self-inspection, (2) Self-diagnosis, (5) Self-organization, (6) Self-adaptation

TAS: (1) Self-inspection, (5) Self-organization, (6) Self-adaptation

DEECo (1) Self-inspection, (5) Self-organization, (6) Self-adaptation

Znn.com (1) Self-inspection, (6) Self-adaptation

Dragonfly (1) Self-inspection, (6) Self-adaptation

DARTSim (1) Self-inspection, (6) Self-adaptation

In summary, it is easy to observe that there is no available solution capable of running heterogeneous Self-* IoT deployments that satisfies needs of all pilots considered by ASSIST-IoT. Particularly, Self-configuration seems to be a missing component. If no tool is able to provide common abstraction for detecting and automatically connecting and configuring various devices that are present in an IoT ecosystem it will be hard to imagine a widespread adoption of IoT-based solutions. As a general note, most projects followed a very high-level approach to Self-*, leaving implementation of components below MAPE-K loop (or analogous solution) to the user. This is understandable, as most of them were not designed with IoT deployments in mind, yet widespread adoption needs to be preceded by developing a well-rounded solution that answers the common Self-* problems on a more concrete level. There is a set of Self-* enabled tools that focus on selected problems (for example TRAPP is specific to a car traffic management) but they are very hard to generalize to conveniently handle as diverse scenarios as worker safety, coordination between port machinery and detect defects in car exhaustion system. Those expectations might sound very ambitious, yet this is a general trend in Software Engineering, where Cloud-based solutions abstracted away many of the hard problems to the point of few clicks in web-based UI. We predict that the same is required in IoT based environments.

5 Concluding remarks

The aim of this work is to consider how existing autonomic computing solutions match actual needs of Internet of Things deployments. Proceeding in this directions we have, first, outlined requirements related to autonomic computing, in 4 real-world pilots, grounding the work to be completed in the ASSIST-IoT project. Second, we have summarized state-of-the-art of existing ready-to-use tools that are claimed to support implementation of autonomic systems. Finally, we have matched the two, and critically analysed the results.

Overall, we conclude that there is no solution available that can address all challenges that have been identified in ASSIST-IoT, in the context of applying Self-* in considered business scenarios and use cases. The existing solutions

would need to be adapted and combined to cover the set of required features. Additionally, some of them would need to be verified for their adaptability and performance in heterogeneous IoT ecosystems that are very ambitious target environment, for the technological solutions. We foresee that ASSIST-IoT will not only give opportunity to verify a set of approaches proposed so far, in a real-life deployments, but will also advance state-of-the-art in Self-* systems in terms of providing Self-* capabilities for IoT-centric environments.

Acknowledgment

Work of Maria Ganzha, Piotr Lewandowski, Marcin Paprzycki, Wiesław Pawłowski and Katarzyna Wasielewska-Michniewska was sponsored by the ASSIST-IoT project, which received funding from the EU's Horizon 2020 research and innovation program under grant agreement No. 957258.

References

1. Cybernetics: Or Control and Communication in the Animal and the Machine. Paris, (Hermann & Cie) & Camb. Mass. (MIT Press) ISBN 978-0-262-73009-9; 1948, 2nd revised ed. 1961.
2. J. O. Kephart and D. M. Chess, "The vision of autonomic computing," in Computer, vol. 36, no. 1, pp. 41-50, Jan. 2003, doi: 10.1109/MC.2003.1160055.
3. IBM, "An Architectural Blueprint for Autonomic Computing," IBM White Paper, 2005. [online] Available: https://www-03.ibm.com/autonomic/pdfs/AC%20Blueprint%20White%20Paper%20V7.pdf
4. Poslad, Stefan (2009). Autonomous systems and Artificial Life, In: Ubiquitous Computing Smart Devices, Smart Environments and Smart Interaction. Wiley. pp. 317–341. ISBN 978-0-470-03560-3
5. Nami, M.R., Bertels, K.: A Survey of Autonomic Computing Systems. In: Proceedinbgs of the 3rd International Conference on Autonomic and Autonomous Systems, pp. 26–30 (2007)
6. What is autonomic computing? (2018, August 1). OpenMind. Retrieved May 15, 2021, from https://www.bbvaopenmind.com/en/technology/digital-world/what-is-autonomic-computing/
7. J. Kramer and J. Magee, "Self-Managed Systems: an Architectural Challenge," Future of Software Engineering (FOSE '07), 2007, pp. 259-268, doi: 10.1109/FOSE.2007.19.
8. S. Sarma et al. (2015). Cyberphysical-System-on-Chip (CPSoC): A Self-Aware MPSoC Paradigm with Cross-Layer Virtual Sensing and Actuation. In Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition (pp. 625–628). EDA Consortium.
9. C. Tsigkanos, L. Nenzi, M. Loreti, M. Garriga, S. Dustdar and C. Ghezzi, "Inferring Analyzable Models from Trajectories of Spatially-Distributed Internet of Things," 2019 IEEE/ACM 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), 2019, pp. 100-106, doi: 10.1109/SEAMS.2019.00021.
10. Yu Zheng and Xiaofang Zhou. Computing with spatial trajectories. Springer Science & Business Media, 2011.

11. J. Erbel, T. Brand, H. Giese and J. Grabowski, "OCCI-Compliant, Fully Causal-Connected Architecture Runtime Models Supporting Sensor Management," 2019 IEEE/ACM 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), Montreal, QC, Canada, 2019, pp. 188-194, doi: 10.1109/SEAMS.2019.00032
12. G. Félix Solano, R. Diniz Caldas, G. Nunes Rodrigues, T. Vogel and P. Pelliccione, "Taming Uncertainty in the Assurance Process of Self-Adaptive Systems: a Goal-Oriented Approach," 2019 IEEE/ACM 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), Montreal, QC, Canada, 2019, pp. 89-99, doi: 10.1109/SEAMS.2019.00020
13. I. Gerostathopoulos and E. Pournaras, "TRAPPed in Traffic? A Self-Adaptive Framework for Decentralized Traffic Optimization," 2019 IEEE/ACM 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), Montreal, QC, Canada, 2019, pp. 32-38, doi: 10.1109/SEAMS.2019.00014.
14. P. A. Lopez et al. "Microscopic traffic simulation using sumo," in The 21st IEEE International Conference on Intelligent Transportation Systems. IEEE, 2018. [Online]. Available: https://elib.dlr.de/124092/
15. E. Pournaras, P. Pilgerstorfer, and T. Asikis, "Decentralized collective learning for self-managed sharing economies," ACM Transactions on Autonomous and Adaptive Systems (TAAS), vol. 13, no. 2, p. 10, 2018.
16. Chen, C., Liu, Y., Kreiss, S., & Alahi, A. (2019). Crowd-robot interaction: Crowd-aware robot navigation with attention-based deep reinforcement learning. In 2019 International Conference on Robotics and Automation (ICRA) (pp. 6015–6022).
17. Thomas Vogel. 2018. MRUBiS: an exemplar for model-based architectural self-healing and self-optimization. In Proceedings of the 13th International Conference on Software Engineering for Adaptive and Self-Managing Systems (SEAMS '18). Association for Computing Machinery, New York, NY, USA, 101–107. DOI: https://doi.org/10.1145/3194133.3194161
18. D. M. Barbosa, R. G. De Moura Lima, P. H. Mendes Maia and E. Costa, "Lotus@Runtime: A Tool for Runtime Monitoring and Verification of Self-Adaptive Systems," 2017 IEEE/ACM 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), 2017, pp. 24-30, doi: 10.1109/SEAMS.2017.18.
19. Liangzhao Zeng, B. Benatallah, A. H. H. Ngu, M. Dumas, J. Kalagnanam and H. Chang, "QoS-aware middleware for Web services composition," in IEEE Transactions on Software Engineering, vol. 30, no. 5, pp. 311-327, May 2004, doi: 10.1109/TSE.2004.11.
20. F. Krijt, Z. Jiracek, T. Bures, P. Hnetynka and I. Gerostathopoulos, "Intelligent Ensembles - A Declarative Group Description Language and Java Framework," 2017 IEEE/ACM 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), 2017, pp. 116-122, doi: 10.1109/SEAMS.2017.17.
21. S. Schmid, I. Gerostathopoulos, C. Prehofer and T. Bures, "Self-Adaptation Based on Big Data Analytics: A Model Problem and Tool," 2017 IEEE/ACM 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), 2017, pp. 102-108, doi: 10.1109/SEAMS.2017.20.
22. M. U. Iftikhar, G. S. Ramachandran, P. Bollansée, D. Weyns and D. Hughes, "DeltaIoT: A Self-Adaptive Internet of Things Exemplar," 2017 IEEE/ACM 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), 2017, pp. 76-82, doi: 10.1109/SEAMS.2017.21.

23. D. Weyns and R. Calinescu, "Tele Assistance: A Self-Adaptive Service-Based System Exemplar," 2015 IEEE/ACM 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, 2015, pp. 88-92, doi: 10.1109/SEAMS.2015.27.
24. J. Keznikl, T. Bureš, F. Plášil and M. Kit, "Towards Dependable Emergent Ensembles of Components: The DEECo Component Model," 2012 Joint Working IEEE/IFIP Conference on Software Architecture and European Conference on Software Architecture, 2012, pp. 249-252, doi: 10.1109/WICSA-ECSA.212.39.
25. T. Bures, I. Gerostathopoulos, P. Hnetynka, J. Keznikl, M. Kit, and F. Plasil, "DEECo: An ensemble-based component system," in Proceedings of CBSE '13, Vancouver, Canada, 2013, pp. 81–90
26. S.-W. Cheng. Rainbow: Cost-Effective Software Architecture-Based Self-Adaptation. Ph.D. Dissertation, TR CMUISR-08-113, Carnegie Mellon University School of Computer Science, May 2008.
27. S. Cheng, D. Garlan and B. Schmerl, "Evaluating the effectiveness of the Rainbow self-adaptive system," 2009 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems, Vancouver, BC, Canada, 2009, pp. 132-141, doi: 10.1109/SEAMS.2009.5069082.
28. P. H. Maia, L. Vieira, M. Chagas, Y. Yu, A. Zisman and B. Nuseibeh, "Dragonfly: a Tool for Simulating Self-Adaptive Drone Behaviours," 2019 IEEE/ACM 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), Montreal, QC, Canada, 2019, pp. 107-113, doi: 10.1109/SEAMS.2019.00022.
29. G. Moreno, C. Kinneer, A. Pandey and D. Garlan, "DARTSim: An Exemplar for Evaluation and Comparison of Self-Adaptation Approaches for Smart Cyber-Physical Systems," 2019 IEEE/ACM 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS),