IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Date of publication xxxx 00, 0000, date of current version December 30, 2020.

*Digital Object Identifier XX.XXXXX/ACCESS.XXXX.DOI*

# Relaxing Platform Dependencies in Agent-based Control Systems

**MARCO PÉREZ HERNÁNDEZ, (Member, IEEE), DUNCAN MCFARLANE, AJITH PARLIKAD, MANUEL HERRERA, (Member, IEEE), and AMIT KUMAR JAIN**
Institute for Manufacturing, University of Cambridge, 17 Charles Babbage Road, CB3 0FS, Cambridge, UK

Corresponding author: Marco Pérez Hernández (e-mail: mep53@cam.ac.uk).

This work was supported by the Next Generation Convergent Digital Infrastructure Project funded by Engineering and Physical Sciences Research Council (EPSRC) and British Telecom (BT) under grant EP/R004935/1.

**ABSTRACT** Agent-based systems have been widely used to develop industrial control systems when they are required to address issues such as flexibility, scalability and portability. The most common approach to develop such control systems is with agents embedded in a platform that provides software libraries and runtime services that ease the development process. These platforms also bring challenges to the agent-based control system engineering. Firstly, they influence the control system design, for example, by assuming the need of a global directory of agents even if this is not required. Hence, introducing unnecessary overhead to the control system that can worsen when it grows. Secondly, as agents are embedded in the platform, it also constraints the deployment of agents across available edge, fog and cloud computing infrastructures. This paper addresses these challenges through an approach to build agent-based control systems, that relaxes the dependencies in multiagent system (MAS) platforms, through the use of container-based virtualisation. This approach enables the implementation of agents as self-contained applications that can be deployed in independent environments but still are able to communicate and coordinate with other agents of the control system. We built a prototype and evaluated this approach in the context of a case study for the supervisory control of digital network infrastructures. This case study enabled us to demonstrate feasibility of the approach and to show the flexibility, of the resulting control system, to adopt several topologies as well as to operate at different scales, over emulated networks. We also concluded that designing agents as individual deployment units is also cost-effective especially in control scenarios with low number of stable agents.

**INDEX TERMS** Agent-based control, Multiagent systems, Micro services, Container-based virtualisation Industrial control, System containers

## I. INTRODUCTION

Modern industrial systems undergo a continuous and massive demand for highly personalised services and products. Flexibility to efficiently produce or provision services of different characteristics as well as the ability to scale up and down this production/provision become key requirements of such industrial systems. Likewise, the control systems should act as facilitators for addressing these requirements and therefore exhibit also scalability and flexibility, while avoiding compromises in performance and effectiveness. Rapid changes in the controlled system implies either a highly flexible control architecture –which is computationally expensive– or very short cycles of re-adjustment and re-tuning to the new conditions by control designers.

Although these requirements have been widely addressed by the research community (section II), recent trends towards cloud manufacturing [1] and software-based services [2] tighten demands for control systems. The availability of diverse computing infrastructures has opened another dimension of flexibility, closely linked to portability, where the system should be able to be deployed across available infrastructure from edge to cloud passing through fog hardware resources [3]. This way the control system can take advantage of the best deployment strategies in order to reduce data transmission to the cloud and latency when the industrial system demands it. As production and service provision systems increasingly rely on software components –e.g. web services– that enable quick reconfiguration of resources and

capabilities, the control systems should operate with similar speed and exhibit the ability to shrink or expand (e.g. in terms of the number of controlling entities) according to demand patterns.

The agent-based paradigm has demonstrated to be effective in addressing different requirements of flexibility (e.g. mix, volume, product, etc.) identified for industrial systems. However, existing approaches for agent-based control are usually coupled to an agent platform which operates on controlled or semi controlled environments. When addressing flexibility and scalability this is a problem, as usually agent platforms have their own limitations and evolution cycles. For example, some of the platforms are strictly academic efforts while others are industry initiatives. Some platforms are more aligned to standards than others and the openness is limited as usually agents implemented in one platform cannot be accommodated in another one [4]. Coupling the control system to the platform brings uncertainty as the platform might not evolve as the control systems requires.

Latest advances in virtualisation and *microservices* architectural patterns have incorporated new approaches and technologies for engineering of complex software systems [5]. Jointly used, these are tools that enable building systems from loosely coupled and distributed services that can evolve independently of each other and are easily portable across the available computing infrastructure [6]. Few efforts have been found in the literature of agent-based control systems that do not rely in conventional MAS platforms but try to combine advantages of agents and these technologies and architectural patterns.

The main contributions of this paper are threefold. First, it proposes an approach for engineering of agent control systems, using container-based virtualisation and microservices architectural patterns as an alternative to conventional approaches where the agent-based control system is coupled to a MAS platform. Second, the paper introduces a reference architecture that shows how this approach can be used to build the mentioned control systems. Thirdly, it discusses a case study in the context of next generation digital network infrastructures where the benefits of this approach are demonstrated, initially at small scale, through a prototype operating in a realistic environment, and later, at a larger scale, via simulation experiments.

This paper is organised as follows. Section II reviews the relevant literature on agent-based control systems, the use of MAS platforms and the tools used in this work. Next, section III introduces the overall approach, discusses how global directory of agents are substituted and how the agents are decoupled from a platform, the end of this section presents relevant design considerations. Section IV elaborates our second contribution by introducing a reference architecture to build industrial systems using control behaviours and containerised agents. The section V describes the case study and the evaluation approach, it also discuses the results obtained and the lessons learned. Finally, section VI summarises concluding remarks and future work.

## II. RELATED WORK

This section reviews key contributions in three directions: the motivation for agent-based control systems, the development approach for such systems and a set of software tools with potential to complement current development approaches.

### A. RATIONALE FOR MULTIAGENT-BASED CONTROL SYSTEMS

Modern industrial control systems aim to meet multiple non-functional requirements. Among these, flexibility [7], scalability [8] and portability [9] are fundamental to ensure their continuous evolution and long-lasting operation. Flexibility refers to how easy the system can be modified for applications or environments that are different to the ones it was initially designed for [10]. In manufacturing control, flexibility refers to the ability to change the mix of products, volumes, the sequence of operations in which they are produced or the flows of material, among others [11], [12]. Portability is defined as "the ease with which a system or component can be transferred from one hardware or software environment to another" [10]. Portability is even more necessary nowadays as computing resources are widely distributed along edge, fog and cloud infrastructures. Scalability refers not only to the ability of the system to accommodate and handle an increasing amount of workload [8] but also to perform efficiently under these conditions [13]. Multiagent systems have been instrumental to address these requirements in several industrial contexts [14] and we review some of these solutions.

ADACOR[15] is an agile and adaptive architecture that enables fast reaction to disturbances in the shop floor. In this architecture agents are used to materialise a set of cooperative and autonomous holons, where a supervisor holon is able to coordinate and make decisions with a global perspective. A prototype of ADACOR was built using JADE as it provided compliance with FIPA agent standards. SOHCA[16] is a service-oriented and holonic control architecture for reconfiguration of disperse manufacturing systems presented. In their paper, authors propose a method to design such reconfigurable systems using Petri Nets and enabling integration between factory control and shop floor control layers.

WANTS[17] introduces a workflow and agents approach for managing large-scale network and service management in the context of Telecom Italia. In their work, the authors report the use of WANTS for access network and part of the metropolitan aggregation network, not for the national backbone. WANTS is developed on top of WADE, a Workflow extension of JADE, that gives JADE agents the ability to execute workflows. WANTS architecture incorporates Agent Applications (AA) that are developed as workflow engines. They report their approach is flexible to changes in devices and in the topology of the network. Authors do not mention portability of their solution across infrastructure. The authors of [18] present a hierarchical agent architecture built with JADE and OSGI.

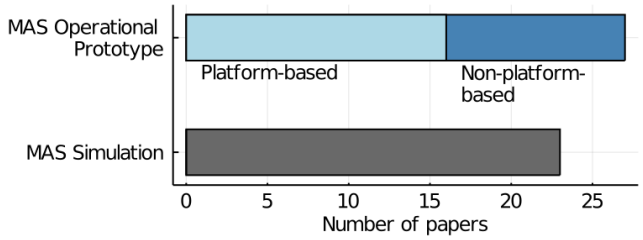


**Fig. 1:** Implementation approaches in the top-50 most cited papers related to multiagent industrial control.

### B. MULTIAGENT-BASED SYSTEM DEVELOPMENT

The development of agent-based control systems relies in Multiagent system (MAS) platforms. Fig. 1 shows a summary of the implementation approaches in the 50 most cited[1] journal and conference papers, related to "multi agent industrial control" that were published in the last decade and that reported some implementation, i.e. excluding reviews and purely theoretical contributions. MAS platforms have been used in 16 out of 27 papers with an operational system prototype, with JADE[19] used or extended in 13, JaCaMo[20] in 2 and Gorite[21] in 1. This sample gives an indication of the importance of MAS platforms in the agent-based control development. However, the reader is directed to [22], [23] for more comprehensive reviews of existing platforms.

Such platforms are technological architectures that provide the environment in which agents operate [24]. They ease the engineering of the control system and facilitate interoperability among standard-compliant platforms [25]. For example, platforms based on the FIPA[2] standard incorporate agent and service directory services where agents register and look for others or the services they offer [26]. However, such a platform-dependent approach has the following implications to the industrial control systems:

- *Platform Localisation*. The assumption that (FIPA-compliant) platforms are controlled environments with central directory services [26] might affect system scalability and robustness [27] as it incorporates a single point of failure to the system [28]. Discovery of other agents and their services is, by design, dependant on a directory, ruling out other approaches such as a decentralised service discovery [29]. This assumption also implies that platforms provide global visibility of the entire MAS. Not only maintaining the global view of the MAS can be computationally demanding as the system grows, but also there are cases where a global view is not necessary to achieve global system goals as shown in several applications within the field of swarm engineering [30].
- *Platform Extendability Constraints*. The platform services are deployed as one or several software applications with agents embedded within the platform [28]. We highlight two implications of this approach: on the one side, it couples the agents to a particular platform, usually one platform cannot accommodate agents developed in other platform [24]. Furthermore, interoperation between multiple platforms is also limited and only recently, progress has been made in defining environments that enable such interactions [31]. On the other side, this approach implies that agents might share resources and then interfere with each other when they are deployed as a single application. Moreover, the ability to monitor individual agents performance and their resource usage varies widely from platform to platform. Monitoring individual agent's resource usage is key to scale the system and make decisions about where to deploy each agent according to their resource demands.
- *Platform Deployment Limitations*. Despite substantial progress enabled by the existing platforms, it is still an open challenge to facilitate the distribution of intelligence across the available infrastructure i.e. edge, fog and cloud [3]. In addition, the amount of manual setup required [27] slows down the processes of porting agents to different computing environments.

Overall, an agent platform eases the implementation of the agent-based control systems, but it also constraints the way that the solutions are built. As a result, the control system ability to address requirements of flexibility, portability and scalability is limited by the platform's capabilities. The efforts to mitigate these limitations can take two directions. On the one side, evolving and extending the current platforms, e.g. by enabling deploying of agents developed in other languages and platforms, as suggested by [4]. Alternatively, the dependency on MAS platforms can be reduced, for example, by bringing, to the control system engineering, other distributed systems development approaches and technologies that have been less exploited in this field.

### C. DISTRIBUTED SYSTEM ENGINEERING TOOLS

This paper presents alternatives to platform-based agent development that can facilitate the development of ad hoc agent-based control systems. To support this, here we review some system engineering tools that have the potential to complement MAS-based industrial control engineering.

#### 1) Microservices

Authors of [32] define *microservices* as decoupled and autonomous software units with a specific functionality in a bounded context. For [33], microservices is an specific approach to implement *Service Oriented Architectures (SOA)* where each microservice is a small autonomous service within a business boundary. Microservices is also regarded as an approach to build distributed systems from "small applications with a single responsibility and that scale up and are deployed independently" [34].

In [6], Richards highlights that microservices is an architectural pattern in which, regardless of the implementation

[1] According to Semantics Scholar on 17/12/2020.
[2] Foundation for Intelligent Physical Agents

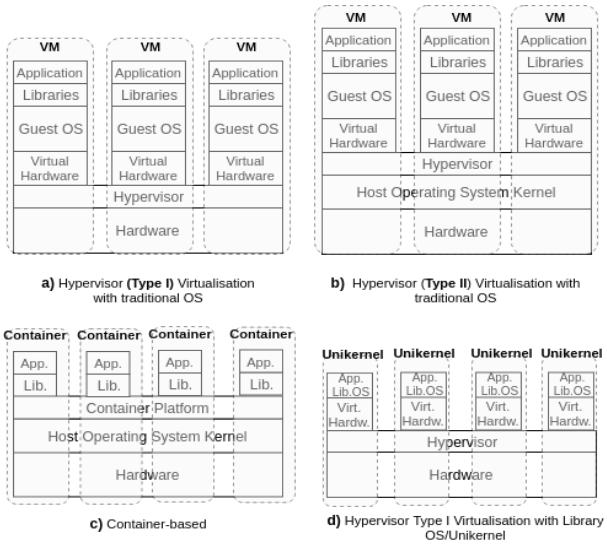


**Fig. 2:** Virtualisation architectures.

topology, a central notion of the pattern is having "separately deployed units" which increase scalability and decoupling of components. Microservices also brings the advantage of real-time production deployments, where the services component can be easily and quickly swapped in real-time. According to [35], the distinctive features of microservices are encapsulation, modularity, distributed composition and network-accessibility.

Applications of microservices in industrial context have shown their benefits. Authors of [36] present their ongoing efforts for transformation of the architecture of a distributed automation system to use microservices. [37] propose to build MAS based on microservices hence offering scalability, flexibility and loosely coupling to Internet of Things (IoT) applications. [38] introduced an edge-based microservices architecture for IoT. They developed a mobility analysis services case with extendable microservices and deployed their system in low-capacity edge servers. Their results confirmed the low latency of edge-computing solutions. [39] introduced the "Multi-Agent Microservices" (MAMS) approach, where *interface agents* expose their inbox –and potentially other aspects of their internal state— as web resources, so other agents use these resources to interact.

### 2) Virtualisation & System Containers

Virtualisation technologies enable decoupling of the hardware and the software running on it [40]. This decoupling facilitates isolation of shared resources and capabilities as well as runtime reconfiguration. As illustrated in Fig. 2 (adapted from [40]) the virtualisation can be achieved through a *hypervisor* or through *system containers* [40]. *Hypervisors* enable the creation of *Virtual Machines* (VMs) with a set of allocated computing resources e.g. CPU, RAM, network and file system. Depending on how the virtualisation is achieved, the *hypervisor* can be of *type I*, if it enables the creation of VMs on top of the bare hardware; or of *type II*, if it enables creation of VMs on top of a host operating system.

Whereas VMs require a full guest operating system, containers offer a more lightweight approach, leveraging on the host operating system kernel and implementing isolation of processes [41], as well as, isolation of context and computing resources [42]. As illustrated in Fig. 3, the containers are built from an image script that links the required operating system libraries and the software dependencies, together with the application source code.

The joint use of microservices and containers brings benefits in code portability, life cycle management and resource utilisation [43]. With this approach, code portability is made simple. A single image works as the template to build and run one or many containers across different computing infrastructures with little configuration required and no additional development. The image-based approach enables automation of the image workflow by re-compiling, testing, deploying and running new containers once the source code is updated [44]. A container manager system –a.k.a engine–, is able to store multiple images and create containers, on demand, from the image built from the scripts as shown in Fig. 3. Since the containers require a lower overhead than a VM, these can be used to implement individual independent applications or microservices. Furthermore, library operating systems, such as *Unikernels*, enable the creation of guest operating systems based on a specialised kernel i.e., tailored to a single-purpose application, making these systems more secure and resource-efficient than conventional virtualised equivalents [45].

Note that these *system containers*, are different to the *application containers* that have been implemented for some MAS platforms, for example JADE. There are multiple differences between these implementations of the container pattern, perhaps the most relevant are that system containers provide isolation of resources while enable the creation of fully specialised software environment from the operating system [40]. On the other side, the application containers are MAS-platform specific and depend on the programming language runtime the platform is implemented e.g. Java Virtual Machine [25].

In the industrial context, system containers have been barely discussed. [46] proposed an architecture for a multi-purpose industrial controller that supports multiple PLC execution engines while addressing flexible function deployment. [47] explored the use of containers in real-time applications to migrate from bare metal hardware to cloud platforms, with comparable latency performance.

## III. MAS DEVELOPMENT WITH REDUCED PLATFORM DEPENDENCIES

Given the issues identified in the previous section with platform-centred approach, it is desirable that industrial agent-based control systems consider these requirements: 1) to avoid by-default single-point of failures, 2) to avoid agent interference by enabling resource isolation and 3) to ease agent portability across the available infrastructure. In this

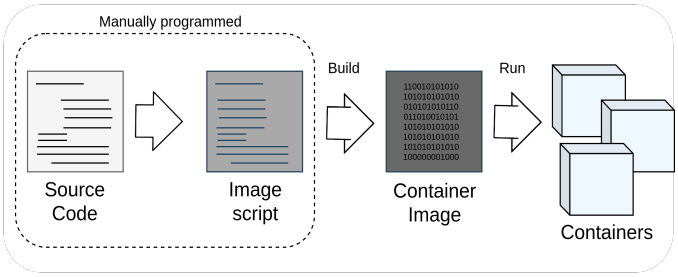


**Fig. 3:** Container generation

section, we discuss how these requirements can be tackled with tools described in II-C.

### A. CONTAINER-VIRTUALISATION FOR MAS DEVELOPMENT

The approach being used in this paper has its origins in the telecom network engineering. Traditional telecommunication networks were built from devices that performed different network functions, e.g. routing or switching. Only recently these functions were decoupled from the network devices in an approach called *Network Function Virtualisation* (NFV) that enables the deployment of functions across distributed computing infrastructure as well as the selection and composition of them for the delivery of the telecommunication services [2].

Inspired by these decoupled functions, agents of an agent-based control system can also be decoupled from the devices they are running on, and especially from a single MAS platform. In consequence, the notion of an agent as a self-contained design entity, goes beyond a design concept to become also a self-contained deployment unit. Compared to the NFV approach, there are two key differences of these agents. First, these agents perform wider control functions that can be extended beyond the telecommunications domain, for example, these agents can perform *production control* [48], including monitoring and scheduling. Second, the agents are active and not passive services that need to be orchestrated as is the case of functions in NFV.

Such agents are containerised software units that carry out industrial control functions. They are implemented as lightweight and self-contained applications with a narrow-bounded responsibility. This enable these agents to be deployed, scaled, tested, evolved, started and stopped independently of other agents and its environment. As the agents are deployed in system containers they can be implemented in different programming languages, runtime platforms and environments. Likewise, as the resulting control system is built from a collection of independent agents, the dependencies to a MAS platform are relaxed.

We now highlight the key properties of this approach and discuss the differences with conventional approach that help to address the shortcomings identified in section II-B.

### B. REMOVING THE NEED FOR A GLOBAL DIRECTORY OF AGENTS AND SERVICES

System containers have their own identification (e.g. IP addresses) for each network they are connected to. Hence, a one-to-one mapping between the container and the agent, removes the need for extra agent IDs. This mapping simplifies discovery of agents and still multiple agents can run in the same machine given they are deployed in different system containers and also multiple overlay networks can be defined. Agents use their network address to locate each other, every agent has a local directory with the addresses of the agents of interest. The system designer determines the constraints on the size of the local directories and the protocols to maintain them. This approach brings flexibility to the control system as decentralised service discovery can be used to locate agents and services. Since there is no global system view by default, the decision of the type of view to maintain is up to the system designer and can vary depending on the number of agents of the control system and the resources available for each one. For example, a design can include distributed registries in each agent that keep references of other agents depending on their needs and their availability, similar to what is done in ad hoc network protocols [49].

### C. DECOUPLING AN AGENT-BASED CONTROL SYSTEM FROM MAS PLATFORM

Each agent of the control system runs an isolated environment provided by the system container. This means the agent is deployed with its own operating system and the software libraries required, according to its functionality. The agent is also able to distribute its functionality along multiple *threads* of execution. Agents in a control system are generally heterogeneous in various aspects. For example, agents representing resources of a manufacturing shop floor have responsibility to continuously monitor and adjust configurations of physical properties of the resource (e.g. a milling machine), and hence these agents can have a long-lasting lifespan while the machine is in operation. On the other side, for example, agents representing orders, might be transient, performing simpler functions within minutes before expiring. Although any of these agents can be implemented with system containers, the container image characteristics and software infrastructure required in each case might be completely different. As the number of simultaneous orders, although transient, might be generally higher than the number of resources, the system containers for implementing these "order" agents have stricter resource constraints. These constraints can be met by implementing these agents in a different programming language and base operating systems than the ones used for the "resource" agents. As indicated in section II-C2, existing technologies enable deployment of highly bespoke operating systems with minimal footprint and overhead. The whole control system is not embedded in a MAS platform but each containerised agent has its own software infrastructure that can evolve independently of the system. In addition, resource usage by each agent can be monitored uniformly at system

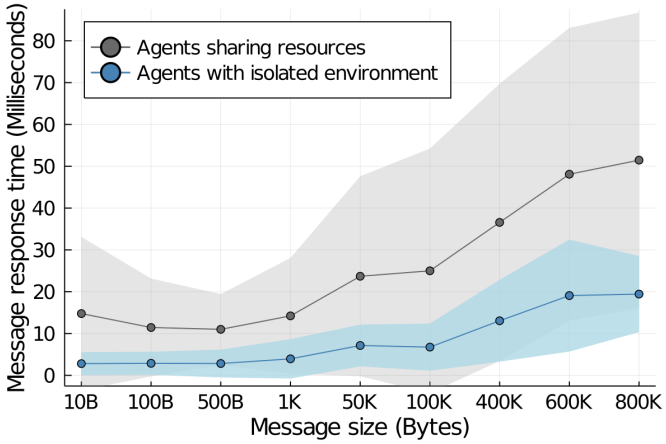


**Fig. 4:** Comparison of agents response time when sharing and not sharing resources for different message sizes, $\bar{x}$ and $s$ for 8 runs per each message size.

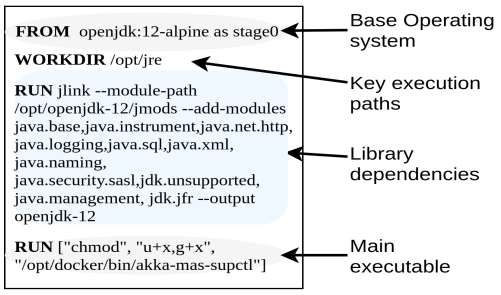


**Fig. 5:** Excerpt of container image showing specific runtime version and modules loaded for a particular agent application.

container level rather than depending on platform-specific services.

To show the effect of sharing resources in the response time, we implemented a simple MAS where a sender agent periodically sent messages, of a given size, to a receiver agent that acknowledges reception. Using container-virtualisation, additional memory-intensive agents were deployed together with the receiver agent, in the first case within the same container and in the second case, in separate containers. As shown in Fig. 4, when agents share resources, e.g. RAM, the performance of each agent is affected when other agents are using resources intensively. In the case illustrated, the mean response time for 576 messages (8 messages of each size per run) of different sizes, is increased when memory-intensive agents are sharing resources with the recipient agent. More important, the standard deviation, represented by the shadow area, is high. As system containers enable isolation of the agents environment, the overall communication time is less sensible to other agents running in the same machine as long as they are all running in isolated environments.

### D. REDUCED PORTABILITY EFFORT

The incorporation of system containers in the control system engineering enables the integration of software development and deployment operations in a infrastructure-as-a-code approach [43]. This approach brings, to the engineering process, practices such as continuous integration, continuous testing and integrated system change management, among others [50] that also boost the system readiness for automation of software evolution and operation activities, including portability. The agent's software and library dependencies are defined in a script (see Fig. 5) that is run once to create the base image of the control agents, then multiple instances, with potentially different configurations, can be started on-demand across the available infrastructure. As agents have their own isolated environment, the version conflicts on host platforms are limited and even completely avoided. Overall the portability is simplified, a base image is built once and containers can be installed many times in multiple target host where the container management system has been previously installed.

In addition, the life cycle of the agents is carried out through a container management infrastructure which enables the agents to be be started and shutdown, quickly on-demand, without affecting others. Therefore, these containerised agents can be quickly scaled up or out (of the initial runtime environment) and replaced with no required changes to the implementation.

### E. DESIGN CONSIDERATIONS

In this section we present a number of considerations for designing an agent-based control following the approach proposed.

1) *Assessment of the agent implementation strategy*. Although it seems obvious, conventional approaches assume the control systems is built on top of a MAS platform, then the first decision is which platform to use. That leads to a selection of programming language and concrete design approach. [51] points out that agent platforms do not meet specific needs of industrial applications, given their general-purpose orientation. On the contrary, a MAS platform-independent approach enables the design of highly bespoke control systems. When agents are mapped one-to-one to the containers, some of the MAS platforms services, such as the life cycle management, can be replaced by container management services. Other services e.g. agent discovery, are designed according to the concrete case. However,the downside of this approach, compared to the platform-dependent solutions, is that the design effort is increased according to the complexity of the use cases.
2) *Individual and shared repositories*. As an agent is intended to encapsulate its own state, its design should link state repositories and knowledge bases to each individual agent. Knowledge sharing among agents is preferred via message exchange rather than using

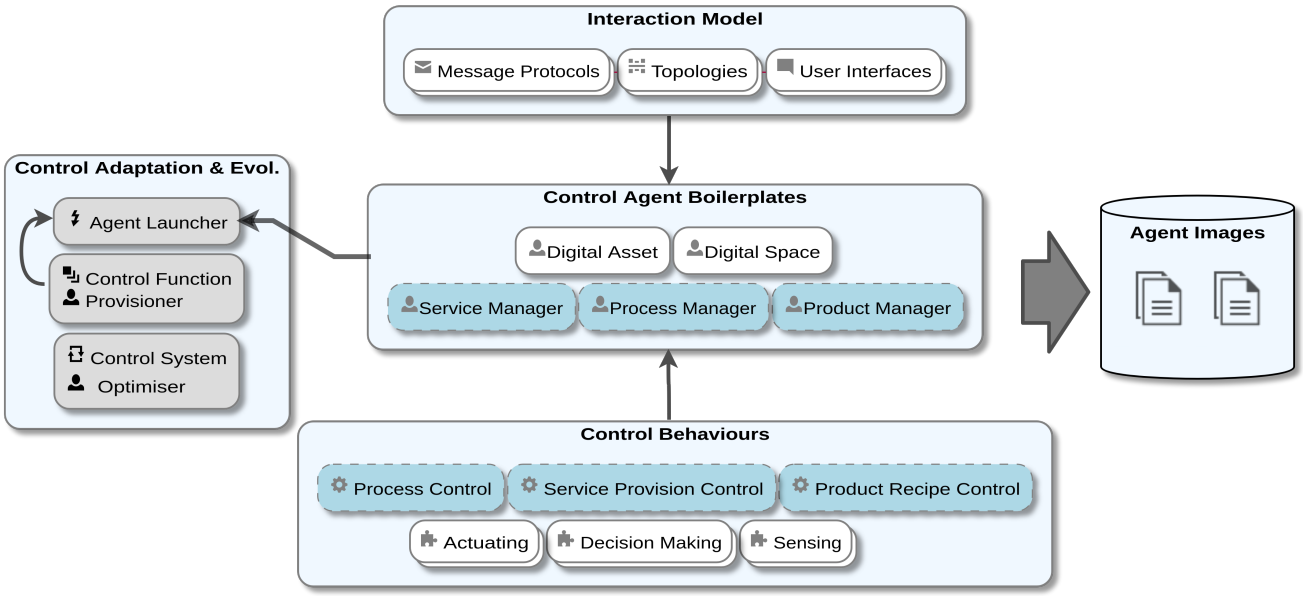


**Fig. 6:** Control System Reference Architecture

blackboard-type spaces that introduce a dependency among the agent and a single-point-of-failure entity or service. Agents might share their own state data (e.g. observations from the elements under control) or configurations that other agents can use to their own states. This has to be defined according to requirements of each case, e.g. privacy and communication overhead. Shared repositories, when required, are not part of the core agent behaviours but are used for supporting activities e.g. backup, archives, etc.

3) *System-Wide Visibility Strategy*. As a basic principle, agents generally have a partial/local view of the environment, however, for implementation purposes, this principle is sometimes relaxed. A global view of the environment is possible by either adding shared spaces and services or enabling connections among all the agents of the system. However, this strategy becomes harder to maintain while the system grows. On the other side, when agents have partial views of the environment, the agents are less sensitive to changes in the scale. The trade-off is that the system becomes more complex as it requires a strategy to make decisions in a rationally-bounded context and deal with inconsistent or approximated views of the environment.
4) *Agent behaviours and deployment* To facilitate the customisation of a deployment environment; the design of the underlying agent behaviours should decouple environment-specific operations from the common operations given by the agent type. For example, following the "separation of concerns" principle [52], the behaviour to trigger a temperature adjustment should separate the rules that lead to specific levels of temperature and the process to trigger the changes, from the actual action that depends on the available physical interfaces. This way, multiple functionally-equivalent agent instances can be parametrised to run with easy portability along different hardware infrastructures. This approach fosters quick reusability of pre-designed elements regardless of the concrete deployment environment, which reduces the extra design effort required when there is no MAS platform support. Evolution of core behaviour is also possible with reduced impact on physical interfaces.

## IV. A REFERENCE ARCHITECTURE FOR CONTAINERISED CONTROL AGENTS

This section presents a control architecture based on the approach and considerations previously described.

### A. SYSTEM VIEWPOINT

The Fig. 6 illustrates the system architecture that is being proposed in this paper. The key design principle is that the control system is built from independent control agents, composed of both, behaviours implementing the agent's interaction model, and atomic control behaviours, specific to the entities or processes of the industrial system under control. Multiple agent "boilerplates" are made out of these behaviours and each of these boilerplates is defined as a self-contained agent image that is deployed on-demand.

The agent boilerplates and their control behaviours cover the different perspectives of interest of the industrial system. The agents monitor and control the condition and performance from each perspective. For example, *Digital Asset* agents are linked to system elements such as machines,

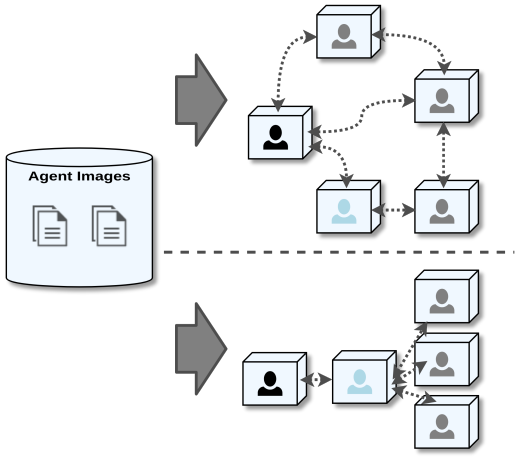


**Fig. 7:** Multiple realisations of the control reference architecture

telecommunication devices or any other assets or their components. The *Digital Space* agents are mapped to the spaces where the assets are located. The granularity of the association between the agent and the assets or spaces depends on the availability of the sensors and actuators. These agents incorporate ad hoc behaviours for sensing and controlling that interface with the available physical sensors and capabilities of the asset or space. Likewise, other agents addressing more functional perspectives include: the *Process Manager* agents controlling one or several processes and workflows of interest that are carried out to deliver services or manufacture products. The *Product Manager* agents controlling the manufacture of one or several individual products (type of products) of interest. The *Service Manager* agents control the provision of one or several individual services (type of services) of interest. The control behaviours enable collection of measurement data for creating views of the industrial system with the metrics of interest from each perspective. These views then enable monitoring and triggering of the corresponding control actions. The individual architecture of a control agent is discussed in section IV-B.

The system architecture does not include fundamental structural relationships between agents. These are replaced by communication and coordination flows that are defined by the agent topologies and corresponding message protocols. Hence, as depicted in Fig. 7, the realisations of this architecture can implement different interaction models while reusing as much as possible the control behaviours and the specific interfaces with the elements under control. Moreover, different interaction models can work for different situations thus providing flexibility to the overall system.

Support functions related to the control system's infrastructure can be either added to the core control agents or composed in dedicated infrastructure agents. These functions enable configuration, agent's life cycle management and evolution of the overall system. The agent launcher configures agent's behaviours to a particular context e.g. hardware or network infrastructure and triggers the initialisation of an individual agent. The *Control Function Provisioner* agents trigger the creation of other control agents, they monitor their state via interaction model and support agent takeovers when required as they have behaviours that interface with the Container Management System. The *Control System Optimisers* agents monitor the overall control system performance, based on configured metrics of interest e.g. resource usage, communication latency, messages passed, etc; and trigger routines for optimisation of the control agents or their interaction models. This set of agents are then the main source of reconfiguration and adaptation of the control system.

### B. INTERNAL AGENT VIEWPOINT

The agent is a modular unit that is assembled on deployment given the configuration passed. This configuration defines specific parameters of operation for the behaviours and interaction model of the agent. As deployment is quick, any substantial change in the agent structure or interaction model leads to the agent being decommissioned and a new instance with updated structure and behaviours is deployed. Fig. 8 shows the simplified view of the internal architecture of an agent representing an asset of an industrial system. The key fundamental agent functions are covered by ad hoc behaviours organised in four groups: observation, communication, control and decision making. Given the configuration defined in the agent launcher, the specific behaviours of the ones presented in Fig. 8 are deployed. An agent can be deployed with all or a subset of the available behaviours. The sensing behaviour enables processing of the data streams coming from the configured data sources, including the implementation of the interfaces with physical sensors deployed along an industrial system. Supporting actors preprocess (e.g. normalise or classify) the data collected and manage the life cycle of the agent's view of the industrial system. Depending on the type of agent, this view can be local or according to interaction model, consider other agent's collected data. The decision-making behaviours drive the control action selection by defining the model and criteria used for this selection. This selection might be as simple or complex as required, for example based on rule-based, probabilistic, neural networks or any other learning models. The actuating behaviours link the available physical and software-based capabilities (actuators), thus enabling the activation of control actions and provide a message-driven interface to these actions. The behaviours can be aggregated or activated in runtime where they remain stable, however if more significant changes are required, for example, upgrade a pre-trained model with new meta-parameters and configuration, a take-over process takes place where the containerised agent is replaced by another one with the updated behaviours and model.

The agent's state is stored in a local repository, however backup functions are intended to be defined as a specialised actuating behaviour copying local state to decoupled repositories. This way the operation of the agent is not dependent

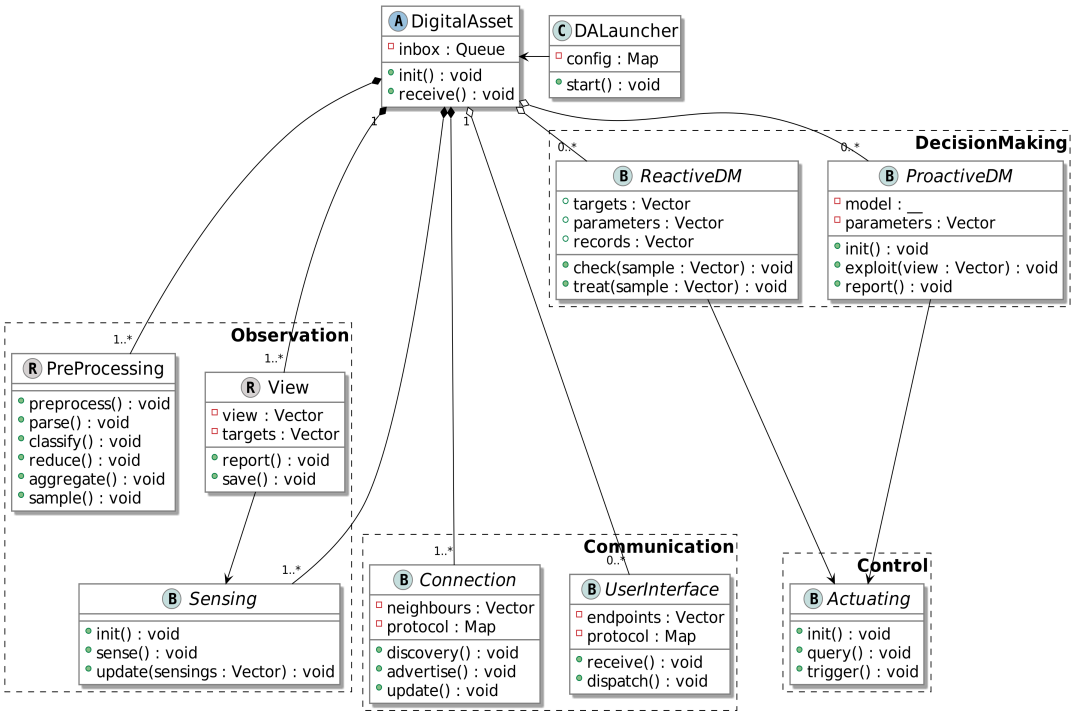


**Fig. 8:** Containerised Agent: Simplified Structural View Of The Internal Architecture. Stereotypes as follows: A) Agent, B) Behaviour, R) Supporting Actor, C) Auxiliary Class.

on remote repositories but it is also possible to instantiate a new or changed version of the agent with a baseline of historical state, for example in the case of an update of the agent's decision models.

## V. CASE STUDY: SUPERVISORY CONTROL OF NEXT GENERATION NETWORK INFRASTRUCTURE

This section introduces a case study of MAS-based supervisory control applied to next generation telecommunications network infrastructures. The study is intended to illustrate the characteristics of an agent-based control approach that relaxes the MAS platform dependencies as discussed in sections III and IV.

### A. CASE STUDY DESCRIPTION

#### 1) Next Generation Digital Infrastructures

Fig. 9 illustrates an example of a next generation digital infrastructure where multiple networks, not only core telecommunications but also wireless sensor and actuator, virtual or even workforce networks, enable the provision of a variety customisable and heterogeneous digital services with some agreed service levels (SLAs). Beyond physical and software-defined telecom networks (SDN) [53] and the convergence of mobile and fixed access networks [54], other networked systems such as the workforce or the supply management systems are also key components of the infrastructure to enable seamlessly and agile service delivery. On the right side of Fig. 9, supervising this infrastructure, an instance of an agent-based control system enables decision-making considering the data gathered at each different network context.

The main type of service of interest is the data transport across different locations of the infrastructure network. Instances of this service define specific requirements of bandwidth and latency, among other properties. The data is transported along multiple sub-networks, some of them with different operators and generally different low-level control entities, such as software defined controllers. Given the huge amounts of data generated by these systems, ensuring that service provision meets customer expectations (e.g. as set in the SLAs) and business goals while considering the operational context, becomes difficult to achieve without automation [55]. A supervisory control –above other existing local network control functions– is required to integrate the multiple perspectives and consolidate control actions given this integrated view.

#### 2) Requirements of Next Generation Digital Infrastructures

Together with academic and industrial partners of the NG-CDI[3] research programme we identified requirements of the future digital infrastructures. We highlight here those key requirements that have implication on the control system and that make the approach discussed in section III particularly useful in this case.

- *Ever growing infrastructure and third-party platform risk aversion.* Telecom network infrastructures are in-

[3]https://www.ng-cdi.org/

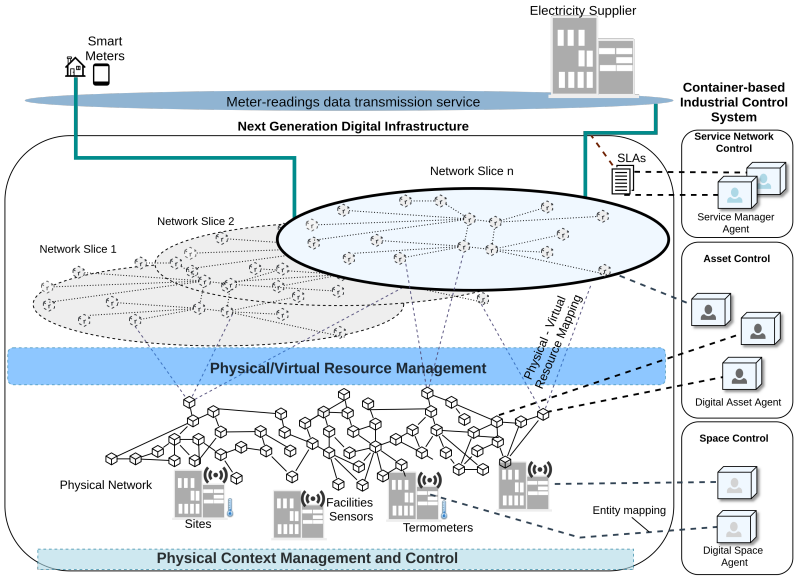


**Fig. 9:** Case Study Environment: Digital Infrastructure and Container-based Control System

herently distributed and in permanent evolution [56], [57]. It is hard to anticipate the dimensions that the infrastructure might reach. For this reason, it is risky and hence undesirable to tie key supervisory control functions to a single third-party MAS platform because there is no certainty about the platform's ability to evolve at the speed and dimensions the infrastructure and control system might require.

- *Geographically distribution and effortless portability.* Network infrastructures have been and will continue to be geographically distributed. Moreover the computing resources that enable their control are also distributed. It is desirable that control system components can be deployed and ported along the cloud or edge computing resources with minimal effort and according to where it is more advantageous from a business or end-user perspective.
- *Elasticity and fine-grained resource monitoring and allocation.* The network infrastructures are expected to be elastic, leveraging on software-defined networks that can grow or shrink on demand [58]. Likewise, it should be possible to scale up or down the control system, according to the network infrastructure needs. Part of this is assumed for agent-based control systems, as agents can be started or shut down on demand. However, determining the best allocation of resources to agents is only possible by individually monitoring agent's resource needs and usage. This way, extra resources can be provisioned for the control system when required and also freed when unused.

### B. INTRODUCTION TO CONTROL SOLUTION APPROACH

A realisation of the reference architecture presented in the section IV was used to implement a control system prototype for supervision of the operation of an instance of a digital infrastructure. The supervisory control is the topmost level control function with two main monitoring and control perspectives: a) monitoring and managing the physical condition of the assets that are involved in the delivery of the digital services and b) assessing and adjusting the service performance with reference to the SLAs. Digital Asset agents are mapped one-to-one to the network elements and monitor their local performance: in/out throughput given the existing links and connections of the network element with others. These agents also monitor the physical condition of the assets (namely board temperature, system errors packets, CPU usage and RAM usage, among others). Control actions, such as re-routing of traffic flow among different paths or provision of new network resources, are triggered via REST API to low-level controllers (e.g. SDN Controller and Container Manager).

The Service Provision Control behaviours include algorithms for both monitoring the performance of a set of services and triggering the actions to ensure the agreed performance. Algorithm 1 shows the routines for calculating network throughput variation $\Delta T$ after the Digital Assets have reported their state. The algorithm consolidates measurements from Digital Assets involved in the service delivery and compares their performance with each other. If the variation $\Delta_i$ for a given time window $(w_s, w_e)$ is higher than the agreed SLA thresholds it triggers the defined action, in this case the replacement of the under performing Digital Asset. This algorithm shows a simplified version, but different actions can be triggered by matching the observed

values.

This control behaviour that can be either implemented by a dedicated Service Manager agent, or a selected Digital Asset agent. In fact, three variations of the control architecture were implemented. The first one, named $H$ (Hierarchical), defines a hierarchy between the Digital Asset agents (DA) and a Service Manager (SM). The other two architectures, are connected resembling the topology $t$ of the network under control. In the architecture named $L_t$ (Leader with topology $t$), one of the DAs is the leader that takes over the Service Provision Control behaviour instead of a dedicated SM. In the architecture named $D_t$ (Distributed with topology $t$), the leader responsibilities are distributed locally among DAs that exchange data collected with neighbours.

The prototype[4] of the control system was implemented in Scala using the akka actor framework[5] and the agents are packed as docker[6] containers.

**Algorithm 1** Monitoring of Throughput Variation ($\Delta T$)

**input:** $w_s, w_e, V, \alpha, p$

**for** $l_i$ *in* $L$ **do**
  $A \leftarrow a(V, l_i, w_s, w_e)$
  $X_i \leftarrow m(A)$
  $G \leftarrow \emptyset, K \leftarrow \emptyset,$
  $B \leftarrow n(V, w_s, w_e)$
  **for** $B_i$ *in* $B$ **do**
    $C \leftarrow c(V, B_i, w_s, w_e)$
    $C' \leftarrow c(V, B'_i, w_s, w_e)$
    $\Delta_i = m(C') - m(C)$
    **if** $\Delta_i > \alpha$ **then**
      $r(p, B_i)$
      $K \leftarrow [K, B_i]$
  $w_s \leftarrow w_e$

### C. EVALUATION OF THE EFFECTIVENESS OF THE CONTAINER-BASED MAS APPROACH TO SUPERVISORY CONTROL

The evaluation of the proposed approach is broken down in two parts. The first part assesses the prototype in operation within a realistic but small scale environment. Because of complexity and availability of resources this is only possible to analyse at small scale. The second part aims to analyse behaviours of the system at larger levels of scale, especially with regards to control agent's resource management (which is linked directly to the ability to ease portability and elasticity). In both parts of the evaluation, the statistical significance of differences among architectures is checked according to these hypothesis: $H_0$: The mean usage of resources is the same among architectures. $H_A$: The mean usage of resources is different among architectures.

[4]Github repository available upon request.
[5]https://developer.lightbend.com/guides/akka-quickstart-scala/
[6]https://www.docker.com/resources/what-container

**TABLE 1**
Symbols used in the Algorithm 1

| Symbol | Description |
|---|---|
| $w_s$ | Time Window Start (Last Time Window End) |
| $w_e$ | Time Window End (Current time) |
| $V$ | Set of measurements reported by Digital Assets |
| $p$ | Function Provisioner Address |
| $m(A)$ | Function that calculates the mean of the given set. |
| $L$ | Set of Service Level Agreements Metrics |
| $a(V, l_i, w_s, w_e)$ | Function that filters measurements reported for metric $l_i$ during interval: $w_s - w_e$ |
| $n(V, w_s, w_e)$ | Function that returns set of Digital Assets that reported measurements during interval. |
| $c(V, B_i, w_s, w_e)$ | Function that filters measurements by the given set of Digital Assets during interval. |
| $B_i$ | Subset containing the Digital Asset $b_i$ |
| $B'_i$ | Subset of Digital Assets not containing $b_i$ |
| $\alpha$ | Parameter that indicates the tolerance threshold for triggering replacement |
| $r(p, B_i)$ | Function that asks function provisioner $p$ to replace set of Digital Assets $B_i$ |

#### 1) Small Scale Evaluation

A network with the topology presented in Fig. 10 was emulated to serve as testbed for the control prototype. This particular topology was chosen as it reproduces a typical topology that appears in this type of network infrastructures [59]. This environment was implemented in mininet[7] with Ryu SDN Controller[8]. Each node corresponds to an emulated switch, the links between them were set with different bandwidth and delays. Ryu performs basic data flow control capturing traffic measurements through the emulated network and making them available via REST API. Over the emulated network, a regular flow of data traffic is simulated using *iperf* tool. The condition of the underlying assets over which each switch is deployed is simulated by a SimAsset. We built this as a Scala actor that exposes a data stream from synthetic CSV files, randomly generated from datasets provided by industrial partners. There is one SimAsset per switch and the data is sensed by a matching Digital Asset agent via one of their sensing behaviours. On deployment of the control system, one agent is created and associated to each switch. This way the supervisory control system observes traffic and network state via Ryu SDN Controller and physical condition via the SimAsset.

The control system prototype was run eight times for each variation of the architecture while completing the following scenarios:

- *Standard Operation:* The Ryu SDN Controller drives traffic considering a combination of shortest path and highest bandwidth. Supervisory control monitors, records and compares throughput against SLAs. Particularly, in this scenario the control system supervises that connectivity and data traffic between hosts connected to switches 8 and 20 (Fig. 10) is within parameters defined in the SLAs.

[7]http://mininet.org/
[8]https://ryu-sdn.org/

- *Condition-driven Operation:* The node 7 in Fig. 10, is simulated to fail due to overheating. The Supervisory control detects this condition degradation and triggers update of data flows to drain the traffic out of switch 7.

### 2) Large Scale Evaluation

Agents can be deployed across large infrastructures as required, only if there are enough resources available. Hence, monitoring the resources each agent uses is critical to determine their demands. Likewise, the control system's ability to operate elastically, is only possible if resources used by agent's can be provisioned and freed on-demand. The aim of this part of the evaluation is to analyse whether the approach explored facilitates overall management of control resources at a large scale. The following experiments were carried out:

- *Portability & Resource Monitoring* The aim was to deploy the control system prototype with hundreds of agents and monitor their resources. This enables to evaluate how easy it is to port and scale the system and monitor each agent's resources. To simplify the experiment we use a ring topology for the emulated network and focus on architectures $H$ (Hierarchical) and $D_r$ (Distributed with a ring topology). The control system was ported from a laptop (Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz with 16Gb of RAM ) to a server (Intel(R) Xeon(R) CPU E5-2680 v4 @ 2.40GHz with 66 Gb of RAM). Resource monitoring was implemented using Prometheus[9] and Kamon[10], reporting to tailored Grafana dashboards[11].
- *Deployment Strategy* As system containers enable different strategies for deployment of heterogeneous agents, this experiment aims to analyse the effect of the strategy when the number of agents increases. The heterogeneity here is evidenced by the resources (RAM) required by each agent and the effect is quantified in terms of the cost of running the control system which is directly proportional to the resources used. Two strategies are studied: one with mixed agent types in a single deployment unit (1-to-Many), which is normally the case in conventional solutions, and the other one with one deployment unit per agent (1-to-1), which is the strategy used in the prototype implementation. Control systems with three proportions of stable/transient agents were simulated: 20/80, 50/50 and 80/20. The agents whose resource demands are stable, are those that control, for example, physical assets that are up and running during most of the time; and agents that are transient, control, for example, temporal entities or assets. In the context of digital infrastructures, these can correspond to network elements of temporal networks set up on demand for music or sports events. Resource demands for each agent are modelled as Poisson processes, where an event indicates a variation in the resource consumption pattern. Details of the simulation parameters are presented in Table 2 and the results are discussed in the next section.

[9]https://prometheus.io/
[10]https://kamon.io/
[11]https://grafana.com/

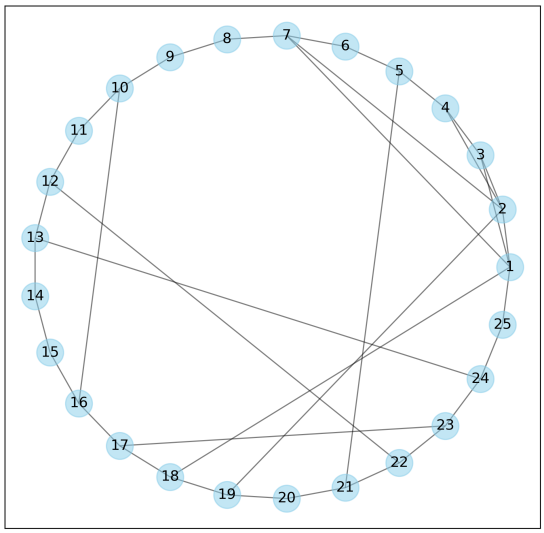


**Fig. 10:** Network used in the emulation. Generated using the Newman-Watts-Strogatz model [60] with $n = 25$, $k = 3$ and $p = 0.4$.

**TABLE 2**
On-Demand Resource Simulation Parameters

| Parameter Name | Value |
|---|---|
| Quantity of agents | [5, 10, 50, 80, 120, 400] |
| Proportion of stable agents | [0.2, 0.5, 0.8] |
| Transient agents operation (time units) | [20, 40, 60] |
| Resource usage profiles: | Stable: Max 0.9<br>Transient: Max 0.5 |
| Agents demand change rate | Poisson process with $\lambda = 0.2$ |
| Times steps | 1000 |
| Repetitions | 8 |

## D. RESULTS & DISCUSSION

### 1) Small Scale

The supervisory control collected and aggregated throughput data, enabling the monitoring of the network services via dashboards as the one presented in Fig. 11. It highlights the aggregated service view created out of the measurements collected by individual Digital Asset (DA) agents. In the standard operation scenario, all three variations of the architecture were effective in enabling traffic

**TABLE 3**
Small Scale: Agent's Summary Statistics

| | RAM(MB) | | | CPU (%) | | | Tx (MB) | | |
|---|---|---|---|---|---|---|---|---|---|
| Topo. | $\bar{x}$ | $s$ | $p$ | $\bar{x}$ | $s$ | $p$ | $\bar{x}$ | $s$ | $p$ |
| $H$ | 136.4156 | 7.0973 | 0.006 | 13.58 | 2.04 | 0.009 | 0.0651 | 0.0203 | 0.008 |
| $L_{sw}$ | 150.5052 | 15.4495 | 0.002 | 16.35 | 3.42 | 0.005 | 0.0979 | 0.0366 | 0.002 |
| $D_{sw}$ | 147.2075 | 15.7945 | 0.003 | 15.45 | 4.03 | 0.060 | 0.0937 | 0.0446 | 0.011 |

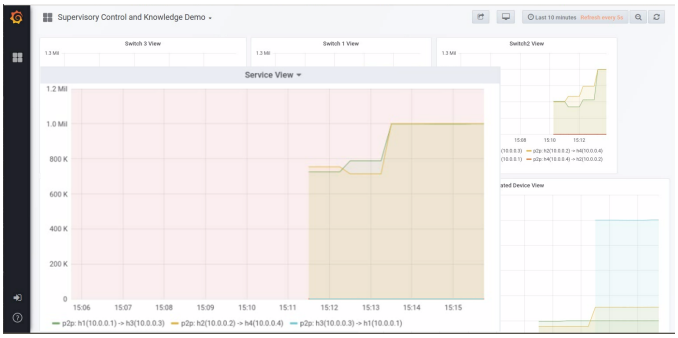


**Fig. 11:** Supervisory Control dashboard, highlighting Service View panel with throughput for p2p connection services as reported on real-time by the prototype.

via the path $[8, 7, 2, 1, 18, 19, 20]$ (See Fig. 10). When the switch 7 was simulated to fail, the new path selected was $[8, 9, 10, 16, 17, 18, 19, 20]$. A summary of the resource usage and communication overhead, for the eight runs, by each architecture is presented in Table 3. The hierarchical architecture ($H$) shows the lowest resource usage (RAM and CPU) on average. To check whether these differences are significant, we first applied the *Shapiro* normality test, obtaining the p-values shown in column $p$ of Table 3. Such a normality hypothesis is rejected for all the groups taking part in the comparison, given the small value of the p-values. Even for the group with a higher p-value, CPU usage p-value$= 0.06$, it is not considered high enough to avoid rejection of the normality hypothesis. Hence, a suitable, non-parametric test for the comparison of groups is *Friedman* test. The p-value for the comparison of the RAM groups is $0.0439$ which leads to a rejection of the hypothesis of groups having equal RAM mean. However, the post-hoc analysis for CPU and bytes transferred (Tx), shows that the difference is not significant to reject that these 2 groups have equal mean, having a p-value of $0.1969$. The RAM differences can be explained as the DA agents in $H$ have lower workload as only the $SM$ agent consolidates measurements and monitor SLAs.

This experiment shows that conventional platform services, such as agent discovery, can be implemented with the support of a system-based container framework, in this case using Docker. This way we address the first requirement identified in V-A2 as there is no fundamental dependency on a MAS platform. In addition, the agents have their own local registries of agents rather than a global one. This way multiple connection topologies and coordination protocols can be implemented with reduced changes among configurations. For example, in $H$, each agent has only the SM address whereas in $L_{sw}$ and $D_{sw}$ architectures, every DA has as many agent neighbour as links exist in the switch being controlled. We have seen this also causes DA agents of $H$ to have a lower RAM usage than in other architectures but making the SM a single point of failure (SPoF). However, this is not a default SPoF, because the control system can easily be adjusted by moving service control behaviours to DAs and updating their local registries with convenient neighbours, as is the case of $L_{sw}$ and $D_{sw}$ architectures.

Each agent container is deployed with the same Scala classes and there is a parameter, passed at launching, that indicates which behaviours (Scala traits) should be enabled in each agent, this makes the system flexible as it is easy to switch between agent architectures, just before starting each agent. The agent container image size is 117MB, and Table 3 shows that resource usage varies with the architecture. For reference, we compare these values with those of a basic installation of the JADE platform. This uses in the order of 70MB of RAM while idle and the size of a JADE image container is around 82MB. So it shows that the values obtained in our prototype are not ideal but still can be deployed in constrained devices such as a Raspberry Pi. Furthermore, there are active efforts towards making container operating systems lighter, breaking them down into libraries [45] and also for enabling customisation of execution environments.

### 2) Large Scale

The mean RAM usage profile per agent in two architectures with up to 300 agents is presented in Fig. 12. Overall, the figures show requirements of less than 170MB, on average, per agent of each system, regardless of the architecture and the number of DA agents (number of assets). After concluding that normality could not be assumed for the analysed groups, we applied *Friedman* tests to check whether the differences in RAM usage within the same architecture but different quantities of agents were significant. In this case, we obtained a p-value of $0.001031$ for $H$ and $0.000079$ for $D_r$, meaning that, the mean RAM usage per agent and architecture is the same regardless of the quantity of agents ($H_0$ accepted). Evaluating differences of the RAM per architecture for a given quantity of agents, we found that p-value was $0$ for all the quantities analysed. This means there are differences among RAM usage per architecture ($H_0$ rejected), which is also shown by the solid lines in Fig.12. However, when comparing the overall mean RAM usage per agent and architecture, it is only possible to say that $H$ architecture has lower RAM requirements (around 40MB less) per agent than the $D_r$ architecture. Only by monitoring individual agent usage per type of agent, we realise that SM agent has higher resource demands that grow linearly as the system is scaled up (inset in Fig. 12). By design this is expected, as responsibilities of Service Manager mean a higher workload that depends on the number of agents. However, this might not be always clear, for example, if the frequency of certain operation in a particular asset is higher than in others, it might cause a higher workload in the controlling DA agent. This type of demands are even more difficult to pick when the number of agents grows. Defining agents as single deployment units (containers) independent of a MAS platform, enables to set resource usage limits per agent. This way thresholds can be monitored and controlled to determine which agents require tuning or extra resources.

For the cost analysis, we define $A_{a/b}$ as the *Cost Advantage* of one deployment strategy $a$ over another $b$, as follows:

$$A_{a/b} = \frac{\hat{C}_b - \hat{C}_a}{\hat{C}_b} \quad (1)$$

Where $\hat{C}_a$ is the mean cost of the system for the strategy $a$ normalised by the number of agents and the observation time. The mean cost is calculated as the mean of the resources used by the all the agents in the system. Thus the cost advantage shows the proportion of the benefit (cost savings) of one strategy $a$ over the reference strategy $b$. The bars in Fig. 13 show the cost advantage of a 1-to-1 agent-node deployment strategy, with on-demand provisioning, over a 1-to-Many deployment strategy, in environments with three combinations of stable/transient agents. There is an advantage in the 1-to-1 over the 1-to-M strategy in all the cases analysed except when the there are 400 agents and 80% of them stable. When the proportion stable/transient is 20/80, this cost advantage of 1-to-1 strategy is consistently close to $40\%$ of the cost of the 1-to-Many strategy. However, it decreases and becomes more variable when the proportion of stable/transient agents is 50/50. Even, when the most of agents are stable the advantage is reduced to less than $10\%$ and no advantage with greatest number of agents. The standard deviation for the eight runs is plotted on top of each bar. This experiment confirms the intuition that 1-to-1 deployment strategy is more cost-effective in scenarios with high number of transient agents and also shows the potential magnitude of the benefit considering mix of agents and scale.

Reflecting on second and third requirements in section V-A2, these experiments show that the ability to monitor individual agents together with the isolation of environment facilitates the portability of agents to where resources are available. Likewise, as agents are individual deployment units the system can scale up and down bringing potential cost benefits during operation.

### E. LESSONS LEARNED

Although we presented a case in the context of telecommunication infrastructure services, this approach is closely aligned with agility and capability-based flexibility principles of cloud manufacturing [1] and the general system architecture for Industry 4.0 applications [61]. Furthermore, the shift from pure goods production towards mixed product-service business models [62], brings the service-oriented scenarios presented in this case study, closer to the manufacturing industry.

We explored the engineering of agent-based control systems by relaxing dependencies on conventional MAS platforms, the case study helped us to bring the following reflections:

- *Dynamic Control.* As industrial systems are increasingly dynamic, it becomes more relevant for the control system to have the flexibility to change its structure. The approach explored provides flexibility to easily change agent system topologies and for communication there is no need of a single agent directory.
- *Decoupled Services.* One benefit of the approach explored is that the supporting tools for agent monitoring, are not integrated to the MAS platform, but are separate services, running in their own containers. This facilitates the replacement of such tools (e.g. Prometheus and Grafana) for alternatives without requiring to migrate agents to another MAS platform. So the services normally offered by a single MAS platform can be consumed from specialised of-the-shelf solutions.
- *Automated Asset Generation.* Due to the container-based virtualisation, new agents can be created from the same template in an automated process, this approach enables the control system to grow in the number of agents as required and to move them along the available infrastructure. Another consequence of these characteristics is that migration of control systems can be eased progressively per control function, rather than requiring a big-bang approach.
- *Programming Language Independence.* Although all our agents were developed in Scala, the discussed approach also offers potential for programming language independence as the only requirement for interaction with other agents of the control system is to be able to send/receive messages according to the ad hoc protocol.
- *Customisation & Resource Usage Improvement.* The image size and resource usage profiles of the agent containers is an area where significant improvements can be made. Key to improve this aspect is the availability of more modular and customisable runtime environments, together with Unikernels and library operating systems. Tools for easing development in different programming languages using these technologies are still insufficient.
- *Complementing MAS Platforms.* Container-based vir-

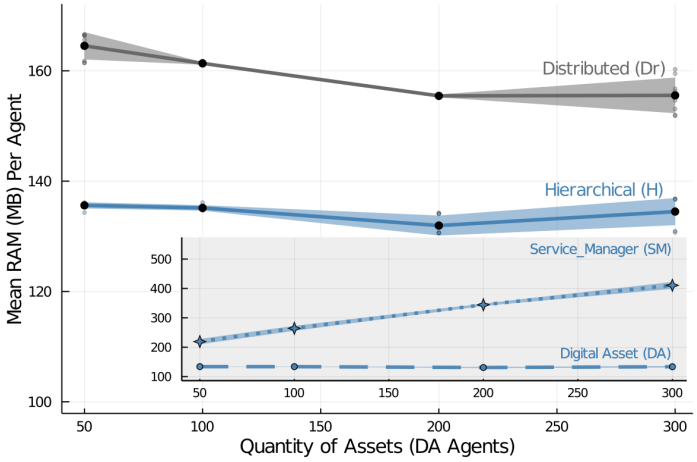


**Fig. 12:** Mean RAM usage for two architectures with up to 300 DA agents. The inset shows mean usage per type of agent in the Hierarchical architecture.

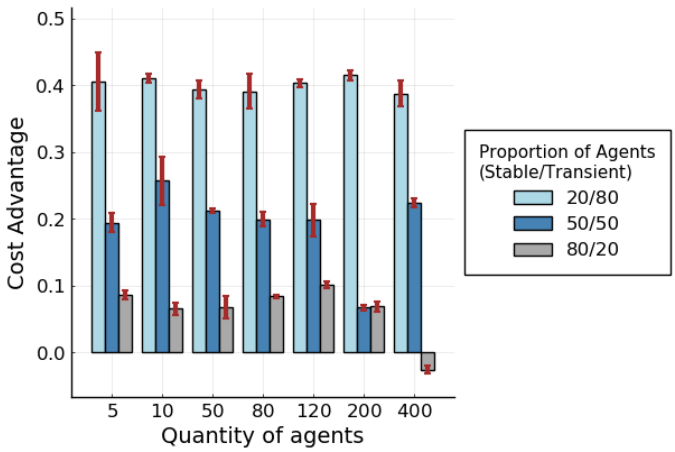


**Fig. 13:** Cost advantage due to on-demand provisioning of agents deployed in individual containers (1-to-1 strategy) for three proportions of stable/transient agents.

tualisation can be used as a complement of MAS platforms. However, there are platforms that implement the container pattern at application level. Working with two levels of containers increases complexity of the development process.

- *Hard Real-time Limitations* Although system containers do not offer hard real-time performance that some industrial applications require, there are multiple ongoing efforts to address this issue [63].

## VI. CONCLUSIONS AND FUTURE WORK

This paper introduces an approach for the engineering of agent-based control systems while relaxing dependencies in MAS platforms. In this approach agents are self-contained applications that are deployed independently of others. The paper also presents a reference control architecture, based on the proposed approach, where decoupled control behaviours are composed into agent images that can be configured for activation on start up. As a result, each agent structure can be easily replicated or deployed along the available hardware infrastructure.

The feasibility of this approach was demonstrated with a case study for the implementation of a supervisory control system in digital network infrastructures. Three variations of the architecture were implemented in a prototype which monitors and controls the condition-driven operation of an emulated network. We tested the ease of portability and ability to monitor resources with specialised decoupled off-the-shelf solutions that substitute equivalent MAS platform services. We found containerised agents are more demanding in terms of computing resources than traditional agents embedded in agent platforms, however this was not a problem for the studied case and there is room for optimisation. This approach also facilitates monitoring of resource usage for each individual agent.

In addition, we analysed the effect of defining agents as individual deployment units, in the overall cost of operation of the control system. The simulation results showed that this approach is cost-effective especially when there is a higher proportion of transient agents than stable ones in the control system.

A route for future works is to explore the creation of containerised agents based on Unikernels. To date, tools for creation and monitoring of Unikernels are scarce and still at an early stage which represents a barrier for the evaluation.

## ACKNOWLEDGEMENTS

Authors thank project partners for the fruitful discussions, especially with colleagues of BT and the University of Lancaster. Likewise, we thank Boyue Zhang from the Computer Science program of the University of Cambridge who contributed to the development of the prototype and the tools used in the case study.

## REFERENCES

[1] L. Zhang, Y. Luo, F. Tao, B. H. Li, L. Ren, X. Zhang, H. Guo, Y. Cheng, A. Hu, and Y. Liu, "Cloud manufacturing: a new manufacturing paradigm," *Enterprise Information Systems*, vol. 8, no. 2, pp. 167–187, 2014.

[2] ETSI Industry Specification Group, "Virtual Network Functions Architecture," Tech. Rep., 2014.

[3] S. Karnouskos, L. Ribeiro, P. Leitao, A. Luder, and B. Vogel-Heuser, "Key directions for industrial agent based cyber-physical production systems," *Proceedings - 2019 IEEE International Conference on Industrial Cyber Physical Systems, ICPS 2019*, pp. 17–22, 2019.

[4] O. Shehory and A. Sturm, "Agent-oriented software engineering: Reflections on architectures, methodologies, languages, and frameworks," *Agent-Oriented Software Engineering: Reflections on Architectures, Methodologies, Languages, and Frameworks*, vol. 9783642544323, pp. 1–331, 2014.

[5] P. Di Francesco, P. Lago, and I. Malavolta, "Architecting with microservices: A systematic mapping study," *Journal of Systems and Software*, vol. 150, pp. 77–97, 2019.

[6] M. Richards, *Software Architecture Patterns*, H. Scherer, Ed. Sebastopol, CA: O'Reilly, 2015.

[7] B. Esmaeilian, S. Behdad, and B. Wang, "The evolution and future of manufacturing: A review," *Journal of Manufacturing Systems*, vol. 39, pp. 79–100, 2016.

[8] H. A. Abbas, "Future SCADA challenges and the promising solution: The agent-based SCADA," *International Journal of Critical Infrastructures*, vol. 10, no. 3-4, pp. 307–333, 2014.

[9] ISO/IEC, "ISO/IEC 25010:2011 - Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models," 2011.

[10] IEEE, "Iso/iec/ieee international standard - systems and software engineering–vocabulary," *ISO/IEC/IEEE 24765:2017(E)*, pp. 1–541, 2017.

[11] A. V. Kapitanov, "Manufacturing System Flexibility Control," *Procedia Engineering*, vol. 206, pp. 1470–1475, 2017.

[12] P. Leitão, "An agile and adaptive holonic architecture for manufacturing control," 2004.

[13] P. Jogalekar, M. Woodside, and S. Member, "Evaluating the Scalability of Distributed Systems," vol. 11, no. 6, pp. 589–603, 2000.

[14] M. Herrera, M. Pérez-Hernández, A. Kumar Parlikad, and J. Izquierdo, "Multi-agent systems and complex networks: Review and applications in systems engineering," *Processes*, vol. 8, no. 3, p. 312, 2020.

[15] P. Leitão and F. Restivo, "ADACOR: A holonic architecture for agile and adaptive manufacturing control," *Computers in Industry*, vol. 57, no. 2, pp. 121–130, 2006.

[16] R. M. da Silva, M. F. Blos, F. Junqueira, D. J. Santos Filho, and P. E. Miyagi, "A Service-Oriented and Holonic Control Architecture to the Reconfiguration of Dispersed Manufacturing Systems," *IFIP Advances in ICT*, vol. 423, pp. 111–118, 2014.

[17] F. Bergenti, G. Caire, and D. Gotta, "Large-Scale Network and Service Management with WANTS," *Industrial Agents: Emerging Applications of Software Agents in Industry*, pp. 231–246, 2015.
[18] H. Elshaafi, M. Vinyals, I. Grimaldi, and S. Davy, "Secure Automated Home Energy Management in Multi-Agent Smart Grid Architecture," *Technology and Economics of Smart Grids and Sustainable Energy*, vol. 3, no. 1, 2018.
[19] F. L. Bellifemine, G. Caire, A. Poggi, and G. Rimassa, "Jade A White Paper," Telecom Italia Lab, Tech. Rep., 2003.
[20] O. Boissier, R. H. Bordini, J. F. Hübner, A. Ricci, and A. Santi, "Multi-agent oriented programming with jacamo," *Science of Computer Programming*, vol. 78, no. 6, pp. 747–761, 2013.
[21] R. Rönnquist, "The goal oriented teams (gorite) framework," in *International Workshop on Programming Multi-Agent Systems*. Springer, 2007, pp. 27–41.
[22] K. Kravari and N. Bassiliades, "A survey of agent platforms," *Journal of Artificial Societies and Social Simulation*, vol. 18, no. 1, p. 11, 2015.
[23] C.-V. Pal, F. Leon, M. Paprzycki, and M. Ganzha, "A review of platforms for the development of agent systems," *arXiv preprint arXiv:2007.08961*, 2020.
[24] A. Sturm and O. Shehory, "The evolution of mas tools," in *Agent-Oriented Software Engineering*. Springer, 2014, pp. 275–288.
[25] F. Bellifemine, G. Caire, A. Poggi, and G. Rimassa, "JADE: A software framework for developing multi-agent applications. Lessons learned," *Information and Software Technology*, vol. 50, no. 1-2, pp. 10–21, 2008.
[26] A. G. T. FIPA, "Fipa abstract architecture specification - tech. rep. xc00001k," Foundation for Intelligent Physical Agents, Geneva, Tech. Rep., 2002.
[27] J. D. Jong, L. Stellingwerff, and G. E. Pazienza, "Eve: A Novel Open-Source Web-Based Agent Platform," in *2013 IEEE International Conference on Systems, Man, and Cybernetics*. Ieee, oct 2013, pp. 1537–1541.
[28] F. L. Bellifemine, G. Caire, D. Greenwood, and Wiley InterScience (Online service), *Developing multi-agent systems with JADE*. Chichester, England; Hoboken, NJ: John Wiley, 2007.
[29] Q. He, J. Yan, Y. Yang, R. Kowalczyk, and H. Jin, "A decentralized service discovery approach on peer-to-peer networks," *IEEE Transactions on Services Computing*, vol. 6, no. 1, pp. 64–75, 2013.
[30] M. Brambilla, E. Ferrante, M. Birattari, M. Dorigo, M. Brambilla, E. Ferrante, M. Birattari, M. Dorigo, and S. Intell, "Swarm Intell Swarm robotics : a review from the swarm engineering perspective," 2016.
[31] A. Ricci, O. Boissier, A. Ciortea, R. H. Bordini, S. Mayer, and J. F. Hiibner, "Engineering scalable distributed environments and organisations for mas," *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS*, vol. 2, pp. 790–798, 2019.
[32] A. R. Sampaio, H. Kadiyala, B. Hu, J. Steinbacher, T. Erwin, N. Rosa, I. Beschastnikh, and J. Rubin, "Supporting microservice evolution," *Proceedings - 2017 IEEE International Conference on Software Maintenance and Evolution, ICSME 2017*, pp. 539–543, 2017.
[33] S. Newman, *Building Microservices: Designing Fine-Grained Systems*, 1st ed., M. Loukides and B. MacDonald, Eds. O'Reilly, 2015.
[34] J. Thönes, "Microservices," *IEEE Software*, vol. 32, no. 1, 2015.
[35] T. Yarygina and A. H. Bagge, "Overcoming Security Challenges in Microservice Architectures," *Proceedings - 12th IEEE International Symposium on Service-Oriented System Engineering, SOSE 2018 and 9th International Workshop on JCC 2018*, pp. 11–20, 2018.
[36] S. Sarkar, G. Vashi, and P. P. Abdulla, "Towards Transforming an Industrial Automation System from Monolithic to Microservices," *IEEE International Conference on Emerging Technologies and Factory Automation, ETFA*, vol. 2018-Septe, pp. 1256–1259, 2018.
[37] T. Salah, M. J. Zemerly, C. Y. Yeun, M. Al-Qutayri, and Y. Al-Hammadi, "Iot applications: From mobile agents to microservices architecture," in *2018 International Conference on Innovations in Information Technology (IIT)*. IEEE, 2018, pp. 117–122.
[38] T. Leppänen, C. Savaglio, L. LoveÌĄn, T. Järvenpää, R. Ehsani, E. Peltonen, G. Fortino, and J. Riekki, "Edge-based microservices architecture for Internet of Things: Mobility analysis case study," *IEEE Global Communications Conference (GLOBECOM)*, no. August, pp. 1–7, 2019.
[39] R. W. Collier, E. O. Neill, and G. M. P. O. Hare, "MAMS: Multi-Agent MicroServices," 2019, pp. 655–662.
[40] R. Morabito, J. Kjällman, and M. Komu, "Hypervisors vs. lightweight virtualization: A performance comparison," *Proceedings - 2015 IEEE International Conference on Cloud Engineering, IC2E 2015*, pp. 386–393, 2015.
[41] P. Sharma, L. Chaufournier, P. Shenoy, and Y. C. Tay, "Containers and Virtual Machines at Scale," pp. 1–13, 2016.
[42] R. Dua, A. R. Raja, and D. Kakadia, "Virtualization vs containerization to support PaaS," in *Proceedings - 2014 IEEE International Conference on Cloud Engineering, IC2E 2014*. IEEE, 2014, pp. 610–614.
[43] H. Kang, M. Le, and S. Tao, "Container and microservice driven design for cloud infrastructure DevOps," *Proceedings - 2016 IEEE International Conference on Cloud Engineering, IC2E 2016: Co-located with the 1st IEEE International Conference on Internet-of-Things Design and Implementation, IoTDI 2016*, pp. 202–211, 2016.
[44] C. Ebert, G. Gallardo, J. Hernantes, and N. Serrano, "DevOps," *IEEE Software*, vol. 33, no. 3, pp. 94–100, 2016.
[45] A. Madhavapeddy and D. J. Scott, "Unikernels: The Rise of the Virtual Library Operating System," *ACM Queue - Distributed Computing*, vol. 11, no. 11, p. 30, 2013.
[46] T. Goldschmidt, S. Hauck-Stattelmann, S. Malakuti, and S. Grüner, "Container-based architecture for flexible industrial control applications," *Journal of Systems Architecture*, vol. 84, no. November 2017, pp. 28–36, 2018.
[47] F. Hofer, M. A. Sehr, A. Iannopollo, I. Ugalde, A. Sangiovanni-Vincentelli, and B. Russo, "Industrial control via application containers: Migrating from bare-metal to IAAS," *Proceedings of the International Conference on Cloud Computing Technology and Science, CloudCom*, vol. 2019-Decem, pp. 62–69, 2019.
[48] S. Bussmann, N. R. Jennings, and M. Wooldridge, *Multiagent Systems for Manufacturing Control*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004.
[49] H.-H. Choi and J.-R. Lee, "Local flooding-based on-demand routing protocol for mobile ad hoc networks," *IEEE Access*, vol. 7, pp. 85 937–85 948, 2019.
[50] R. Jabbari, N. Ali, and K. Petersen, "What is DevOps ? A Systematic Review on Definitions and Practices," *Proceedings of the Scientific Workshop Proceedings of XP2016 (XP '16 Workshops)*, 2016.
[51] L. Ribeiro, *The Design, Deployment, and Assessment of Industrial Agent Systems*. Elsevier Inc., 2015.
[52] W. L. Hürsch and C. V. Lopes, "Separation of concerns," 1995.
[53] S. Sezer, S. Scott-Hayward, P. Kaur Chouhan, B. Fraser, D. Lake, C. Systems Jim Finnegan, N. Viljoen, N. Marc Miller, N. Rao, and S.-h. Layout, "Are We Ready for SDN? Implementation Challenges for Software-Defined Networks," *Future Carrier Networks*, vol. 51, no. 7, pp. 36–43, 2013.
[54] F. Leitão, R. David Carnero Ros, and J. RiusRiu, "Fixed-mobile convergence towards the 5G era: Convergence 2.0: The past, present and future of FMC standardization," *2016 IEEE Conference on Standards for Communications and Networking, CSCN 2016*, pp. 1–6, 2016.
[55] V. Q. Rodriguez, F. Guillemin, and A. Boubendir, "Automating the deployment of 5G Network Slices with ONAP," 2019.
[56] A. Stavdas, *Core and Metro Networks*, 2010.
[57] G. Wikstrom, J. Peisa, P. Rugeland, N. Johansson, S. Parkvall, M. Girnyk, G. Mildh, and I. L. Da Silva, "Challenges and technologies for 6G," *2nd 6G Wireless Summit 2020: Gain Edge for the 6G Era, 6G SUMMIT 2020*, 2020.
[58] D. Kreutz, F. M. V. Ramos, P. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-Defined Networking: A Comprehensive Survey," pp. 1–61, 2014.
[59] M. Newman, *Networks*. Oxford university press, 2018.
[60] M. E. Newman and D. J. Watts, "Renormalization group analysis of the small-world network model," *Physics Letters A*, vol. 263, no. 4-6, pp. 341–346, 1999.
[61] E. Trunzer, A. Calà, P. Leitão, M. Gepp, J. Kinghorst, A. Lüder, H. Schauerte, M. Reifferscheid, and B. Vogel-Heuser, "System architectures for Industrie 4.0 applications: Derivation of a generic architecture proposal," *Production Engineering*, vol. 13, no. 3-4, pp. 247–257, 2019.
[62] L. Mastrogiacomo, F. Barravecchia, and F. Franceschini, "A worldwide survey on manufacturing servitization," *International Journal of Advanced Manufacturing Technology*, 2019.
[63] V. Struhár and A. V. Papadopoulos, "Real-Time Containers : A Survey," no. 7, pp. 1–7, 2020.

• • •