Orchestration Done Upside Down: Self-aware Applications for Substation Automation

Christian Göttel, David Kozhaya, Enrico Fregnan, Philipp Sommer, Sandro Schönborn,
ABB Schweiz AG, Corporate Research Center, Baden-Dättwil, Switzerland, `first.last@ch.abb.com`

***Abstract*—Electrical substations run real-time Protection and Control (P&C) applications to ensure the safety of the power grid. To guarantee that the operational constraints of these applications are met, P&C platforms are generally provisioned with spare hardware resources. In this paper, we propose an orchestration solution to deploy non-safety-critical P&C applications taking advantage of all those spare hardware resources. Each application instance is made self-aware by including a monitoring and governance service that periodically checks the global execution constraints (*e.g.*, execution time limits) from an application rather than a system perspective and takes actions if these are not met: *i.e.*, by voluntarily terminating the application instance. Contrary to traditional microservice architectures, an orchestrator component is only used for deploying application instances from a list of to be executed applications onto available spare resources.**

***Index Terms*—Real-time systems, Protection & Control, Self-aware applications, Microservices, Autonomous systems**

I. Introduction

Systems utilizing real-time software are ubiquitous nowadays. Examples include aviation and aerospace [1], [2], healthcare [3], and even communication and messaging systems [4]. The main characteristic of real-time software is its ability to respond within a known fixed time frame to stimuli from the environment [5]. Electrical substations as in Figure 1 are one domain where the use of real-time software is crucial. Substations are responsible for routing power from generators to loads through a network of transmission lines as well as protecting the power grid from incidents [6]. For this reason, substations run specialized real-time applications, called Protection and Control (P&C) functions, to swiftly react to anomalies in the grid and take appropriate mitigation actions.

To guarantee that P&C functions always adhere to their real-time performances, providers generally overprovision the hardware resources on which P&C applications are hosted. Furthermore, the execution of these applications is more often than not bound to vendor-specific hardware as well as predefined compute units and configuration options. These factors altogether make the addition of any new P&C application costly in terms of both hardware resources and engineering effort, while simultaneously leading to a vast amount of computational resources being left unused.

In this paper, we propose a new paradigm that leverages spare hardware resources of substations by simultaneously deploying new non-safety-critical (*e.g.*, data analytics applications) alongside safety-critical applications without disrupting the performance of the safety-critical applications or overprovisioning the substation automation system.

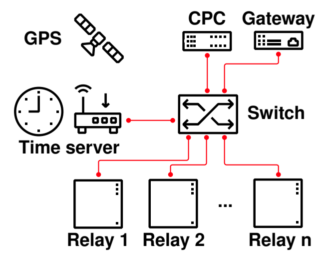

Fig. 1. Substation automation system with relays for digitizing readings of the grid streamed to CPC device. Time servers are receiving the clock signal from GPS satellites and use it for synchronizing all devices in the substation.

Deploying applications on multiple hardware resources requires a central service that keeps track of the execution status of each application (or even application instance). However, acquiring such system-wide knowledge, particularly in real-time, is challenging and may introduce inaccuracies that negatively impact the execution of real-time applications [7]. To this end, we propose a solution that shifts the burden of monitoring and controlling application execution from a centralized global service to the local individual application instances: New non-safety-critical applications are made self-aware through the addition of a *monitoring and governance service*, which oversees the global operational status of an application instance to ensure that its constraints (defined at deployment time) are met. If these constraints are not met, an application can independently change its operational status: *e.g.*, requesting an orchestrator to deploy additional instances or terminating unnecessary instances.

In addition, an orchestrator keeps track of all pending applications to execute and deploys the next application based on defined metrics, such as the available hardware resources.

II. Background & Related Work

Background. With the advent of the cloud, microservices have become the most popular solution for deploying applications on the Internet. Microservices are deployed and managed through an (open-source) orchestrator, *e.g.*, Apache Mesos, Docker Swarm, or Kubernetes. The orchestrator has complete insight into the system and deploys microservices based on the available and required resources as well as predefined policies: *e.g.*, affinity, access to shared resources, and scalability. Throughout the life cycle of a microservice, the orchestrator remains in full control over the microservice for resource accounting and scheduling purposes. Microservices

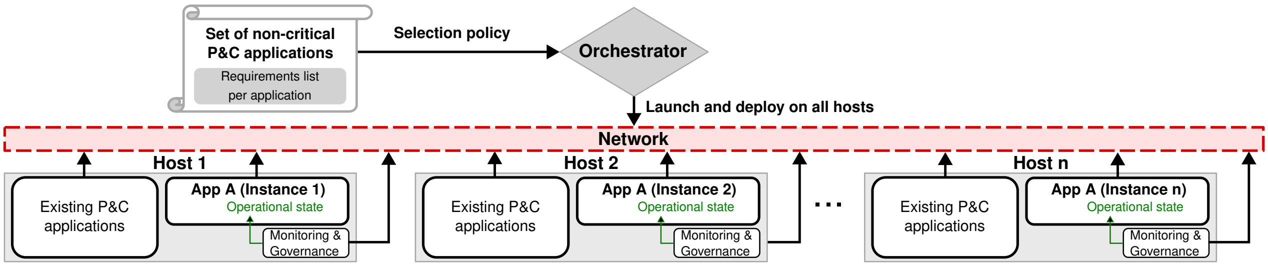

Fig. 2. General overview of our proposed solution design. The orchestration service selects a candidate application to deploy from the set of non-critical P&C applications based on available resources on the hosts within the network. The monitoring & governance service within each instance provides information about the operational state.

are implemented following a stateless design and leverage virtualization technologies (*e.g.*, containers, virtual machines), which abstract away the surrounding system state and make them dependent on orchestration actions.

In the move towards Industry 4.0, Operational Technology (OT) systems have been interconnected with Information Technology (IT) systems to provide automation, cloud connectivity, and remote control capabilities. Substation's Centralized Protection & Control (CPC) systems [8], as in Figure 1, play a key role in today's substation automation. These systems aggregate streams of digitized readings from the grid on a single device. The readings from these streams originate from relays and are then processed by the P&C applications executed on the CPC devices. P&C applications require specific resources and real-time guarantees to respect the strict deadlines for sending signals to trigger actuators in the event of an incident (*e.g.*, a lightning hitting a transmission line).

Related work. Similar to the solution presented in this paper, other initiatives like ACTORS [9] and DREAMS [10] projects propose to automatize the distribution of resources on multi-core systems at run-time (*e.g.*, for streaming and avionics applications). In ACTORS, similar to our orchestrator, a centralized resource manager assigns the necessary resources (*i.e.*, *service levels*) to each application to be deployed. Each application can also adjust its resource consumption dynamically by means of a bandwidth controller. Such controller shares similarities with our monitoring service, but its functionalities are limited to bandwidth adjustments and do not allow, *e.g.*, the application to terminate itself autonomously.

In a different approach, the DREAMS project introduces a hierarchical resource management platform, where a Global Resource Manager acts in combination with multiple Local Resource Managers. Each Local Resource Manager possesses (1) a Resource Monitor, which monitors resource availabilities and timing constraints of an application, and (2) a Local Resource Scheduler, which performs the run-time scheduling of resources based on the configuration set by the Local Manager. This platform allows the scheduling of mixed-criticality applications, while our solution only focuses on the deployment of non-critical applications using the "leftover" resources. This minimizes the responsibilities assigned to the centralized resource manager (*i.e.*, orchestrator) – no need to implement complex scheduling strategies to guarantee the timing deadlines of critical applications.

In the context of cyber-physical systems that lack sufficient resources for having cybersecurity mechanisms, Maurio *et al.* [11] explored a similar approach to ours. *Agile services* were evaluated under two different use cases: (1) as microservices retrofitting Industrial Control Systems (ICSs) with autonomic properties and (2) as intelligent agents for a simulation framework of autonomous Unmanned Aerial Systems (UASs). Mikkelsen *et al.* [12] explored architectures and design principles for autonomous microservices. They identified several enabling properties such as asynchronous non-blocking point-to-point communication, message brokering done by a dedicated service, no traffic controlling service, and low resource consumption for scalability.

In the field of Internet of Things (IoT) and digital twin systems, Ferrera *et al.* [13] proposed an autonomic computing approach (BRAIN-IoT) where systems were able to take timely actions and learn from the past through the support of cognitive capabilities. Their design enables automation for the management of large-scale IoT-based systems. However, another challenge is represented by the continuously fluctuating conditions under which IoT systems are deployed. To meet the design goals and requirements for such applications, Lalanda *et al.* [14] explored design and run-time models to improve autonomy and reduce the need for human interaction.

Different organizations are currently standardizing CPC system architectures, such as E4S [15] and SEAPATH [16], which provide real-time capabilities to traditional cloud stacks for substation automation.

III. SYSTEM DESIGN

This section illustrates how new non-critical P&C applications can be autonomously (i) deployed on existing substation resources (ii) scaled onto those resources and (iii) monitored for correct behavior. We detail the different components for enabling self-awareness within P&C applications by leveraging a minimal set of services without reinventing the wheel.

A. Solution Approach Overview

The proposed solution entails two services: an orchestration service – a "centralized" system wide service – and a decentralized dependability monitoring and governance service.

Orchestration Service. Our orchestrator has a system-wide view of all available substation resources, as shown in Figure 2. It can obtain this information since we assume that the critical systems, the hardware they run on, as well as their worst case performances are known and fixed. In addition, since the orchestrator is the sole entity that can deploy additional applications, it can calculate the expected remaining resource availability in a dynamic fashion based on the predefined resource needs of applications that have been deployed and are still running. The orchestrator holds a list of all new non-critical P&C applications that are supposed to run. For each new non-critical P&C application, its needs and requirements have to be specified from a predefined set of attributes: *e.g.*, CPU, memory, I/O, architecture, timing deadlines, resilience to process faults (no resilience, crash resilience, Byzantine resilience, *etc*.), and number of instances. These new non-critical P&C applications are then deployed onto available resources, as illustrated in Figure 2. However, unlike traditional microservice orchestrators, our orchestrator is limited to the above functionality.

The orchestrator uses a novel selection policy for deploying P&C applications from the list. This policy is driven by the type of the already deployed applications as well as the rate at which those applications fail to be deployed. Once the selection is done, the instance(s) of the chosen application are dispatched to be deployed on all available substation resources.

Dependability Monitoring and Governance Service. This service is defined within every deployed instance of a P&C application on some substation resource. The dependability monitoring and governance service provides:

1) Information to individual application instance(s) as well as the application as a whole to determine at run-time normal and correct operation. Correct operation is measured by the instances' ability to satisfy all the application needs and requirements according to the application's set of defined attributes. Hence, every application instance is aware of its environment and its ability to meet – individually or jointly with other running instances – the desired application needs.

2) The ability to change the operating mode of an application instance. For example, application instances that cannot meet predefined application requirements or that are not essential for the correct operation of the entire application can use this service. Applications can gradually scale down to use fewer resources by shutting down unnecessary or non-functional instances while maintaining correct application operation.

If an application has no successfully running instances on any of the available resources, then the application has failed to deploy and the orchestrator will attempt re-deploying later.

B. Detailed Solution Description

As mentioned earlier, P&C systems within a given asset, *e.g.*, a substation, are commissioned and run on their respective infrastructure, which is typically vendor-specific hardware. In the proposed solution, all additional non-critical P&C applications that a substation operator would like to run – be them real-time or not – *e.g.*, redundant applications for better resilience or data analytics applications, are added to a pool of applications. Every application in the pool should define its requirements by specifying its needs from a list of predefined attributes including:

- The different independent components forming the entire application (these components can run in parallel).
- The desired level of redundancy of each components.
- The required type of resilience against faults: *e.g.*, crash resilience for being able to tolerate honest crashes of the devices hosting applications, or byzantine resilience for being able to tolerate arbitrary malicious behavior of devices hosting applications, *etc*.
- Application execution requirements: *e.g.*, required execution/response time. This metric can be defined at the level of the application components.
- An approximate of the application's resource needs, memory, CPU, and I/O.
- The need to execute on a given hardware architecture, *e.g.*, ARM, x86, *etc*. or using a specific hypervisor.

An orchestrator is aware of all hardware used for P&C in a given asset, which includes hardware necessary to run the P&C applications as well as any underlying platform such as operating system (*e.g.*, Linux) or hypervisor (*e.g.*, VMware), and has access to the pool of available applications. The orchestrator decides which application to run next from the pool and when to deploy that application. This selection process is smartly adapted to the feedback the orchestrator receives from failed deployments. More specifically, an orchestrator monitors the rate at which applications fail to deploy and their predefined needs. Based on those two metrics the orchestrator adapts its selection strategy and the rate at which it dispatches applications from the pool for deployment.

After selecting an application from the pool, the orchestrator deploys virtual instances of the application components to all available substation resources with an inferior priority compared to the critical systems running on that hardware. After this step, the application's execution is no longer controlled by the orchestrator. Unlike traditional microservices architectures, the orchestrator's job in our solution is minimal and is limited to selecting which application to run next and performing its deployment. The management decisions, from scaling to assessing conformance to expected behavior, are taken autonomously by each application (or application instance).

Every application instance is equipped with a dependability monitoring and governance service that judges if that application instance is operating as expected (according to the predefined application requirement attributes). For example, the monitoring service within each virtual instance verifies if the instance's approximate resources are met, if the instance's performance meets the expected timing deadlines, and if the threshold for the required fault resilience is met. If any of the predefined requirements is not met, then the monitoring service of that virtual instance notifies the orchestrator and instantiates a mechanism to allow the instance to change its mode of operation: *e.g.*, by terminating itself and decommissioning from the host. In the event where termination receipts from all instances of an application are detected, the orchestrator assumes a *failed application deployment* and returns the application back to the pool for later deployment.

The dependability monitoring and governance service of an application instance can be viewed as a built-in service to extract information both local (from the host it runs on) and remote (other hosts running instances of the same application), essential for that local instance to decide on which operation mode to assume. To obtain information on the host (*e.g.*, currently available CPU) the monitoring service relies on Linux or the hypervisor. This service may be devised as a standalone library that applications (application instances) must be developed with. On the one hand, the extraction of local information, *i.e.*, information extracted from the host on which this application instance runs, can be built on top of (1) existing tools, for example those used in monitoring CPU and memory usage of the host machine, and/or (2) built-in loggers that provide performance statistics. On the other hand, regarding the extraction of remote knowledge and exchanging information with other instances, the dependability monitoring and governance service can execute periodical heartbeat exchanges with the rest of the nodes in the system. The frequency of sending heartbeats is determined by the response time needed by that application. The heartbeat messages themselves are used as a vessel to carry all information that should be communicated by the application instance with the outside world. Such a technique allows the application instance to track its visibility to all other instances and the reachability of its messages in real-time [7]. These heartbeats can also help determine if the needed resilience level of an application is satisfied by monitoring the number of alive instances of that application in the entire system.

The dependability monitoring and governance service can also suggest to an application instance to change its mode of operation: *e.g.*, terminate itself. This can happen, for instance, when the service determines that an application instance is not needed to ensure that the application's operational requirements are met. Hence an application is rendered *elastic* as it can scale down or up itself to a smaller or larger set of resources judged necessary for the correct operation of the application. For an application to scale, consensus is necessary amongst all running instances of that application, which collectively decide which instances can be released.

The change in operation mode is not limited to the application instance being on or off. The application behavior can consider the number and state of other available instances. Such an application can adjust its functionality to the state of the system. For example, a non-critical protection function may enforce tighter protection limits if it is running alone, whereas it can be more permissive when knowing that another high-level protection application will act on the now missed faults with more processing insights. The system as a whole can thus achieve a well-defined degradation behavior.

IV. Conclusion & Open Research Challenges

This paper presented a system to deploy non-safety-critical P&C functions leveraging the spare hardware resources in a substation, where each application instance monitors and evaluates its own operational status. In case of operational violations (*e.g.*, not meeting the required timing deadlines), an application is terminated and returned to a central pool. An orchestrator gathers all applications to be executed and deploys them when system resources become available.

Such a design can pave the road towards more intelligent and flexible substations, executing not only fundamental P&C functions but also a plethora of other non-safety-critical applications – real-time or non-real-time (*e.g.*, for data collection and analytics) – to improve substation functionalities. Nonetheless, further studies should be conducted to better understand the limits of this architecture: *e.g.*, how network faults can affect the coordination among applications' instances.

References

[1] J. H. Lala and R. E. Harper, "Architectural principles for safety-critical real-time applications," *Proceedings of the IEEE*, vol. 82, no. 1, pp. 25–40, 1994.

[2] T. Abdelzaher, E. M. Atkins, and K. G. Shin, "Qos negotiation in real-time systems and its application to automated flight control," *IEEE Trans. on Comput.*, vol. 49, no. 11, pp. 1170–1183, 2000.

[3] A. I. Siam, M. A. El-Affendi, A. Abou Elazm, G. M. El-Banby, N. A. El-Bahnasawy, F. E. Abd El-Samie, and A. A. Abd El-Latif, "Portable and real-time iot-based healthcare monitoring system for daily medical applications," *IEEE Trans. on Computational Social Syst.*, 2022.

[4] C. Nicolaou, "An architecture for real-time multimedia communication systems," *IEEE Journal on selected areas in communications*, vol. 8, no. 3, pp. 391–400, 1990.

[5] T. E. Bihari and K. Schwan, "Dynamic adaptation of real-time software," *ACM Trans. on Computer Syst.*, vol. 9, no. 2, pp. 143–174, 1991.

[6] M. Kezunovic, "Substation automation research frontiers," in *IEEE Power Systems Conference and Exposition*, 2009, pp. 1–2.

[7] D. Kozhaya, J. Decouchant, V. Rahli, and P. E. Veríssimo, "PISTIS: an event-triggered real-time byzantine-resilient protocol suite," *IEEE Trans. Parallel Distributed Syst.*, vol. 32, no. 9, pp. 2277–2290, 2021.

[8] *IEC 61850: Communication networks and systems for power utility automation*, International Eletrotechnical Commission, Geneva, Switzerland, 2003.

[9] E. Bini, G. Buttazzo, J. Eker, S. Schorr, R. Guerra, G. Fohler, K.-E. Arzen, V. Romero, and C. Scordino, "Resource management on multicore systems: The actors approach," *IEEE Micro*, vol. 31, no. 3, pp. 72–81, 2011.

[10] G. Durrieu, G. Fohler, G. Gala, S. Girbal, D. G. Pérez, E. Noulard, C. Pagetti, and S. Pérez, "Dreams about reconfiguration and adaptation in avionics," in *ERTS 2016*, 2016.

[11] J. Maurio, P. Wood, S. Zanlongo, J. Silbermann, T. Sookoor, A. Lorenzo, R. Sleight, J. Rogers, D. Muller, N. Armiger, C. Rouff, and L. Watkins, "Agile services and analysis framework for autonomous and autonomic critical infrastructure," *Innovations Syst Softw Eng*, vol. 9, pp. 145–156, 2023.

[12] A. Mikkelsen, T.-M. Grønli, D. Tamburri, and R. Kazman, "Architectural Principles for Autonomous Microservices," in *Hawaii International Conference on System Sciences*, 2020, pp. 6569 – 6578.

[13] E. Ferrera, X. Tao, D. Conzon, V. S. Pombo, M. Cantero, and T. Ward, "BRAIN-IoT: Paving the Way for Next-Generation Internet of Things," in *International Conference on Internet of Things, Big Data and Security*, 2020, pp. 470—477.

[14] P. Lalanda, S. Chollet, and C. Hamon, *Leveraging Design and Runtime Architecture Models to Support Self-awareness*, 2017, pp. 669–686.

[15] E. Alliance, "Edge for Smart Secondary Substation (E4S) Alliance," https://e4salliance.org/, May 2024, Accessed on: 2024-05-16.

[16] L. F. Energy, "Software Enabled Automation Platform and Artifacts (THerein)," https://lfenergy.org/projects/seapath/, May 2024, Accessed on: 2024-05-16.