Synchronized Real-time Communication In The Eclipse Arrowhead Framework

Attila Frankó[‡&], Gergely Hollósi[‡], Dániel Ficzere[‡], and Pál Varga[‡]

Abstract—**Real-time, deterministic communication is a common requirement in most time-critical industrial applications. Due to the increasing complexity, interoperability of such systems must be handled by using appropriate design principles and convenient tools that enhance and ease composability. The Eclipse Arrowhead Framework aims to be a basis for such industrial Cyber-Physical System of Systems by providing a standard for services and enabling interoperability between systems and services. However, precise timing requirements cannot be handled on a service basis as they are closely related to hardware capabilities and the structure of the physical underlying network. This paper addresses this by proposing an extension of the Arrowhead framework to bridge synchronization between instances to the service level without losing any abstraction, thus preserving fundamental achievements of the Arrowhead, such as late binding and loose coupling.**

Index Terms—**real-time, synchronization, arrowhead, ptp, cyber-physical systems, system of systems, clock domains**

I. INTRODUCTION

In the dynamic landscape of industrial services and applications, time synchronization emerges as a critical factor influencing the efficiency, reliability, and interoperability of diverse processes. As industries undergo digital transformation and embrace cutting-edge technologies, the orchestration of interconnected Cyber-Physical Systems of Systems (CPSoS) becomes increasingly complex, necessitating precise timekeeping for seamless coordination. Accurate time synchronization is integral to successfully executing numerous industrial functions, ranging from manufacturing processes to data communication and control systems. In manufacturing, synchronized time ensures the harmonious collaboration of machines and devices, optimizing production workflows and minimizing delays. Furthermore, in distributed control systems and sensor networks, precise time synchronization is paramount for maintaining temporal coherence, enabling accurate data correlation, and enhancing overall system resilience.

Maintaining precise time synchronization becomes paramount in cloud environments, where diverse applications, services, and resources interact across geographically dispersed data centers. One of the key areas where time synchronization plays a pivotal role is in distributed computing and virtualization. Cloud services often rely on virtual machines and containers distributed across multiple servers. Accurate time synchronization is essential for these virtualized instances to operate cohesively, facilitating streamlined communication and synchronization of activities.

One of the key motivations for prioritizing time synchronization within an Industrial IoT (IIoT) framework lies in its role in enabling predictable and deterministic behaviors in industrial systems. From manufacturing processes to control systems, synchronized time ensures that actions occur precisely when needed, contributing to the overall predictability and efficiency of industrial operations. In this work, we use the Arrowhead IIoT framework for its capabilities, such as loose coupling, interoperability, and widespread research support, facilitating the practical application of new results. The Arrowhead framework's commitment to time synchronization is also evident in its support for distributed architectures and service-oriented paradigms. As industrial systems evolve toward decentralized and interconnected models, precise timekeeping becomes indispensable for maintaining synchronization among distributed entities, thereby enhancing system resilience and responsiveness.

Within our paper, we introduce the architecture and functioning of the synchronized real-time communication embedded in the Arrowhead Framework. Additionally, the paper delineates the primary requirements for time synchronization functionality, generally for Service Oriented Architectures (SoA) and also specifically within the Arrowhead framework, accompanied by a case study focusing on scheduled execution and synchronized service orchestration.

The paper structured as follows: Section II introduces the major concepts and elements of the Arrowhead framework and PTP protocol as the primary enabler of precise time synchronization. Section III details the motivation and requirements for achieving synchronized communication between services in Arrowhead and also presents the proposed architectural extension. Section IV shows an application example of the proposed solution while section V concludes the paper.

II. RELATED WORKS

A. Arrowhead Framework

We outline the fundamental principles of the Arrowhead project to understand the key guidelines and motivations behind the development of its time synchronization architecture. The Arrowhead project is guided by three core principles, which are as follows:

[‡] Dept. of Telecommunications and Artificial Intelligence, Faculty of Electrical Engineering and Informatics, Budapest University of Technology and Economics, Budapest, Hungary
[&] IIoT & AI Division, AITIA International Inc., Budapest, Hungary
corr.: attila.franko@vik.bme.hu

DOI: 10.36244/ICJ.2026.2.11

1) Local information exchange: It recognizes that automation and IIoT systems often come with specific requirements, resulting in ”local” information exchanges. In such settings, systems are in close proximity to each other, forming a closed System of Systems architecture. Consequently, Arrowhead emphasizes the importance of addressing the ”closedness” of industrial applications. It operates under the assumption that various application systems and devices can only operate within their own local or closed automation network, known as the ”local Arrowhead network” or ”cloud.” Essentially, an Arrowhead Local Cloud (LC) is a System of Systems, where a node, referred to as a ”system,” within the LC can itself be a complex system, such as a machine, engaging in information exchange with other ”systems” like controllers.

2) Service-Oriented Architecture: Arrowhead advocates the utilization of Service-Oriented Architecture (SOA) within the realm of automation. Within an Arrowhead LC, multiple application systems communicate and provide various functional services to each other. In Arrowhead’s terminology, this is represented as one application system serving as a Service Provider, offering services to another application system, and becoming a Service Consumer. In this context, Arrowhead incorporates all three essential aspects of SOA: (i) loose coupling, (ii) late binding, and (iii) the ability to look up partners at runtime. The definition of a Service outlines the invokable functionality and its interface implementation [9].

Loose coupling Strict, hardwired connections between entities are strictly prohibited.

Late binding Services are dynamically discoverable at runtime by systems that need them rather than being predefined.

Self-containment and modularity Services are self-contained and modular, offering a set of logically cohesive interfaces (e.g., an Application Programming Interface - API) that implement similar functionalities.

Interoperability Systems utilizing different platforms and programming languages should be able to communicate with each other using pre-defined implementations of various services.

Generally, an application system is an execution logic deployed on a hosting device and is part of a cyber-physical system with real-world sensing and actuation capabilities. Arrowhead does not make assumptions about what can constitute an application system; it can be as simple as a single sensor mote or as complex as an entire smart city. The emphasis is on the fact that each application system provides and consumes services from other systems within the LC, with these information exchanges governed by the Arrowhead Core Systems [22].

3) Core Systems: Arrowhead acknowledges that a System of Systems cannot operate entirely in a decentralized manner due to operational and business considerations. To address this, Arrowhead offers several Core Systems designed to control, facilitate, and assist service connections within an LC. There are three mandatory core systems:

Service Registry Service Providers use the Service Registry to register the services they offer and their network addresses, facilitating loose coupling and partner lookup.

Authorization System This system is responsible for managing access permissions, and it issues authorization tokens that can only be decrypted by the designated Service Provider System. These tokens contain information about the consumer system, the allowed service interface, and the duration of access.

Orchestrator The Orchestrator supports the establishment of connection rules, matching service consumers with suitable service providers. It relies on the Orchestration Store, where LC operators can define orchestration rules.

Arrowhead also provides non-mandatory supporting core systems to enhance the central functionalities of application systems. These systems assist with tasks related to event handling, protocol translation, Quality of Service management, and more [11]. Fig. 1 shows the mandatory core systems and supporting core systems for inter-cloud communications implementing the IIoT reference architecture of the Industrial Internet Committee.

B. Time synchronization In Industrial Automation Systems

In recent years, the emergence of CPSoS led us to highly automated manufacturing with autonomous production lines and shop floors, or even lights-out, smart factories that minimize the required human interaction during the process [10], [19]. These systems usually involve mission-and time-critical requirements in order to control and monitor autonomous devices (vehicles, robots, etc.) precisely [16], [20]. In such complex systems, these components usually interact with services outside of the mission-critical part and take information from outside of the system as they are integral parts of a global supply chain network [7], [12].

Achieving real-time or deterministic communication and execution has always been a key factor for time-critical applications. Nowadays, due to the increasing number of cyber-physical-systems-driven production systems in smart industry [21] this topic is more crucial than ever. Generally, these systems often have to meet real-time requirements or rely on accurate time synchronization to coordinate the actions of multiple machines precisely.

In the last few decades, great efforts have been made in order to provide different solutions that can address issues related to real-time requirements such as low latency communication, time synchronicity, scheduling, priority handling (known as traffic shaping), fault tolerance, especially in challenging environemts [14], [18]. Recently, the existing, widely used, and industrially accepted solutions have been standardized and grouped under the umbrella term Time-sensitive Networking (TSN) [3]. TSN is really a set of various standards that aims to be an enabler of such systems, which have to meet one or more aspects of the aforementioned real-time requirements [2]. In this paper, we mainly focus on time synchronization, including the underlying technologies and protocols where necessary.

Precision Time Protocol (PTP) or IEEE 1588 is a protocol that is used to provide time synchronicity between different clocks in a network [4]. PTP is one of the most important protocols of TSN as part of the IEEE 802.1AS standard, which restricts the usage of PTP to a specific profile often referred

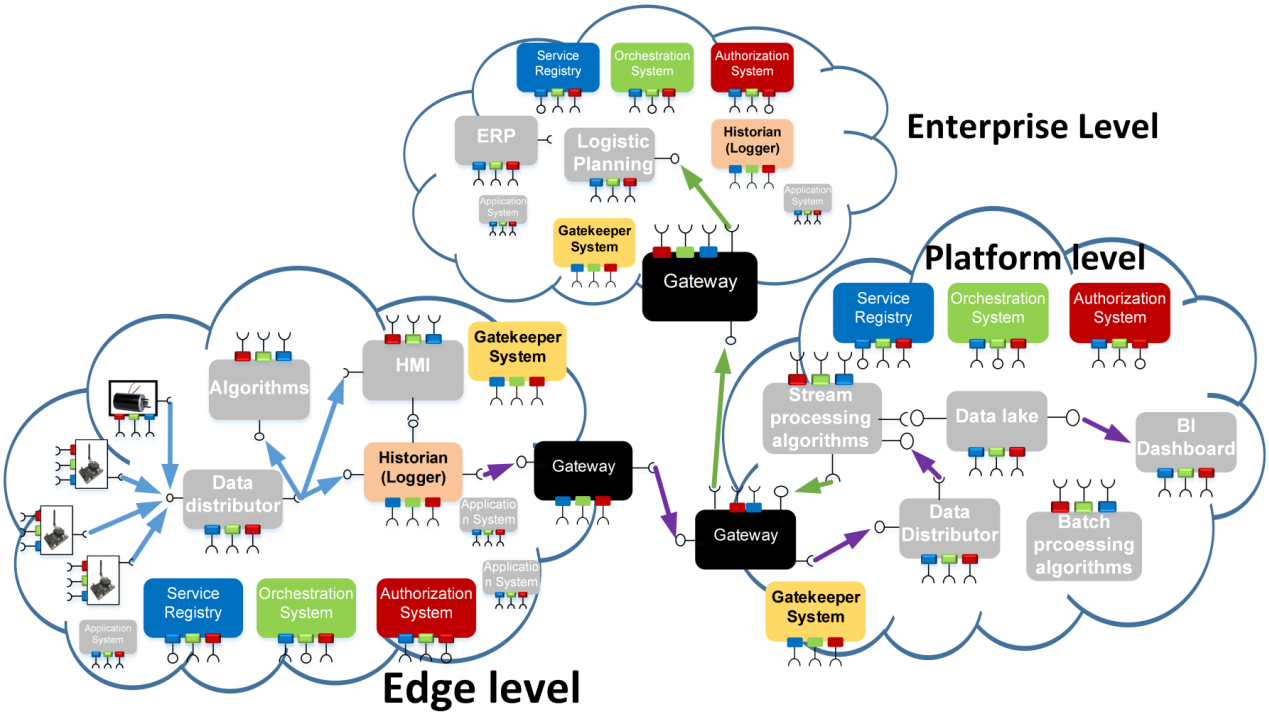

Fig. 1: The IIC (Industry IoT Consortium) IIoT Reference Architecture mapped in Arrowhead presenting its core systems and related systems [11]

to as gPTP (where 'g' stands for general(ized)) defined by the standard [5].

IEEE 1588 strictly defines the clock synchronization model, the PTP entities, the clock types, and the messaging itself. However, we only discuss those parts that are mandatory for presenting the proposed concept. In a computer network, PTP can synchronize different PTP *instances*, which can be parts of one or multiple PTP nodes. Practically, instances are dedicated physical clocks of a Network Interface Card (NIC), while a node can be a server machine or a switch. A network to be synchronized does not require a consisting of only PTP nodes. Moreover, switches do not even have to be PTP-capable ones; in practical cases, they are usually demanded, while in TSN, they are mandatory.

To achieve synchronization, a grandmaster (GM) clock is selected from all PTP instances and will be the reference for all clocks in the network. PTP allows different synchronization methods (protocols) to be used: End-to-end (E2E) and peer-to-peer (P2P) over multiple hops, but each case can be simplified to a master-slave synchronization of two neighboring (i.e., they are connected directly with a link) instances as shown in Figure 2. The core method involves several messages that include the timestamp of receiving and transmitting a specific message; the slave can calculate the link delay and the time offset between the two instances – if the offset is smaller than a certain value, then only the frequency of the clock is trimmed. At the application level, a clock instance can be accessed directly or via the operating system (OS). Thus, any application is able to use any precisely synchronized clock of the node.

Since one PTP node can be a part of multiple networks, thus PTP instances of a node can synchronize to different clocks as well: a collection of instances synchronized to the same clock (and part of the same network) is a clock domain. It is often desired in industrial environments that one node can use timing from different clock domains for its operations. Moreover, according to (the current status of) IEC/IEEE 60802 TSN Profile for Industrial Automation, the usage of more than one clock domain is the standard in industrial use-cases in order to use Working Clock and Global Time domains and also for hot-standby operations [13]. Our proposed solution also supports the usage of multiple clock domains to be aligned with the industrial requirements.

[6] provides a comprehensive survey on Time-Sensitive Networking within the automotive domain, identifying key research opportunities in the integration of deterministic communication with higher-level software architectures. Our work extends this vision by proposing a service-oriented bridge that enables the dynamic orchestration of these timing-critical features within the Eclipse Arrowhead framework, a layer that is often treated as a static configuration in current automotive TSN implementations.

To provide a structured overview of the current landscape, Table I compares existing and prevalent industrial synchronization approaches based on the taxonomy and requirements identified in the literature, specifically focusing on the trade-offs discussed by [17]. It is worth noting that they are not synchronization protocols per se, but architectures and frameworks

TABLE I
COMPARSION OF COMMUNICATION ARCHITECTURES FOR SYNCHRONIZATION IN THE IIOT LANDSCAPE [17], [6], [8]

	SDN-TSN	OPC UA PubSub	FIWARE
Layer Focus	Data Link (L2)	Middleware (L4-L7)	Application (SOA)
Synchronization Precision	Nanosecond	Micro/Nanosecond	Millisecond
Time Source	HW Grandmaster	HW/SW Hybrid	Network (NTP)
Determinism	Gating (TAS)	Priority-based	Best-effort
Coupling	Very Tight	Tight (Static)	Loose
Configuration Phase	Design-time	Semi-static	Runtime / Dynamic

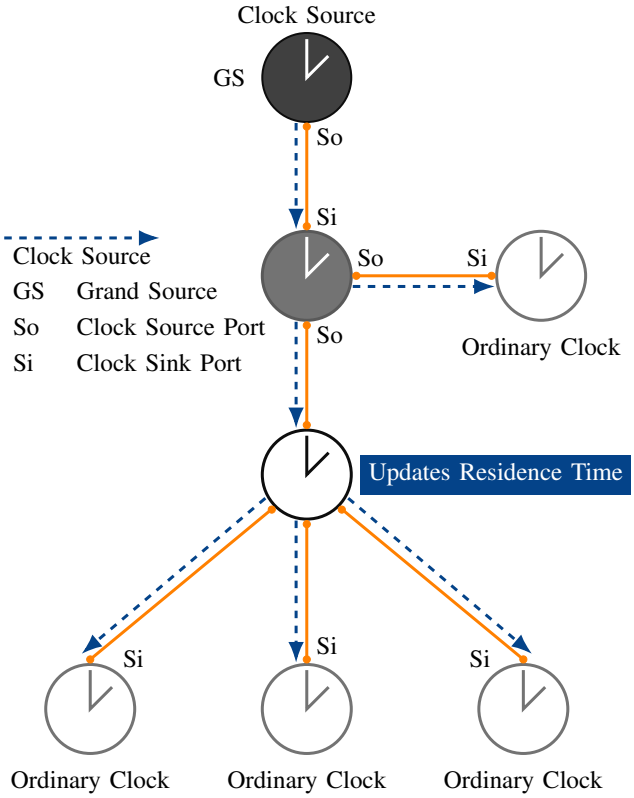

Fig. 2: Different types of PTP clocks and how shared time getting distributed – based on [1]

that use PTP or NTP for achieving synchronicity. The analysis covers a spectrum from low-level network protocols to high-level cloud-based IoT frameworks. This comparison highlights a significant r esearch g ap: w hile h ardware-centric solutions like SDN-based TSN [17] offer nanosecond precision, they often impose rigid, design-time configurations. T hese solutions utilize PTP as the underlying protocol that implements synchronization, so they differ in the level of abstraction and, most importantly, in their focus. Conversely, service-oriented frameworks like FIWARE [8] prioritize flexibility a nd loose coupling but fail to provide the deterministic timing necessary for mission-critical industrial operations, as it uses NTP, which provides only millisecond precision. By evaluating these dimensions—precision, determinism, and coupling strategy- it becomes evident that a middleware-independent, service-level synchronization bridge is required to maintain the dynamic nature of System-of-Systems architectures without sacrificing real-time performance.

III. SERVICE CONSUMPTION IN A SYNCHRONIZED MANNER WITHIN ARROWHEAD

A. Requirements and motivation

Generally, IIoT and SoS frameworks and platforms offer different functions to enhance and ease the design and implementation of microservice-based solutions. Usually, these tools are promoted as major enablers for time-critical applications; however, in most cases, only real-time response times are targeted, which are achieved as per service response time and fast communication to minimize the delay. Nonetheless, since synchronicity requires special hardware resources, including end devices and network devices as well, it is not addressed on a service basis but on a device basis (such as Network Interface Cards or clock instances).

Arrowhead is unique regarding its approach to interoperability and interchangeability. However, service consumption is a focal point as it is a key element of modern system architecture. In order to enable real-time, deterministic communication between Arrowhead-compliant services – including precise, scheduled execution of different tasks provided by services – the current architecture has to be expanded further; it is worth noting that scheduling here does not mean chaining calls based on a predefined order, as it is done very similarly by supplementary core system Workflow Choreographer [15]. It means only precise execution of a task, i.e., invocation and running a service at the exact time. Nonetheless, in industrial systems, automated, flexible workflow execution is often required to be performed with precise, synchronized timing. Thus, the Choreographer and our proposed solution can jointly make systems meet these requirements.

In [15], authors provide a real use case in order to demonstrate the need for workflow management and apply their solution to that. The use case includes a generalization of a common problem of moving and packing objects on shop floors or production lines. This task requires not only workflow management but precise, synchronized execution as well, since in a real environment, each step has to be performed just in time, especially if we extend the number of actors that have to execute actions cooperatively. Consequently, all devices involved in a time-critical process have to be synchronized to enable precise, scheduled execution.

A fundamental issue is the control plane being defined on the service level, including the core systems of the local cloud, while synchronicity exists per device – or more precisely, per instance – level. Therefore, the main issue here is not achieving synchronicity between services within the Local Cloud, as it would highly restrict service abstractions, but using the already existing time synchronicity between devices that run the services. It is crucial to avoid involving concrete devices in the design of systems. Therefore, we have to transform the timing capabilities to the service level.

Another issue raised by industrial use cases is the problem of multiple clock domains. A naive way to achieve synchronicity within a local cloud could be the enforcement of synchronicity between all hosts that provide or consume a service in a local cloud. However, this approach assumes using only one clock domain as it implicitly requires the node to be synchronized with the same grandmaster. Our proposed solution follows a different concept that does not restrict the number of used clock domains. Thus, the synchronicity of the instances does not have to be global in the local cloud. Instead, Arrowhead is fully aware of the existence of different clock domains within the local cloud and is able to take this information into consideration during the orchestration process.

B. Arrowhead TimeSync Instance and Time Sync Relay

The bridging of synchronicity to the control plane of Arrowhead is handled by a Time Synchronization Relay (TSR) supporting core system, as it can be seen in Figure 4 and a special consumer service profile called Arrowhead TimeSync Instance (ATI) as seen in Figure 3.

The ATI is a special service in systems that enables synchronization for certain services within a system. It operates as a scheduler proxy for any service that runs on the same PTP instance as the ATI. Since the synchronization is handled by the PTP protocol itself, the ATI has access to a physical clock that is synchronized to one or more other PTP instances in the Local Cloud. Generally, there are no guarantees that a service can use any physical clock of the machine that runs it – e.g., if the service runs in an isolated environment such as a container, it might not be able to use the clock. However, an ATI must always have access to the desired PTP instance. It is worth emphasizing that since devices can have multiple PTP instances that are synchronized to different clock domains, it is possible to run multiple ATI-s on the same device to schedule tasks by different clocks or have dedicated ATI-s for different services (see below).

With that approach, service invocation can be planned beforehand since ATI can schedule it precisely: we will call this service of an ATI a Scheduled Execution (SCE). It is important to emphasize that a service can be invoked directly via Arrowhead as a regular service, even if it offers real-time execution, which can be soft real-time or hard real-time. These characteristics of ATI depend upon the implementation of ATI, as calling the desired service might cause delays during invocation. To meet soft real-time requirements, ATI can be realized as a general software that can call other services locally via protocols using shared memory or inter-process calls (IPC). In order to provide hard real-time execution with practically zero delay, ATI must be implemented as an integral part of any real-time-capable service (using it as a dedicated library). In such a way, the service has its own dedicated ATI, so there might be different ATI-s using the same PTP instance. It is worth noting that the soft real-time approaches enable legacy services to offer real-time execution as they don't have to be bundled with a dedicated ATI.

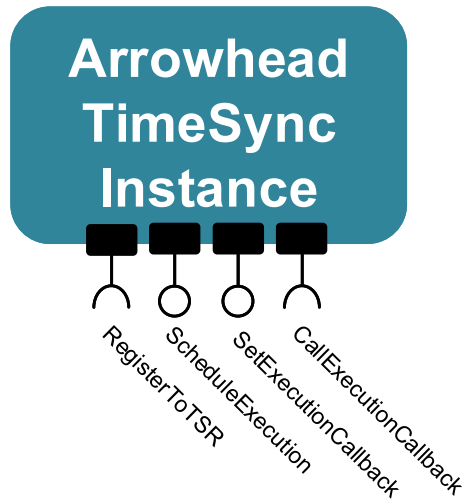

Fig. 3: Overview of Arrowhead TimeSync Instance

Time Sync Relay provides services for systems to ensure time-critical execution. Services of a system can register themselves at any time to advertise their SE capability. However, these services might offer non-scheduled execution as well – it depends on the service itself. When a service registers itself to the TSR, it has two options: (i) it must declare its ATI explicitly, thus enabling legacy applications to use external ATI for soft real-time scheduling or (ii) if they have internal ATI for hard real-time execution, then it must be determined automatically (e.g., domain name matching, etc.) so if an execution has to be scheduled for the service, the TSR knows which ATI should be addressed with the scheduling request (SCR). This implies that a SE-capable service must always know the identity of its ATI during its lifecycle. It is worth noting that despite the implied local hard coupling, this approach does not violate the overall philosophy and composability of the Arrowhead. From an orchestration perspective, the ATI is transparent; only the running service must know its ATI since they are coupled by the fact of using the same PTP instance.

Nevertheless, this is still not sufficient f or S E s ince it does not provide any guarantees of two services being synchronized. It technically means that their corresponding PTP instances are synchronized to the same grandmaster, i.e., they are in the same PTP domain. The root of the problem is that two instances only know the grandmaster, to which they are synchronized, and real-time services usually do not require additional knowledge as they run on the same physical network by design. It is still true; however, during the orchestration

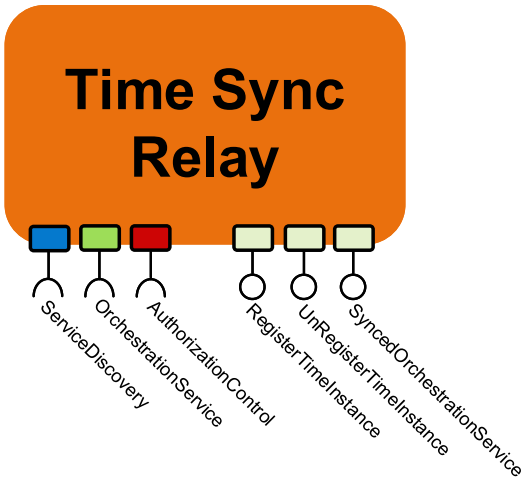

Fig. 4: Overview of Time Sync Relay

process, it is not revealed if a provider service runs on the same physical network as the consumer. Due to this, the orchestration process must offer services that run on the same physical network as the consumer. To address this issue, during registration, an ATI must always declare the identifier o f the PTP domain in which it is located. Thus, two services can determine whether they are synchronized to the same GM. An overview of the concept of ATI-s can be found in Figure 5.

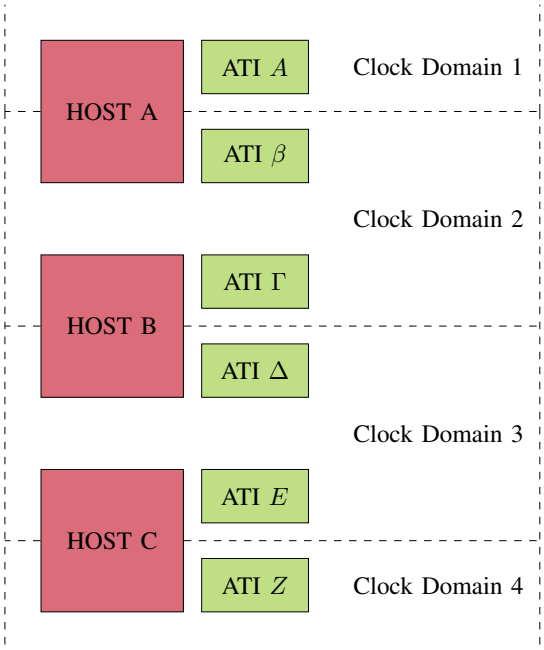

Fig. 5: Multiple hosts providing services while synchronized to different clock domains by multiple ATI-s

IV. Case study: Scheduled Execution and Synchronized Service Orchestration

In this section, we present an example of a Scheduled Execution (SE) and a Synchronized Service Orchestration (SSO), too, as they can be seen in Figure 6. This case shows a TSR and two real-time services with the corresponding ATIs in one LC. The steps of registration of a real-time service will be shown, but mandatory functions of an LC, such as registering a service to the Service Registry and setting authorization rules, will not. Moreover, the synchronization via the PTP process is also omitted as it is well-defined in the corresponding standard [4], so all PTP instances are assumed to be already synchronized to a common GM.

The example shows an abstract SE and SSO. However, it is the same for any configuration, including the production line use case mentioned in subsection III-A. It goes as follows:

1) The ATI of the ”to-be-provider” service registers itself to the TSR and also declares the PTP Domain ID to which it is synchronized.
2) The ”to-be-consumer” service requests regular orchestration with the desired flags and properties from the orchestrator.
3) The orchestrator returns the list of available services, such as for regular orchestration.
4) The ”to-be-consumer” service requests special orchestration for real-time services (that share a common clock domain) from the TSR based on the orchestrator’s response. The service calls the ”SyncedOrchestrationService“ endpoint of the TSR, passing the service description of the service to be consumed.
5) The TSR finds the corresponding ATI based on the request (service, clock domain) by applying different policies (e.g., URL matching) and methods and returns them to the caller. Now, it is possible for the caller to initiate an SSO directly with the provider service.
6) To initiate an SE, the consumer sends an SE request to the ATI, including the service definition of the service to be scheduled and other parameters (optionally the callback URL) by calling the ”ScheduleExecution“ service.
7) The ATI schedules the given call as it is defined in the request and then invokes the target service at the scheduled time.
8) Optionally, the provider service can return data (depending on its functions and service definition) to ATI.
9) The ATI forwards the return data to the consumer system via the preconfigured callback service ”CallExecutionCallback“.

V. Conclusions

Due to the ever-growing complexity of industrial systems, design principles address composability and interoperability issues between the components of a system. Arrowhead framework is specifically built to support these principles, enabling all of its users to work with a common and unified approach. However, these principles operate on a service basis, whereas specific industrial requirements, such as real-time, deterministic communications, function on a different level. This discrepancy makes it impossible to meet such demands solely through service abstractions.

This paper presented an approach to extend the current capabilities of Arrowhead to bridge time synchronicity achieved by

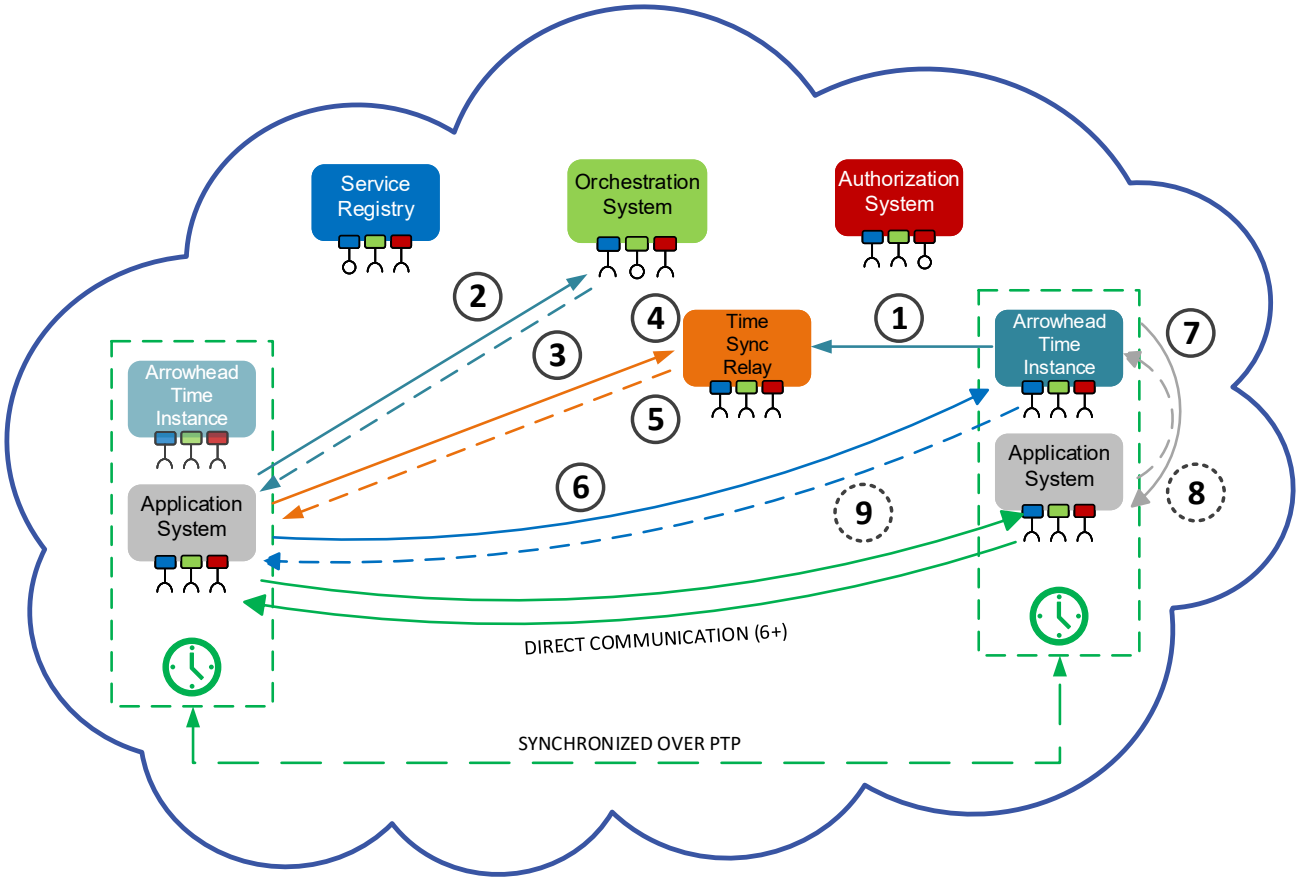

Fig. 6: Overview of an SE and SSO in a Local Cloud

PTP to service level, enabling loosely coupled CPSoS-es with precise, real-time communication and execution requirements. The overall solution extends the orchestrating capability of the Arrowhead with the help of Time Synchronization Relays to enable matchmaking that takes the time synchronicity of the services into account. Moreover, the Arrowhead Timesync Instance was introduced to provide a convenient way for proxying sync information to service level and also scheduled requests to services. Compared to the synchronization architectures summarized in Table I, the proposed approach combines deterministic PTP-based synchronization with the flexibility of a service-oriented architecture, eliminating the need for rigid design-time or semi-static configurations while preserving loose coupling. Furthermore, by supporting run-time orchestration and multiple clock domains, the proposed solution enables dynamic and interoperable industrial CPSoS deployments without compromising real-time communication and execution requirements.

REFERENCES

[1] Aos-cx 10.14.xxxx fundamentals guide, chapter 9: Ptp. https://arubanetworking.hpe.com/techdocs/AOS-CX/10.14/PDF/fundamentals_8100-83xx-9300-10000.pdf. [Online; accessed 23-April-2026].

[2] A real-time ethernet solution available in an open ieee 802.1 standard. https://tsn-network.com. [Online; accessed 1-Februar-2024].

[3] A deterministic ethernet solution based on time sensitive networking (tsn). https://www.ddc-web.com/resources/FileManager/dbi/Whitepapers/TSN%20White%20Paper.pdf, 2020. [Online; accessed 13-April-2026].

[4] IEEE standard for a precision clock synchronization protocol for networked measurement and control systems. *IEEE Std 1588-2019 (Revision of IEEE Std 1588-2008), pages 1–499, 2020.*

[5] IEEE standard for local and metropolitan area networks–timing and synchronization for time-sensitive applications. *IEEE Std 802.1AS-2020 (Revision of IEEE Std 802.1AS-2011)*, pages 1–421, 2020.

[6] Mohammad Ashjaei, Lucia Lo Bello, Masoud Daneshtalab, Gaetano Patti, Sergio Saponara, and Saad Mubeen. Time-sensitive networking in automotive embedded systems: State of the art and research opportunities. *Journal of Systems Architecture*, 117:102137, 2021.

[7] Pradeep Bedi, Kamal Upreti, Anand Singh Rajawat, Rabindra Nath Shaw, and Ankush Ghosh. Impact analysis of industry 4.0 on realtime smart production planning and supply chain management. In *2021 IEEE 4th International Conference on Computing, Power and Communication Technologies (GUCON)*, pages 1–6, 2021.

[8] Flavio Cirillo, Gürkan Solmaz, Everton Luís Berz, Martin Bauer, Bin Cheng, and Ernő Kovacs. A standard-based open source iot platform: Fiware. *IEEE Internet of Things Magazine*, 2(3):12–18, 2019.

[9] Jerker Delsing, editor. IoT Automation. CRC Press, February 2017. [10] Isa Gerekli, Tarık Celik, and Ibrahim Bozkurt. Industry 4.0 and smart production. *TEM Journal*, 10:799–805, 05 2021.

[11] Csaba Hegedus, Pál Varga, and Attila Frankó. Secure and trusted inter-cloud communications in the arrowhead framework. In *2018 IEEE Industrial Cyber-Physical Systems (ICPS)*, pages 755–760, 2018.

[12] Jonny Herwan, Seisuke Kano, Ryabov Oleg, Hiroyuki Sawada, and Nagayoshi Kasashima. Cyber-physical system architecture for machining production line. In *2018 IEEE Industrial Cyber-Physical Systems (ICPS)*, pages 387–391, 2018.

[13] IEC/IEEE. Iec/ieee 60802 time-sensitive networking profile for industrial automation. https://www.ieee802.org/1/files/private/60802-drafts/d2/60802-d2-4.pdf, 2024. [Online; accessed 17-June-2024].

[14] Kyoung-Don Kang and Sang H. Son. Real-time data services for cyber physical systems. In *2008 The 28th International Conference on Distributed Computing Systems Workshops*, pages 483–488, 2008.

[15] Dániel Kozma, Pál Varga, and Kristóf Szabó. Achieving flexible digital production with the arrowhead workflow choreographer. In *IECON 2020 The 46th Annual Conference of the IEEE Industrial Electronics Society*, pages 4503–4510, 2020.

[16] Vuk Lesi, Zivana Jakovljevic, and Miroslav Pajic. Synchronization of distributed controllers in cyber-physical systems. In *2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 710–717, 2019.

[17] Lucia Lo Bello and Wilfried Steiner. A perspective on ieee time-sensitive networking for industrial communication and automation systems. *Proceedings of the IEEE*, 107(6):1094–1120, 2019.

[18] Jannis Ohms, Martin Böhm, and Diederich Wermser. Concept of a tsn to real-time wireless gateway in the context of 5g urllc. In *2020 8th International Conference on Wireless Networks and Mobile Communications (WINCOM)*, pages 1–6, 2020.

[19] Peter A. Okeme, Anastasiia D. Skakun, and Alexander R. Muzalevskii. Transformation of factory to smart factory. In *2021 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (ElConRus)*, pages 1499–1503, 2021.

[20] Miguel Saez, Francisco P. Maturana, Kira Barton, and Dawn M. Tilbury. Real-time manufacturing machine and system performance monitoring using internet of things. *IEEE Transactions on Automation Science and Engineering*, 15(4):1735–1748, 2018.

[21] Devarpita Sinha and Rajarshi Roy. Reviewing cyber-physical system as a part of smart factory in industry 4.0. *IEEE Engineering Management Review*, 48(2):103–117, 2020.

[22] Pal Varga, Fredrik Blomstedt, Luis Lino Ferreira, Jens Eliasson, Mats Johansson, Jerker Delsing, and Iker Martínez de Soria. Making system of systems interoperable – the core components of the arrowhead framework. *Journal of Network and Computer Applications*, 81:85–95, 2017.

ACKNOWLEDGMENT

The research leading to these results is funded by the EU Chips-JU organization under grant agreement 101112089, within the project AIMS5.0 and from the partners' national programs and funding authorities. Part of this work is created within the Arrowhead fPVN project, supported by the Chips-JU and its members, as well as by national funding authorities from involved countries under grant agreement no. 101111977.

Attila E. Frankó received his M.Sc. degree in 2019 in Electrical Engineering and his Ph.D. degree in 2024 from the Budapest University of Technology and Economics (BME), Hungary. He is currently working at the Department of Telecommunications and Artificial Intelligence (BME) as an assistant professor and also as a senior researcher at AITIA International Zrt. His research interests include data science, artificial intelligence, Industrial Internet of Things (IIOT), indoor positioning systems, anomaly detection in telecomunniation networks.

G. László Hollósi received his M.Sc. degree in electrical engineering with honors from the Budapest University of Technology and Economics (BME) in 2009, an engineering-economics degree from the Budapest Business School in 2014, and his Ph.D. degree from BME in 2025. From 2008 to 2009, he worked as a test automation engineer at Ericsson Hungary Kft. In 2009, he joined the Department of Telecommunications and Media Informatics (currently the Department of Telecommunications and Artificial Intelligence) at BME, where he currently serves as an assistant professor. Since 2026, he has held the position of chairman of the artificial intelligence section of the Scientific Association for Infocommunications. His research interests include data analysis, artificial intelligence, image processing, complex networks, indoor positioning, and time synchronization in packet-switched networks. He is a two-time recipient of the Pollák–Virág Award and serves as a regular reviewer for several journals, including IEEE Transactions on Measurement and Instrumentation, IEEE Transactions on Service and Network Management, IEEE Access, and Infocommunications Journal.

Dániel Ficzere is an Assistant Lecturer at the Budapest University of Technology and Economics (BME), Department of Telecommunications and Artificial Intelligence. He actively conducts research and publishes in the fields of telecommunications and artificial intelligence, specializing in generative AI for network operations, document processing, network science, and sports analytics. Alongside his academic work, he serves as the Secretary of the Artificial Intelligence Section of the Scientific Association for Infocommunications (HTE).

Pal Varga received his Ph.D. degree from the Budapest University of Technology and Economics, Hungary. He is currently an Associate Professor and head of Department of Telecommunications and Artificial Intelligence at Budapest University of Technology and Economics. His main research interests include communication systems, Cyber-Physical Systems and Industrial Internet of Things, network traffic analysis, end-to-end QoS and SLA issues – for which he is keen to apply hardware acceleration and artificial intelligence, machine learning techniques as well. Besides being a member of HTE, he is a senior member of IEEE, where he is active both in the IEEE ComSoc (Communication Society) and IEEE IES (Industrial Electronics Society) communities. He is Editorial Board member of the Sensors (MDPI) and Electronics (MDPI) journals, and the Editor-in-Chief of the Infocommunications Journal.