Dynamic Execution of Engineering Processes in Cyber-Physical Systems of Systems Toolchains

Federico Montori, *Member, IEEE*, Marek S. Tatara, *Member, IEEE*, and Pál Varga, *Senior Member, IEEE*

***Abstract*— Engineering tools support the process of creating, operating, maintaining, and evolving systems throughout their lifecycle. Toolchains are sequences of tools that build on each others' output during this procedure. The complete chain of tools itself may not even be recognized by the humans who utilize them, people may just recognize the right tool being used at the right place in time. Modern engineering processes, however, do not value such ad-hoc choice of tooling, because of their uncontrolled nature. Building upon the Extended Automation Engineering Model defined by the IEC 81346 standard, this paper proposes to automate the toolchain building and execution process for Cyber-Physical System of Systems (CPSoS), utilizing key principles of the Eclipse Arrowhead framework. The proposed toolchain automation solution addresses issues such as tool interoperability, interaction, automation, and dynamic choreography. The feasibility of this set of integrated concepts is validated through an Arrowhead-based toolchain choreography demonstration.**

***Note to Practitioners*—The paper discusses approaches to the automated execution of various industry-related processes. As the processes are becoming more complex and involve numerous systems which have to be orchestrated, a simple and preprogrammed workflow is not enough anymore. Therefore, building on top of the principles of the Eclipse Arrowhead framework, an adequate model of toolchains, allowing for their automated execution, is proposed. Different approaches to supervision of toolchain execution are discussed showing the benefits of reaching higher automation levels. Further, four adoption levels are introduced, which are a measure of the toolchain automation progress. Finally, a simplified demonstrator is shown and steps to elevate it to higher adoption levels are highlighted. To ensure that the approach is industry-oriented, several examples of how the proposed methodology can be used in the industrial context are discussed.**

***Index Terms*— Toolchains, industry 4.0, interoperability, engineering process, IoT automation, service oriented architecture, service orchestration, service choreography.**

Manuscript received 4 December 2023; revised 26 January 2024; accepted 31 January 2024. This article was recommended for publication by Associate Editor V. Villani and Editor B. Vogel-Heuser upon evaluation of the reviewers' comments. This work was supported in part the Electronic Components and Systems for European Leadership (ECSEL) under Grant 826452 and in part by the European Union Horizon 2020 research and innovation programme. *(Federico Montori, Marek S. Tatara, and Pál Varga contributed equally to this work.) (Corresponding author: Federico Montori.)*

Federico Montori is with the Department of Computer Science and Engineering, University of Bologna, 40126 Bologna, Italy (e-mail: federico.montori2@unibo.it).

Marek S. Tatara is with the Faculty of Electronics, Telecommunications and Informatics, Gdańsk University of Technology, 80-233 Gdańsk, Poland, and also with DAC.digital, 80-233 Gdańsk, Poland.

Pál Varga is with the Department of Telecommunications and Media Informatics, Budapest University of Technology and Economics, 1111 Budapest, Hungary.

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TASE.2024.3362132.

Digital Object Identifier 10.1109/TASE.2024.3362132

I. Introduction

HETEROGENEOUS, dynamically changing System of Systems (SoS) must work in an effective and sustainable way. In order to reach this, we must start with proper planning – although the engineering procedure does not end with maintenance but continues in evolution circles. The generic engineering process needs a closer look here and, inevitably, the tools used in the process steps. Unavailable or inappropriate tools slow down the given step in the process, which then becomes a bottleneck – eventually for the whole supply chain. Choosing and using the appropriate tool at the given step of the engineering procedure should be planned well. Nevertheless, just like the dynamic reconfiguration of SoS on the fly, the engineering procedure itself requires solutions for choosing or changing tools at certain steps. The motivation for *dynamic* toolchains – the sequence of engineering tools – traces back to the information sharing requirements of Industry 4.0. As the elements of distributed SoS change over time, interoperability becomes a natural requirement, as well. This goes together with integrability, since legacy and new systems need to work together. When the tools form a toolchain, the output of one becomes the input of the next – in an ideal situation, without any human interference or data format manipulation. This semantic interoperability by itself is a huge challenge – offering a promising gain of complete toolchain automation. The goal is exactly this toolchain automation, although with dynamically changeable tools controlled by a choreography mechanism to complete the engineering procedure.

There are various, overwhelmingly complex theoretical and practical problems arising when addressing this domain set. Our approach towards the solution is to apply the same concepts for tool interoperability as for SoS interoperability. The distributed nature of SoS requires the exact opposite of monolith thinking: implementations based on Service-Oriented Architectures (SOA) or microservices, which are very similar service-based concepts.[1] Many of these issues are inherently addressed by the Eclipse Arrowhead framework [1], which is utilized when proposing toolchain-related solutions. The initial step towards automation is the modeling of the toolchain – for which the IEC 81346 [2] standard can be extended to fit an evolving lifecycle-type reality of the industry. In case there are more tools to choose from at each lifecycle step, the SOA approach helps in the actual matchmaking of interacting tools.

[1]Nota Bene: in this paper we use and refer to the SOA concept because of its systems engineering angle, but the ideas utilized here apply to the microservices domain as well.

The tool input/output format mismatch problem still needs to be solved at each lifecycle step. The orchestration should make schematic matching possible – if not by design then by summoning semantic adaptors as it is done within Eclipse Arrowhead in facing interoperability issues [3], [4].

In this paper, we suggest using Arrowhead Orchestration and Choreography to step through the lifecycle with dynamically chosen tools. Namely, we suggest addressing the issue of dynamic tool selection through the Orchestration of Eclipse Arrowhead. Moreover, the workflow execution across the toolchain is suggested to be controlled by the Choreographer of Eclipse Arrowhead. This ensures to keep track of the process and to push it to the next engineering step according to a toolchain recipe. The concepts of orchestration and choreography are widespread in service computing, sometimes addressed as duals; in this work, we will make orthogonal use of them, following the theory for which they should coexist [5]. As greenfield Arrowhead-compatible systems and tools are rare, adoption levels are also defined here – so users can decide at what level can/will they adopt the Arrowhead concepts at the toolchain level.

It is worth mentioning that the solution presented in this paper specifically refers to a subset of Industry 4.0 use cases, as it is primarily targeting the Arrowhead Community. Engineering Processes within such an environment are CPSoS driven by software organized in SOA, thus, although heterogeneous, our solution does not aim to be all-encompassing. On the other hand, though, we estimate our proposal to have a significant resonance, as the Arrowhead Community (see the beginning of Section III) is widespread. The contributions of this paper are threefold. It provides

1) an overview of Engineering Processes and shows the gap for toolchain automation;
2) a solution by applying the SOA-based Arrowhead Workflow Choreography idea to the SoS toolchain domain;
3) the definition and usage of Adoption Levels for toolchain integration.

These contributions are connected in a linear way: the gaps are identified, the automation can be solved through applying Arrowhead Choreography, which then becomes the highest level of adoption for the ecosystem stakeholders. Given the definitions in this paper, stakeholders can clearly identify which Arrowhead adoption level they need, and how to approach its application to their case. Furthermore – as it turns out – the highest level of Arrowhead adaptation is when the process workflow is conducted through an automatized execution, enabled by the Choreographer. The proposed set of solutions is validated through a demonstrator.

The particular challenges related to Cyber-Physical Systems-of-Systems the Authors attempt to address in this paper are the following:

- Seamless interoperability [6] between various systems in SoS
- Automated and coordinated execution of toolchains [7] in service-oriented architecture
- Quantification of the maturity level allowing for further toolchains automation

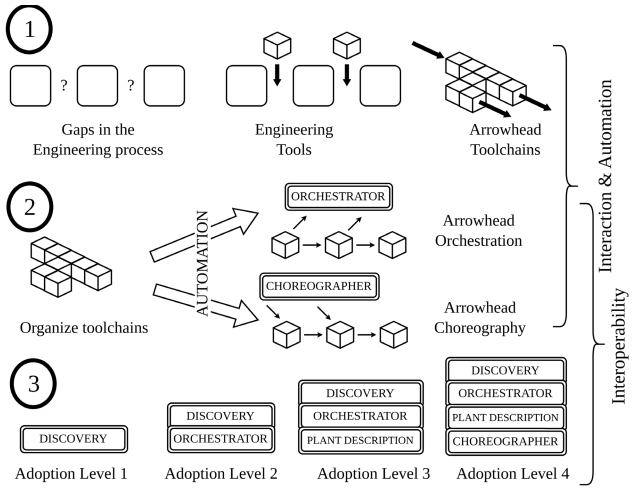


Fig. 1. The three main groups of objectives of this paper.

The Cyber-Physical context is included here by taking into account the entire lifecycle of the engineering process, as well as the effect of real-world data (including human-in-the-loop) on the execution of particular workflows. In order to show how these challenges can be addressed in the industrial context, four use cases will be used to visualize how the proposed methodology can be applied, namely: sensor reconfiguration for haulers in the milk collection and delivery process, robotic operation supervision, automotive assembly plant, and sensor onboarding.

Our contributions are also illustrated in Figure 1, where the logic behind each conceptual step is clarified through images, focusing on how each contribution uses the output of the previous one. In order to support structured understanding, each contribution got a dedicated section, as specified in the paper organization that follows: Section II gives a background overview of the work related to engineering processes; Section III details the proposed toolchain model and its elements (accomplishes contribution 1); Section IV describes the ways of supervising the toolchain execution process (accomplishes contribution 2); Section V puts the presented approach in the perspective of Eclipse Arrowhead, showing its practical feasibility through framework adoption levels, accomplishing contribution 3; Section VI demonstrates the contributions through working examples; Section VII concludes the paper.

II. Related Work

The ultimate aim of the engineering processes for systems is to have a structured way of handling their birth, integration, and evolution throughout the lifecycle. The IEC 81346 standard [2] provides a widely accepted designation method for a common, structural description of systems from different aspects. These aspects describe the system either from the product aspect, the functional aspect, or the location aspect, and these structures can be visualized in parallel as well. The IEC 81346 standard also gives examples of domain-specific usage. The methods described in this paper build upon

the engineering process defined by IEC 81346, which has been further extended within the European ECSEL project "Arrowhead Tools.[2]" The basic idea of the toolchain model has been described in [8], providing definitions of tools and toolchains in the Arrowhead perspective. The phases and the ontology of the Arrowhead Engineering Process (Arrowhead-EP) model are described by [9]. This was extended by a feasibility and use-case study on large-scale IoT ecosystems in [10], and further detailed through multi-stakeholder use-cases in [9]. The latter deals in depth with the Arrowhead-EP and gives guidelines on how to identify the phases in legacy engineering processes, however, it does not give the means to actuate a proper orchestration nor a choreography on top of such a framework. For the sake of integrity, we briefly recall the toolchain model in the current paper (Section III). A methodology to design business process models in BPMN according to the ANSI/ISA-95 for is presented in [11], although – as many similar works – merely focuses on manufacturing. Such process description mappings are used by all the workflow management and execution approaches presented in this paper. A current review of workflow management in smart manufacturing is presented in [12].

Model-Based System Engineering (MBSE) has a key role in the lifecycle of modern SoS. A systematic review of MBSE approaches is provided in [13]. A good summary of current tools and toolchains in the Cyber-Physical SoS (CPSoS) domain can be found in [14]. A good example of integrating MBSE to the toolchain for software development is [15]. This approach can be expanded for CPSoS – by also having feedback from the live system changes to the model changes – is described by [16]. There are various concrete toolchains used in software development. Without going into a comparative study, the widely used DevOps approach with CI/CD (Continuous Integration / Continuous Delivery) toolchaining is gaining worldwide acceptance [17], and its combination with the SOA approach is even applied for cloud software development [18]. This best practice has been suggested for application for CPSoS in [19]. It is clear that common information formats help system interoperability. Unfortunately, but naturally, many ontologies, object models, and communication semantics exist, hence neither object connection in plant descriptions nor data exchange among systems are completely straightforward. One prominent example of describing production plants through hierarchical object information is the IEC62424 standard [20], also known as CAEX (Computer Aided Engineering Exchange). The ISO15926 standard [21] aims at data integration, sharing, and exchange mainly for industrial automation systems and process plants – including a generic data model. Recently, the OPC-UA has been emerging due to its capabilities to fit the SOA approach, its wide extensions to TCP/IP-based application protocols, and its APIs to various programming languages [22].

Model-driven development (MDD) and SOA have already been used in various domains, such as the Ambient Assisted Living (AAL) area. The EU IST project MPOWER applied the SOA and MDD principles in various AAL scenarios [23], however, their focus remained merely on the development part of the toolchain. The SPIRIT framework [24] uses an SOA-based MBSE approach for toolchain handling, but it focuses only 3 engineering process phases, all in design time. There is also a recently presented, unified MBSE design platform framework [25] that supports modeling information exchange between different engineering tools. The aim here is similar to one of ours, namely to allow toolchain input-output information seamlessly to flow among engineering tools, however, it specifically targets information exchange among *modeling tools*, which limits its direct usability in current toolchains. The concept of Domain Specific Systems Engineering (DSSE) is described and applied as an MBSE approach for Cyber-Physical Systems by the authors of [26]. The solution is thorough in the way it is describing Process and Architectural models and their connections, but on the other hand, it does not suggest any tool discovery or execution methods.

Specifically for manufacturing production, the authors of [27] survey the motivations and approaches on why and how Manufacturing Engineering Systems (MES) evolve for Industry4.0, whereas [28] provides an overview for MES-integrated digital twin frameworks. Smart factory reconfiguration for healthcare [29] and Edge-Cloud collaborative manufacturing [30] are further concrete examples of using SOA and microservice approaches at the MES and the manufacturing edge-cloud infrastructures, respectively.

Altogether, we found only these few traces in scientific literature of automated discovery of engineering tools. When it comes to CPSoS – and not merely production, or office workflow or cloud software engineering –, parts of the problem-space are addressed by different papers, but not as a whole. Regarding automatic information passing among tools throughout such a complete toolchain, we failed to find publications outside the Arrowhead Tools ECSEL project. Moreover, the toolchain choreography through the complete engineering process is a completely unaddressed issue for which the current paper is the first one that provides a solution. These issues, namely (i) automated tool discovery (ii) together with automatic information passing among the tools, and (iii) automatized execution of the overall engineering process constitute clear research gaps in the domain – which should be addressed together.

III. The Proposed Toolchain Model

This section defines the main concepts upon which we build our proposal: The Arrowhead Framework, the Arrowhead Engineering Process, and Arrowhead Tools and Toolchains. These are widely adopted within the Arrowhead Community, which is the main target for our solution. The community started with the Arrowhead project,[3] which kicked off in 2013 and was carried out by more than 80 EU partners (one of the biggest EU projects at its time). Later on, many other projects continued on the same path, such as Productive 4.0, FAR-EDGE, building up a dynamic, heterogeneous, and vast

[2]https://tools.arrowhead.eu/home/

[3]https://cordis.europa.eu/project/id/332987

community. Specifically, the present contribution was a result of the Arrowhead Tools project, which shared the participation of more than 90 EU partners from 18 countries, all bringing in their different use cases to a reference architecture: the Arrowhead Framework. We believe that our solution can be a reference for many use cases outside the Arrowhead community, however, the community is specifically our target.

A. The Arrowhead Framework

The initial goal of the Arrowhead Framework [3] was to address interoperability and integration [31] issues of the Industry 4.0 initiative in a safe and secure manner [32]. The framework follows the principles of SOA, enabling the collaboration of newly built as well as legacy systems, hence complex and dynamically changing Cyber-Physical System of Systems (CPSoS) can be created.

The Arrowhead Framework provides several mandatory and supporting core systems for CPSoS developers and integrators, whereas all other modules are called Application Systems in the Arrowhead nomenclature. These systems provide and/or consume the various application services – as the SOA principles suggest: in a discoverable, late-bound, loosely coupled way. Altogether, these SoS are uniquely defined as Local Clouds, among which secure Inter-Cloud information exchange is also supported [1]. The mandatory core systems are the Orchestration System (mainly service discovery and late binding), Service Registry (so services providers can announce their active services), and Authorisation System (to provide authorization and authentication). The supporting core systems are summarized well in [33] and documented in detail in their Eclipse-approved open source GitHub repository [34].

The most important supporting core system for the current work is the Workflow Choreographer [35], which executes the production recipe by triggering the next step in the process execution. Each of these steps can lead to a separate service Orchestration, so the consumer and provider systems that realize the service could be dynamically connected (loosely coupled, late bound) [36]. This paper proposes that the idea of this SOA-based Workflow Choreography for production floor process execution for Cyber-Physical Systems (CPS) should be mapped for the Engineering Process execution for toolchains.

B. The Arrowhead Engineering Process

Complex SoS engineering needs a standard environment on top of which industrial processes are founded. Such an environment is defined as an *engineering process model*, which has been proposed lately in [9], [10] and to which we will refer to as Arrowhead Engineering Process (Arrowhead-EP). The current version of the Arrowhead-EP features concepts that are in line with earlier similar models, to retain backward compatibility, as well as combining the flexibility and adaptivity that is necessary to meet current industrial requirements. Arrowhead-EP is inspired by ISO/IEC 81346 standard [2], on top of which features that are proper of SOA allow for a more automated, flexible, and yet decoupled flow of information that interconnects the Engineering Process Phases (EPPs) which no longer need a fixed order if the use case demands so. The eight EPPs are depicted in Figure 2; they are ordered and connected via *interfaces*, more in detail, each EPP has an *incoming* (EP-I) and an *outgoing* (EP-O) interface. The term engineering process unit (EPU) can mean any of the three concepts.

C. Arrowhead Tools and Toolchains

Within the scope of the Arrowhead Tools project, one goal is to explicitly address the capabilities of tools in Industry 4.0, as SoS automation scenarios demand more and more of the involvement of supportive toolchains that can work without human intervention. What has also been underlined within such a heterogeneous environment, is that the concept of "tool" is not always clear. Within the scope of the Arrowhead Tools project, we can state that a tool *is* or *has* a piece of software and it is specifically supporting an engineering process that leads to the realization of artifacts. It is important to stress that henceforth we will use the word "tool" to identify "Arrowhead Tools" just as they are defined below, not by its English dictionary definition, even though this definition can be extended to any engineering tool. In short, a tool is a software or a hardware (with adequate software on board) entity/artifact that replaces manual labor throughout the Arrowhead-EP of some artifact, either in its design time or run time, that *improves an already established industrial baseline* by satisfying non-functional requirements. Tools, in our context, are also Arrowhead systems, either service providers or consumers. The definition is intentionally loose, in order to be a reference to heterogeneous use cases – *e.g.* pure software-driven, manufacturing, constructions, etc. – however, it draws some fixed lines over key points that separate conceptually a tool from non-tools in the loop. For a better understanding of these concepts, the definition accepted within the project and examples of tools and non-tools can be found in [8].

Arrowhead tools, as detailed in the previous section, feature properties (such as atomicity and interoperability) that open up their suitability for compositional architectures. Following this line, here we introduce the concept of *toolchains*, which well merges tools and engineering processes. Just as we did for tools, let us first report the official definition of an Arrowhead toolchain [8]: *A toolchain is a collection of tools and of definitions of the corresponding interfaces potentially organized in chain-based or parallel structures. Tools in a toolchain can be substituted/replaced with other tools with the same input/output interfaces.* Hence, the toolchain definition above naturally integrates with the Arrowhead-EP as it follows the same loose coupling principle. Moreover, we stress the importance of making this integration explicit through an *Engineering Process Mapping* (EPM) that brings to the surface a new requirement to satisfy: whenever two EPPs are connected in a use-case specific Arrowhead-EP, then the corresponding toolchain shall establish an automated (when possible) data exchange between tools that address such phases.

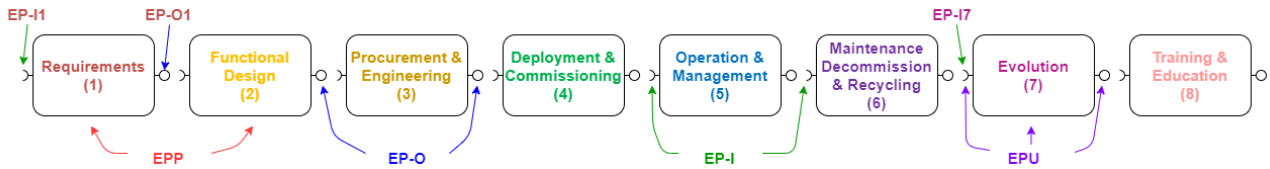


Fig. 2. A graphical overview of the Arrowhead tools engineering process.

IV. Supervising the Process

The automated execution of toolchains can be approached in different ways, depending on what functionalities are supported by the tools. On the other hand, every toolchain execution can be described by a sequential list of tools, starting from the first one to be executed, and ending with the last one. Such an approach, however, can be attributed to linear (*i.e.*, tools are connected one-to-one and sequentially) toolchains only, which might limit its applicability. Nevertheless, this case is the easiest one to analyze, and the potential approaches to tools execution automation will be discussed assuming the linearity of the toolchain. The following approaches to toolchain execution automation, detailed below, are identified:

1) Hard-coded sequence
2) Individually orchestrated sequential execution
3) Supervised execution

This list is not intended to be exhaustive, rather it provides the reader with a brief overview of the approaches to toolchain automation in SOA.

The first approach is the simplest automated execution, where each tool has the information about the next tool to be executed and, once it finished the processing, triggers the next in line. Such an approach requires high effort to be put into the design time to arrange the connections between subsequent tools in a pre-programmed manner. Moreover, when one of the tools is no longer available, the execution of the toolchain usually stops at this particular point, and modifications to the pipeline require modifying the involved tools.

In our case, points 2) and 3) correspond, respectively, to an Arrowhead-orchestrated scenario and an Arrowhead-choreographed scenario. Orchestration of services is seen here to instruct each service consumer *which* is the information provider that it is committed to query and *where* to find it. Conversely, choreography instructs service consumers on *when* and *how* to query those services, making the role of these two policies orthogonal. In fact, we consider Arrowhead choreography to always imply an underlying Arrowhead orchestration to be implemented. More in detail, the second method, where each tool is orchestrated statically, relies on a predefined set of rules that tell which tools should be connected to each other (in a consumer/producer relationship). This approach does not have any dynamical description of the aforementioned relations, and the subsequent tools are manually executed as soon as the input data is available. This makes the synchronization of different data sources difficult or even impossible. The last method elevates the toolchain automated execution to a higher level of abstraction, where there is a dedicated system – a choreographer – responsible for the execution of subsequent steps in a given toolchain recipe [35]. Although the implementation of such a system might appear simple, it should be taken into account that the introduction of a choreographer in SOA should ideally not require any changes in the existing services, but in the systems (adding a new service). Hence, all of the services are orchestrated to the choreographer, *i.e.*, all providers in the toolchain are consumed by it, and it serves as a provider for all consumers regarding control-flow operations.

Various SOA-based workflow management and choreography approaches are compared in [37], concluding that the Arrowhead choreography solution – that we propose to adopt for toolchaining – shows superiority above others in workflow lifecycle stage support, asynchronous service request handling, and parallel execution capabilities. These are key features for the proposed supervised toolchain execution (point 3 in the above list).

The choreographed approach can be built to execute not only linear toolchains, but also to make branched execution, loops, or even reusing the recipes to assemble Toolchains-of-Toolchains (ToT). Another advantage is the ability to synchronize multiple data providers for a common consumer, which enables not only multi-stakeholder operation but also multi-stakeholder cooperation. Finally, toolchains involving design-time tools, which often require actions to be taken by humans that can last for days, can be integrated into such a pipeline as well. The above-mentioned advantages would not be possible with statically defined orchestration rules – more complex logic and supervision are required along with control over the correctness of the execution of subsequent tools.

The main benefit of the proposed approach is that it allows dynamic tool assignments based on the relevant tool's servicing capability and availability, as it works in all Service Oriented Architectures. Eventually, this can lead to an automatic build-up of toolchains covering the engineering lifecycle first partially, then fully. Unlike any other methods or approaches, our proposal covers the first two challenges described in the introduction. The third challenge is related to the adoption for legacy systems – which is discussed in the next chapter.

V. Mapping the Toolchain Model to the Eclipse Arrowhead Framework

With respect to the definitions of tools and toolchains given in Section III, we observe that Eclipse Arrowhead finds its natural application in supporting automation between tools as they take part in a toolchain. In fact, tools as we know them are – a particular instance of – systems, and toolchains can compose a significant set of SoS. The key observation is that automation is enabled by an automatic information flow between tools, especially when those are decoupled and communication between tools is not hardwired.

The latter concept is required in compliance with novel IIoT guidelines and, in our case, with the Arrowhead-EP presented in Section III-B. In this section, we aim to give a consistent hierarchical framework to describe rigorously the extent to which an industrial toolchain is integrated with the Eclipse Arrowhead and the expected implications. This is done by taking into account mostly a selected subset of the Arrowhead Core Services described in the previous Section, in particular those that have a significant repercussion on an engineering process. More in detail, we define a series of Adoption Levels that characterize the level of integration of a toolchain by isolating the functionalities that each core service brings in. The reason why this framework is presented in a hierarchical way (*i.e.* by using the connotation "levels") is that a target adoption level requires most likely the implementation of all the other underlying ones. Note that this does not promote in any way that a higher adoption level means a better or more efficient SoS, as, in fact, a certain adoption level that suits best a definite use case might be overkill for others. Rather, it aims to provide a survey that guides the practitioner in getting familiar with the features of the Eclipse Arrowhead towards Industry 4.0 engineering toolchains and building a solid environment progressively as per the requirement of the target SoS. For the sake of compactness, let us abbreviate the term Adoption Level with AL. A more technical display of the Arrowhead functionalities illustrated in this section can be found in the official documentation [34], alongside with the code of the platform, examples, and application skeletons. Information models of the framework have also been largely investigated in literature, an example is [38], which proposes SysMLv2 for capturing its structural aspect, and describes a real use case in a Norwegian chemical factory.

A. Adoption Level 0 (AL0): Legacy Infrastructure

In this case, no interaction between engineering tools is supported by Eclipse Arrowhead. This includes legacy toolchains that either make use of manual interaction or pass information via a different automation facility. In any of such cases, engineering tools cannot take an active part in processes managed within an Arrowhead Local Cloud.

B. Adoption Level 1 (AL1): Basic SOA

A simple, yet efficient way of adding SOA capabilities to an SoS by setting up a local cloud is to leverage the **Service Registry** core service. The Service Registry provides discovery and loose coupling to a local cloud of systems: service providers register their service record to the Service Registry by specifying their endpoint and a set of metadata, while service consumers query the Service Registry to obtain the endpoint of services that fit their specification. The official implementation of the Core Service can be found in the Arrowhead Tools repository [34], and its deployment requires a relational database and a number of HTTP endpoints that host its services. A custom implementation is also possible by following the guidelines in the documentation. This simple architecture poses a broker between providers and consumers and only adds a step in the service discovery phase while eliminating all at once the need for hardwires between systems and services. In the same way, tools in a toolchain are here considered loosely coupled Arrowhead systems. However, by introducing only the Service Registry, an SoS unavoidably faces a set of drawbacks, for instance, the amount of logic that each system must implement, which gives little to no management capability to the administrator(s) of the toolchain. In the AL1 block of Figure 3 we notice how a consumer system needs to explicitly know which service it is looking for on the Service Registry.

C. Adoption Level 2 (AL2): Mandatory Core Services

Even though a simple SOA can be set up through the use of the Service Registry, by means of the **Authorisation** and the **Orchestration** we achieve what has been defined as the mandatory level of integration of a local cloud in the industrial scenario. More in detail, these two core services shift the level of control for authorization and management to the framework level. This means that authentication and authorization do not need to be managed in a customized way by each couple of interacting systems, instead access control is checked within the authorization layer, which is also committed to releasing usage tokens based on certificates. In particular, access rules stored in the Authorisation system describe the access policies between a consumer and a provider system. This core system is also responsible for providing the session control functionality, achieved through a token generation system. On the other hand, the Orchestration system is responsible for coupling consumers to their specific provider through rules, so the local cloud Manager (an entity that supervises and manages all the interactions in the local cloud) can coordinate the interaction without the need for the single systems to implement the discovery by themselves. This means that consumers only have to put a query to the Orchestration instead of searching their provider in the Service Registry, as the provider is already assigned through an Orchestration rule. In particular, the Orchestration service returns the information (address, port, context path, tokens) that the requesting application needs in order to consume the specified service. The Orchestration system can be also used as a dynamic search engine, in fact the requesting application system can also be asked to find one or more accessible providers that meet certain requirements or metadata, in absence of an orchestration rule. A complete set of information about the Authorisation and the Orchestration systems can be found in the official Arrowhead Tools documentation [34].

D. Adoption Level 3 (AL3): Design Time Planning

A local cloud using the mandatory core services, in some cases, lacks the support for an organization at design time: orchestration rules are often pushed by a human or dedicated software and, when the SoS to be managed is large, this typically becomes impractical. A key observation here is that a large SoS with many atomic services, in industrial scenarios, often has a limited number of configurations that are implemented. For such reason, for efficient design-time planning, the **Plant Description System** is a powerful

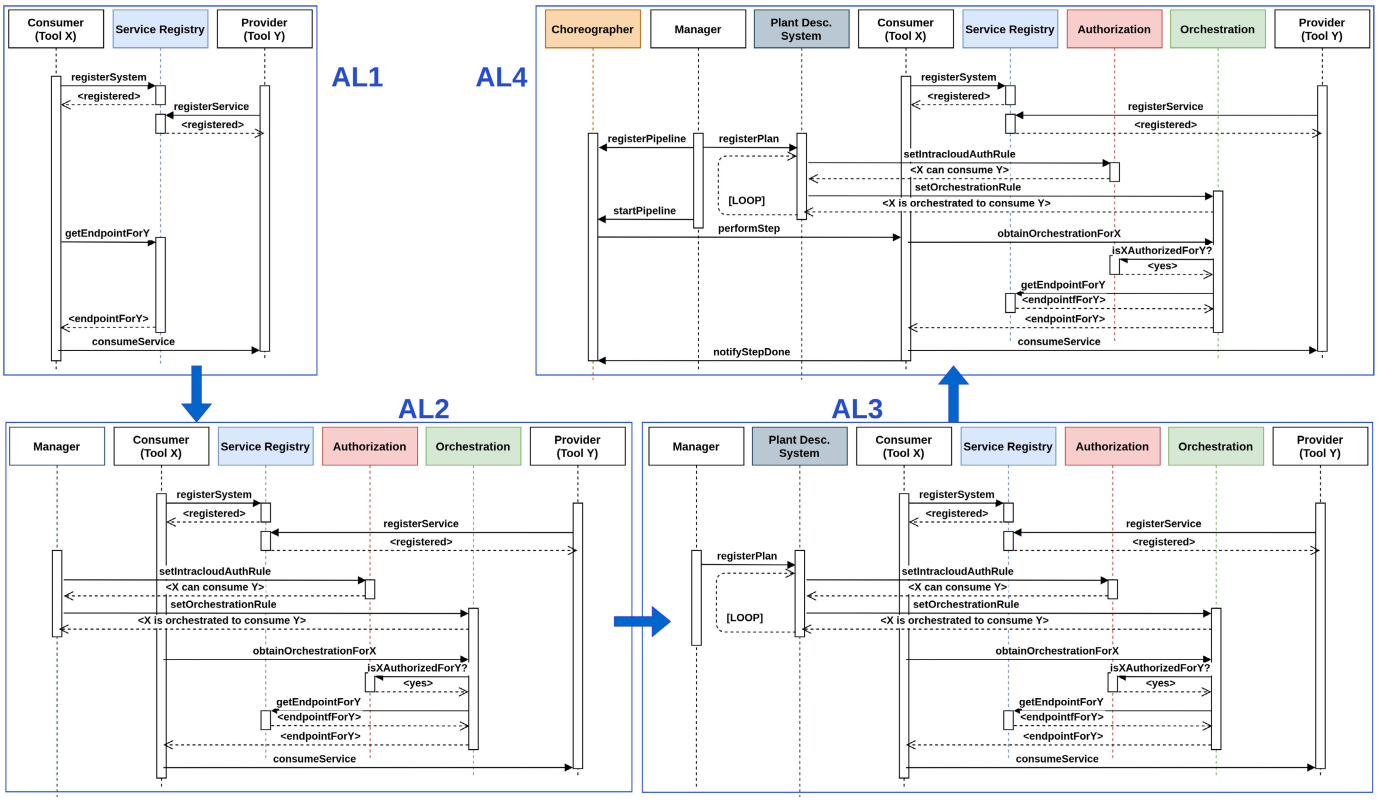


Fig. 3. Sequence diagram models for all Arrowhead adoption levels, in which one provider and one consumer are exemplified.

aid. It allows to specify a whole configuration (i.e. a set of orchestration rules) beforehand by using a description language and to push such rules in bulk onto the Orchestration. Furthermore, it provides the administrator with the feature of saving different configurations and loading them as needed at runtime.

E. Adoption Level 4 (AL4): Runtime Supervision

Some of the industrial toolchains may need a monitoring facility, especially when tools are organized in a pipeline. The **Workflow Choreographer** introduces the capability of supervising the whole runtime process by calling services explicitly when they are needed while keeping track of the whole workflow. More in detail, the workflow logic is specified through a definition language and allows for calling services, waiting for services to respond, executing conditional branches, passing the output of a tool in input to another tool, etc. In order to do so, the Choreographer instantiates a set of **Workflow Executors (WE)**, which are separate processes, each of them interacting with a different tool. In this case, the local cloud Manager first registers a new "recipe" to the Choreographer – *i.e.,* a pipeline of service calls and constraints written in a dedicated language (Arrowhead uses JSON syntax [35]). The recipe specifies a number of service interactions in the local cloud that need to be triggered by the Choreographer itself. With respect to Figure 3, when the pipeline starts, the Choreographer calls directly the service consumer to perform one computation step and return the result. Such a computation step may require querying a service provider, which is orchestrated to the consumer as in AL2. When the consumer is done with the step, it notifies back to the Choreographer. This practice unavoidably requires some changes in the design of tools, as they need to implement either a listening interface for WEs or to be implemented as callable executables. The latter is more suitable for atomic, short tool processes (*e.g.* a robotic arm performing a single operation), more similar to microservices. In other words, getting to AL4 brings undeniable monitoring advantages, however, in some cases, the outcome may not be worth the effort.

VI. The Arrowhead Reference Demonstrator

Within the scope of the Arrowhead Adoption Levels introduced earlier, we present in this section a demonstrator, implemented by the authors, that acts as a set of implementation guidelines for developers. The demo has the role of a training and education tool, which is kept intentionally simple, rather than a complex scenario. The simplified version of the demo that was used as a skeleton for the presented application is open source and the code is publicly available.[4] The demonstrator was embedded in a real-world application: the milk collection and delivery process, operated by DAC.digital as a part of the MuuMap platform. An onboard computer installed in a milk hauler collects data on the amount of milk collected, as well as the temperature of the transported milk. The temperature sensor, which is wirelessly connected to the gateway, is battery-powered. A sensor reconfiguration is made to save battery life on the basis of current and previous measurements. It can be simplified to a rule-based procedure, which reduces the sampling time in cases where the

[4]https://github.com/stradivarius/wp4_toolchain_demo

temperature is constant, and increases if it changes. In order to show the violation of the threshold, the data are stored through the persister tool.

The persister tool (physically executed on a gateway) offers persistence capabilities to the scenario: it periodically queries the temperature sensor, which replies with a data chunk containing the last temperature values and saves the data obtained onto a database. The Persister then exposes a service that allows a consumer to get access to the data stored in the database, as well as to obtain a *configuration* calculated by the Persister itself. Such a configuration is an optimal estimate of k, which the Persister periodically calculates, on top of the last w temperature values. In detail, the new configuration k' is calculated as follows: $k' = max(|\frac{kw}{2}|, 1)$, which is a discretization of common approaches of duty cycle updates based on data change frequency, such as in [39]. The idea is that the sampling interval tends to be shortened as the number of equal values w is large (because it means that the temperature value changes more slowly than the sampling frequency) and the opposite as w tends to zero. k' is then the new suggested configuration, however, it is not applied directly to the temperature sensor. The Gateway Controller in fact, is in charge of consuming the service: it obtains the data points from the Persister, displays them, and ultimately sends a configuration (*i.e.* the value of k') to the temperature sensor, which, in turn, changes the sampling frequency accordingly. The Gateway Controller can choose to use the suggested k', or set a completely arbitrary one, as per the user's choice. The architecture of the system is presented in Figure 4.

A. Elevating Adoption Levels

With AL1, we notice how the sole deployment of the Service Registry is sufficient to guarantee a SOA-mediated interaction. The three systems – the monitoring dashboard has been omitted as it is hardwired and its role can be taken up by the Gateway Controller – have been registered into the Service Registry and, upon consuming a service, they request for the exact service endpoint to the Service Registry. This implies, however, that the service name or some metadata is known to each of the systems, leaving them with a certain degree of autonomy. With AL2 such autonomy is limited because the service interactions are handled by orchestration rules, i.e. systems cannot directly obtain information from the Service Registry, instead they query the Orchestration system for the services that they have to consume, according to orchestration rules set by a local cloud manager. Note that the setup of orchestration and authorization rules by the local cloud manager at design time can be taken up by additional design tools. Examples can be the Magic Draw Tool [16] and the AHT Management Tool.[5] The first one allows one to design the local cloud interactions via a SysML diagram and export it in a format that is readable by the Management Tool. The latter then registers the services into the Service Registry, the authorization rules into the Authorisation, and the orchestration rules into the Orchestration. The output of the Magic Draw Tool can be alternatively fed to the Plant Description System, which eventually registers the orchestration rules as a single configuration, as it happens in AL3.

[5]https://github.com/arrowhead-f/mgmt-tool-js

B. Choreography of the Toolchain

Finally, AL4 involves the Choreographer as the main active element in the Local Cloud, to comply with the concepts outlined in Section IV. This is the only AL that requires a change in the implementation of the application systems and tools, as they all need to host an endpoint that is reachable by the WEs associated with the Choreographer, as presented in Figure 4. So far, the Choreographer has been shown to work with application systems designed to perform predefined tasks in a programmed sequence, without involving any data processing, yet requiring to notify the Choreographer that the step has been completed. Even though this has been implemented here, the demo is a proof of concept that would not necessarily benefit from AL4, as some of the tools and systems are not conceptually passive and consumers need to additionally indicate when they have finished processing the data. However, if we imagine this demo to scale up, involving a number of different data sources, each one with its own time-driven constraint, then correctly choreographing such interactions becomes crucial.

Our proposal of process supervision through the Arrowhead Choreographer is novel from two main points of view. First of all, this is the only process supervision that spans the whole EP, while other efforts in the literature focus on its design-time parts [23], [24], thus not entirely addressing the problem space – *i.e.*, the run-time part of the EP. This makes it difficult to estimate how such supervision may take place in Arrowhead-free scenarios, due to the lack of terms of comparison. Secondly, to our knowledge, this is the only process supervision applied to SOA (differently from [23], [24]), which necessarily requires every service to be compliant. For this reason, we currently do not foresee an easy way of integrating services without changing them. We consider this an important challenge and an interesting future direction of research. On the other hand, though, we note that such a change is only necessary once at design time, yet it enables the setup of different pipelines without any run-time change to services.

C. Passage to Industrial Use Cases

The demonstrator described in this section serves as a easily comprehensible showcase of how one could benefit from implementing the subsequent adoption levels and Arrowhead Choreographer and their technical implications. However, in order to couple the presented ideas and methodology with industrial use cases, a link between the presented demonstrator and practical implementations is shown in a few examples. The first two examples show the utilization of runtime supervision (AL4) with the Choreographer, while the latter two will be used to show how the proposed methodology can be used in these cases in order to increase the automation level.

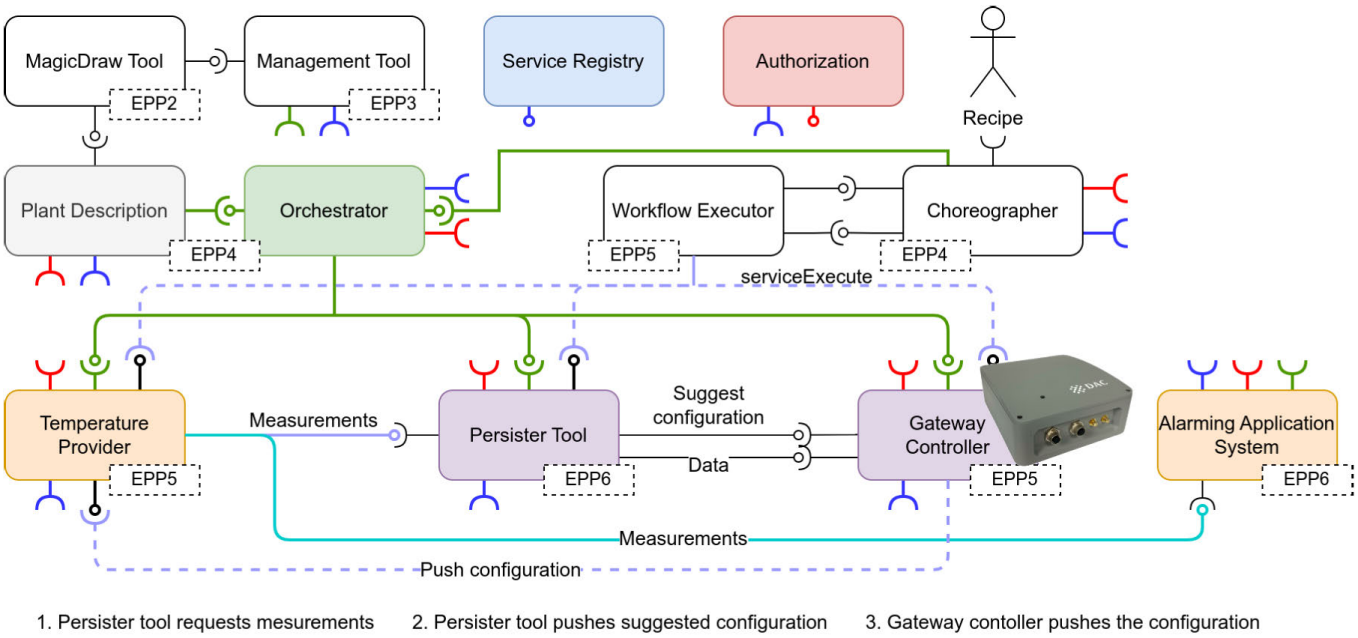


Fig. 4. Architecture of the demonstrator system with the workflow choreographer at the adoption level 4. EPPX describes the number of engineering phase to which the tool or interaction is attributed (see Figure 2). It is assumed that all tools are orchestrable and choreographable.

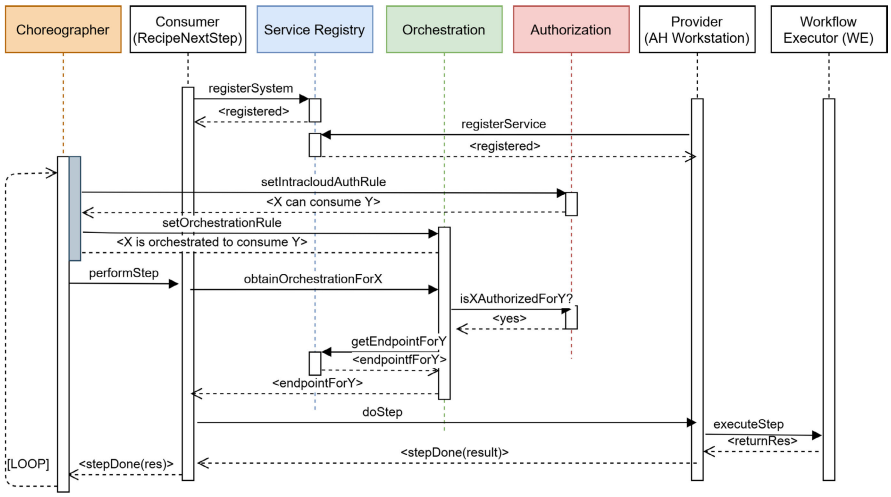


Fig. 5. Sequence diagram for choreographer-based supervision with Workflow Executors (WE), applicable for the robotic and automotive use cases.

D. Robotic Operation Supervision

In [40] a robotic operation is used as a showcase of how the Choreographer can be utilized to perform supervised robotic operations for flexible manufacturing systems. The Choreographer receives production orders from the Enterprise Resource Planning system, resulting in a set of recipes for WEs, where for each recipe a new independent WE is spawned to supervise the robotic operation. This use-case utilizes the Choreographer for the run-time production operation at CPS level in the same way as we suggest for the tools in the toolchain. Figure 5 demonstrates the sequence diagram for the robotic operation supervision.

Note that in this industry-driven scenario, the operational supervision of the Choreographer and the separation of the Arrowhead service provider workstation and the WEs are emphasized. Hence, while the services from AL2 (Manager) and AL3 (Plant Description) are not explicitly separated from the Choreographer system, they are implemented. In this scenario, the Choreographer-based supervision manages the interactions in the local cloud and pushes orchestration rules to the Orchestrator in bulk based on a design-time-prepared description of SoS.

The proposed approach aligns with the robotic supervision use case. Choreographer is again employed to oversee the supervised process through WE. Each requested operation recipe prompts the creation of a new WE to control the process. Similar to the reference demonstrator, the spawned WE receives feedback from another system on the successful execution of the requested operation, subsequently notifying the choreographer. Depending on the external system, new operations may be requested and the choreographer will be responsible for creating a new dedicated WE. Although both use cases are applied in different domains, the applied supervision methodology can be enclosed in the same framework involving the choreographer.

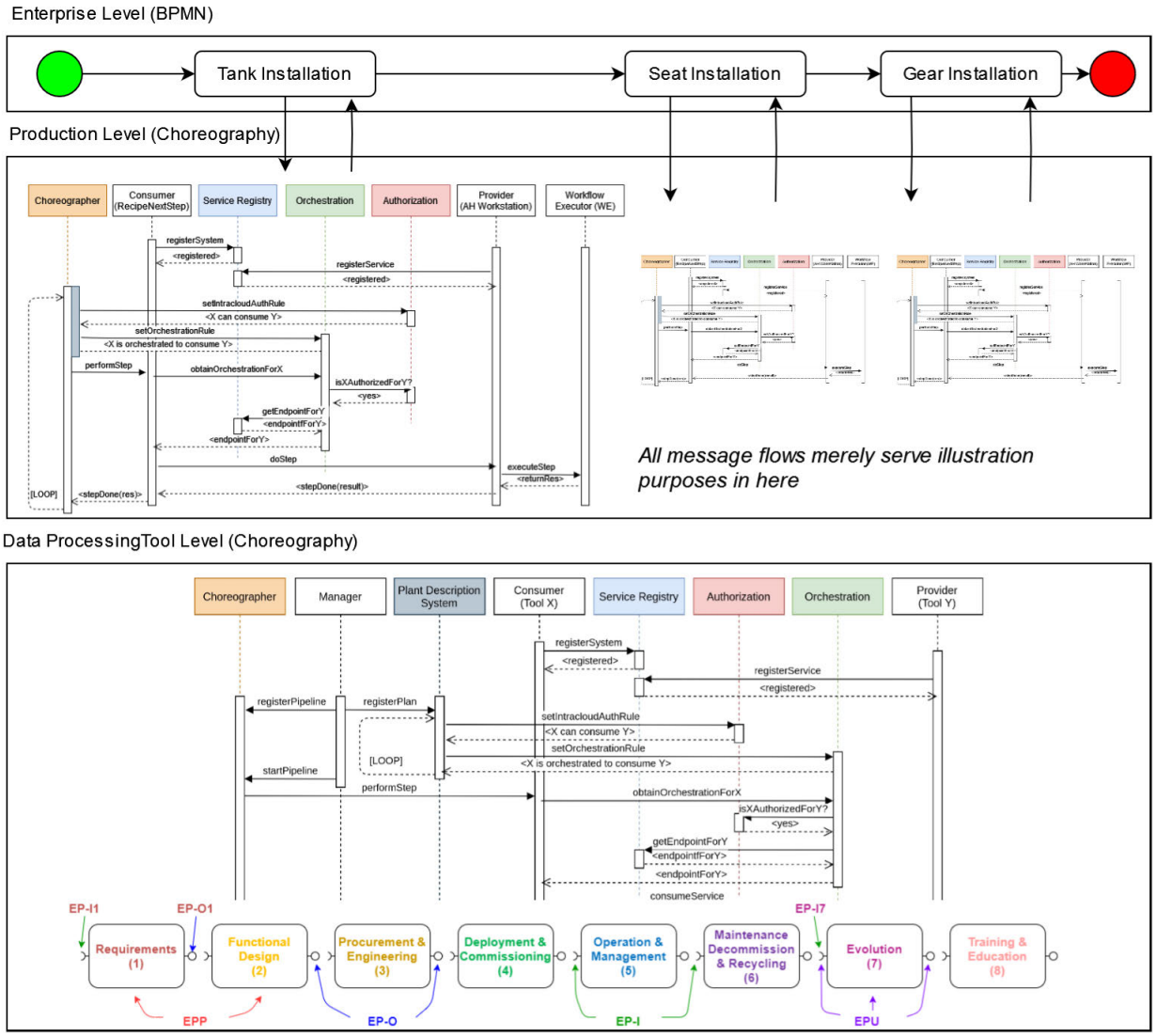


Fig. 6. Multi-level choreography – automotive use case. The data processing toolchain runs simultaneously with the enterprise level-triggered Production level choreography. NOTE that the internal message flow figures simply illustrate that there are both sequential and parallel flows being executed at different levels.

E. Automotive Assembly Plant

An Arrowhead-managed assembly shop is introduced in [36], showcasing a part of a manufacturing plant that assembles vehicles. The Choreographer is used to supervise the ordered execution of particular assembly blueprints, where each possible operation is defined as a recipe related to a workflow and managed by the WE invoked by the Choreographer. The WE assures that each step of a workflow is executed in an ordered manner, while the Choreographer delegates the adequate WEs, requests services from the Orchestration, assures proper configuration/reconfiguration of services, and analyzes the results from WEs. The operational sequence described in Figure 5 applies here as well. In this demonstrator use case, a multi-level workflow is executed, where the workstation recipes refer to, *e.g.*, tank installation or seat installation processes.

Note that there are many other workflows defined for the automotive assembly plant, which in our demonstrator on adoption levels can be defined as, for instance, data processing workflow including MLOps [41] or different provisioning system injection flows. The integration of the *production level* workflow into the multi-level workflow (with Choreography as in Figure 5) is shown by Figure 6, including a simultaneous data processing Toolchain Choreography for the Arrowhead Tools Engineering Process (Figure 2) that happens at the data processing level (with the AL4 sequence of Figure 3. On the *production level* there are separated Choreography instances (per recipe) controlling the workflow as described in [36], whereas in the *data processing tool level* there is a separated Choreographer instance responsible for the Engineering Process Toolchain choreography.

F. Onboarding Toolchain

In [10] the onboarding toolchain was described as one of the use cases where Eclipse Arrowhead was used as the interoperability enabler. The toolchain was developed at DAC.digital[6] in order to simplify and automate the

[6]https://dac.digital

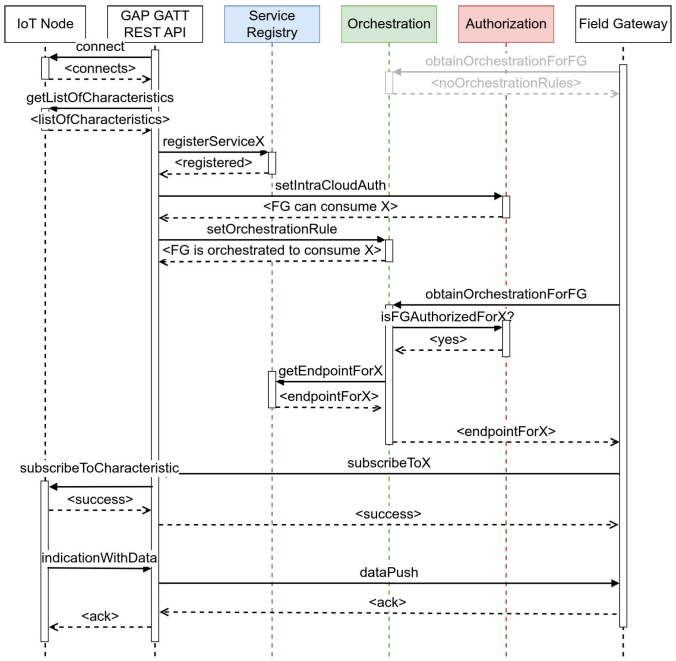


Fig. 7. Sequence diagram for onboarding toolchain.

onboarding process of new IoT devices to existing local clouds and was eventually applied to logistics use cases where package delivery was constantly monitored from pick-up to delivery stages by providing Inertial Measurement Unit (IMU) measurements and temperature readings of the transported goods (*e.g.*, medicines). The toolchain was implemented up to AL2, where all core services were utilized, and additional scripts were developed to ensure run-time orchestration of new devices. Still, one of the most problematic issues was handling devices that are appearing in and disappearing from the range of the local cloud, so that they are properly orchestrated and served. AL3 and AL4 could be used here to handle that, where Plant Description would have a predefined set of orchestration rules for a specific type of device, while the Choreographer could be used to invoke the device registration workflow, and then supervise the data provisioning part of the use case. In addition, the off-boarding procedure could be handled gracefully for further reuse of the same device. The sequence diagram of this use case is presented in Figure 7.

G. Edge-to-Cloud Integration for Production Monitoring

The authors of [42] propose to use Eclipse Arrowhead in order to build a hierarchical structure from local clouds in order to implement production/assembly monitoring application. The idea is to have services that could be flexibly delegated from the edge level to the cloud computing environment through both intra- and inter-cloud communication. The data is collected and visualized in a local cloud close to the workstation, while data processing happens on the cloud. Again, this use case reached AL2. The vision of implementing AL3 and AL4 appears to be clear as the authors provide the flow between different services in various scenarios as directed acyclic graph. First of all, the Plant Description can be used here to design interactions between particular services on different levels (edge and cloud). The Choreographer can be utilized for the supervised execution of various workflows related to desired processing functionalities, for the edge, for the cloud, and for interconnected workflows. In that way, manual reconfiguration and supervision would be automated.

VII. Conclusion

In this paper, the issue of automated execution and synchronization of engineering tools within toolchains is addressed. In particular, related work and standards are introduced as a baseline for this paper, on the basis of which basic definitions are formulated. Different approaches to toolchain automation are described and discussed, and choreography is proposed as the main enabler for complex toolchain automation and supervision. The toolchain model is then mapped to fit in the Eclipse Arrowhead framework, which is used as the ecosystem with the services needed to ensure the appropriate level of interoperability within the toolchain. The use of mandatory and supporting core services of Eclipse Arrowhead is organized in Adoption Levels (AL), introduced and defined in this paper. For each of the Adoption Level the sequence diagrams are provided. The final results are demonstrated as an actual implemented use-case, where the toolchain is presented at each of the proposed AL, and the Choreographer is used up to AL4 to automate the execution of the considered toolchain, validating the proposed methodology.

References

[1] J. Delsing et al., *3 The Arrowhead Framework Architecture: Arrowhead Framework*. CRC Press, Feb. 2017, pp. 43–88.

[2] *Industrial Systems, Installations and Equipment and Industrial Products—Structuring Principles and Reference Designations—Part 1: Basic Rules*, Standard IEC 81346-1:2022, International Electrotechnical Commission, 2022.

[3] P. Varga et al., "Making system of systems interoperable—The core components of the arrowhead framework," *J. Netw. Comput. Appl.*, vol. 81, pp. 85–95, Mar. 2017.

[4] C. Paniagua, J. Eliasson, and J. Delsing, "Interoperability mismatch challenges in heterogeneous SOA-based systems," in *Proc. IEEE Int. Conf. Ind. Technol. (ICIT)*, 2019, pp. 788–793.

[5] C. Peltz, "Web services orchestration and choreography," *Computer*, vol. 36, no. 10, pp. 46–52, Oct. 2003.

[6] T. Burns, J. Cosgrove, and F. Doyle, "A review of interoperability standards for industry 4.0," *Proc. Manufacturing*, vol. 38, pp. 646–653, 2019.

[7] D. Gürdür, F. Asplund, and J. El-Khoury, "Measuring tool chain interoperability in cyber-physical systems," in *Proc. 11th Syst. Syst. Eng. Conf. (SoSE)*, 2016, pp. 1–4.

[8] G. Kulcsár, M. S. Tatara, and F. Montori, "Toolchain modeling: Comprehensive engineering plans for industry 4.0," in *Proc. IECON 46th Annu. Conf. IEEE Ind. Electron. Soc.*, Oct. 2020, pp. 4541–4546.

[9] G. Urgese, P. Azzoni, J. van Deventer, J. Delsing, A. Macii, and E. Macii, "A SOA-based engineering process model for the life cycle management of system-of-systems in industry 4.0," *Appl. Sci.*, vol. 12, no. 15, p. 7730, Aug. 2022.

[10] G. Kulcsár et al., "Modeling an industrial revolution: How to manage large-scale, complex IoT ecosystems?" in *Proc. IFIP/IEEE Int. Symp. Integr. Netw. Manage. (IM)*, May 2021, pp. 896–901.

[11] L. Prades, F. Romero, A. Estruch, A. García-Dominguez, and J. Serrano, "Defining a methodology to design and implement business process models in BPMN according to the standard ANSI/ISA-95 in a manufacturing enterprise," *Proc. Eng.*, vol. 63, pp. 115–122, 2013.

[12] W. Ochoa, F. Larrinaga, and A. Pérez, "Context-aware workflow management for smart manufacturing: A literature review of semantic web-based approaches," *Future Gener. Comput. Syst.*, vol. 145, pp. 38–55, Aug. 2023.

[13] J. Ma, G. Wang, J. Lu, H. Vangheluwe, D. Kiritsis, and Y. Yan, "Systematic literature review of MBSE tool-chains," *Appl. Sci.*, vol. 12, no. 7, p. 3431, Mar. 2022.

[14] P. Varga et al., "Converging telco-grade solutions 5G and beyond to support production in industry 4.0," *Appl. Sci.*, vol. 12, no. 15, p. 7600, Jul. 2022.

[15] W. Chang, S. Zhao, R. Wei, A. Wellings, and A. Burns, "From Java to real-time Java: A model-driven methodology with automated toolchain," in *Proc. 20th ACM SIGPLAN/SIGBED Int. Conf. Lang., Compil., Tools Embedded Syst.*, Jun. 2019, pp. 123–134.

[16] G. Kulcsár et al., "From models to management and back: Towards a system-of-systems engineering toolchain," in *Proc. NOMS IEEE/IFIP Netw. Operations Manage. Symp.*, Apr. 2020, pp. 1–6.

[17] L. E. Lwakatare et al., "DevOps in practice: A multiple case study of five companies," *Inf. Softw. Technol.*, vol. 114, pp. 217–230, Oct. 2019.

[18] T. Maikantis, T. Chaikalis, A. Ampatzoglou, and A. Chatzigeorgiou, "SmartCLIDE: Shortening the toolchain of SOA-based cloud software development by automating service creation, composition, testing, and deployment," in *Proc. 25th Pan-Hellenic Conf. Informat.*, Nov. 2021, pp. 306–311.

[19] C. Hegedus, P. Varga, and A. Frankó, "A DevOps approach for cyber-physical system-of-systems engineering through arrowhead," in *Proc. IFIP/IEEE Int. Symp. Integr. Netw. Manage. (IM)*, May 2021, pp. 902–907.

[20] *Representation of Process Control Engineering—Requests in P&I Diagrams and Data Exchange Between P&ID Tools and PCE-CAE Tools*, Standard IEC 62424:2016, International Electrotechnical Commission, p. 2016.

[21] *Industrial Automation Systems and Integration—Integration of Life-cycle Data for Process Plants Including Oil and Gas Production Facilities*, Standard ISO 15926:2004, ISO Central Secretary, 2004.

[22] *OPC Unified Architecture—Part 1: Overview and Concepts*, Standard IEC 62541:2020, OPC Foundation, 2020.

[23] E. Stav, S. Walderhaug, M. Mikalsen, S. Hanke, and I. Benc, "Development and evaluation of SOA-based AAL services in real-life environments: A case study and lessons learned," *Int. J. Med. Informat.*, vol. 82, no. 11, pp. e269–e293, Nov. 2013.

[24] J. Lu, D. Chen, J. Wang, and M. Torngren, "Towards a service-oriented framework for MBSE tool-chain development," in *Proc. 13th Annu. Conf. Syst. Syst. Eng. (SoSE)*, 2018, pp. 568–575.

[25] Y. Guo, J. Wei, G. Chen, and S. She, "A unified model-based systems engineering framework supporting system design platform based on data exchange mechanisms," in *Knowledge and Systems Sciences*, J. Chen, T. Hashimoto, X. Tang, and J. Wu, Eds. Singapore: Springer Nature, 1007.

[26] C. Neureiter and C. Binder, "A domain-specific, model based systems engineering approach for cyber-physical systems," *Systems*, vol. 10, no. 2, p. 42, Mar. 2022.

[27] S. Mantravadi and C. Möller, "An overview of next-generation manufacturing execution systems: How important is MES for industry 4.0?" *Proc. Manuf.*, vol. 30, pp. 588–595, 2019.

[28] E. Negri, S. Berardi, L. Fumagalli, and M. Macchi, "MES-integrated digital twin frameworks," *J. Manuf. Syst.*, vol. 56, pp. 58–71, Jul. 2020.

[29] J. Wan et al., "Reconfigurable smart factory for drug packing in healthcare industry 4.0," *IEEE Trans. Ind. Informat.*, vol. 15, no. 1, pp. 507–516, Jan. 2019.

[30] Y. Wang et al., "MPCSM: Microservice placement for edge-cloud collaborative smart manufacturing," *IEEE Trans. Ind. Informat.*, vol. 17, no. 9, pp. 5898–5908, Sep. 2021.

[31] G. Schneider, P. Patolla, M. Fehr, D. Reichelt, F. Zoghlami, and J. Delsing, "Micro service based sensor integration efficiency and feasibility in the semiconductor industry," *Infocommunications J.*, vol. 14, no. 3, pp. 79–85, 2022.

[32] S. Maksuti, M. Zsilak, M. Tauber, and J. Delsing, "Security and autonomic management in system of systems," *Infocommunications J.*, vol. 13, no. 3, pp. 66–75, 2021.

[33] D. Kozma, P. Varga, and G. Soós, "Supporting digital production, product lifecycle and supply chain management in industry 4.0 by the arrowhead framework—A survey," in *Proc. IEEE 17th Int. Conf. Ind. Informat. (INDIN)*, vol. 1, Jul. 2019, pp. 126–131.

[34] The Arrowhead Consortia. *Eclipse Arrowhead Developer Resources*. Accessed: Jan. 20, 2024. [Online]. Available: https://projects.eclipse.org/projects/iot.arrowhead/developer

[35] P. Varga, D. Kozma, and C. Hegedús, "Data-driven workflow execution in service oriented IoT architectures," in *Proc. IEEE 23rd Int. Conf. Emerg. Technol. Factory Autom. (ETFA)*, vol. 1, Sep. 2018, pp. 203–210.

[36] D. Kozma, P. Varga, and F. Larrinaga, "Dynamic multilevel workflow management concept for industrial IoT systems," *IEEE Trans. Autom. Sci. Eng.*, vol. 18, no. 3, pp. 1354–1366, Jul. 2021.

[37] J. G. Represa et al., "Investigation of microservice-based workflow management solutions for industrial automation," *Appl. Sci.*, vol. 13, no. 3, p. 1835, Jan. 2023.

[38] J. Delsing, G. Kulcsár, and Ø. Haugen, "SysML modeling of service-oriented system-of-systems," *Innov. Syst. Softw. Eng.*, pp. 1–17, May 2022.

[39] F. Wang, S. Wu, K. Wang, and X. Hu, "Energy-efficient clustering using correlation and random update based on data change rate for wireless sensor networks," *IEEE Sensors J.*, vol. 16, no. 13, pp. 5471–5480, Jul. 2016.

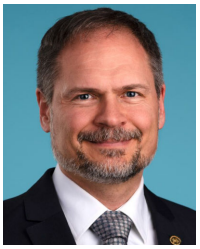
[40] D. Kozma, P. Varga, and K. Szabó, "Achieving flexible digital production with the arrowhead workflow choreographer," in *Proc. IECON 46th Annu. Conf. IEEE Ind. Electron. Soc.*, Oct. 2020, pp. 4503–4510.

[41] C. Hegedus and P. Varga, "Tailoring MLOps techniques for industry 5.0 needs," in *Proc. IFIP/IEEE Int. Conf. Netw. Service Manage. (CNSM)*, 2023, pp. 1–7.

[42] D. Hästbacka et al., "Dynamic edge and cloud service integration for industrial IoT and production monitoring applications of industrial cyber-physical systems," *IEEE Trans. Ind. Informat.*, vol. 18, no. 1, pp. 498–508, Jan. 2022.

Federico Montori (Member, IEEE) received the B.S. and M.S. degrees (summa cum laude) in computer science and the Ph.D. degree in computer science and engineering from the University of Bologna, Italy, in 2012, 2015, and 2019, respectively. He was a Visiting Researcher with the Swinburne University of Technology, Australia, Luleå Tekniska Universitet, Sweden, and Technische Universität Ilmenau, Germany. He is currently a Senior Assistant Professor with the University of Bologna. He has participated in several EU projects and he was the WP Leader for the H2020 Project Arrowhead Tools. His primary research interests include mobile crowdsensing (MCS), pervasive and mobile computing, the IoT automation, and data analysis for the IoT scenarios.

Marek S. Tatara (Member, IEEE) was born in Olsztyn, Poland, in 1991. He received the M.Sc. degree and the Ph.D. degree in the field of automatic control, electronics and electrical engineering from the Gdańsk University of Technology, in 2019. He is currently an Assistant Professor with the Gdańsk University of Technology and the Chief Scientific Officer with DAC.digital. His research interests include the mathematical modeling of physical processes, diagnostics, signal processing applied to industrial processes, and evolutionary music composition. He is a member of Polish Society for Measurement, Automatic Control and Robotics, and the Vice-Chair of the Technical Committee TC 7.5. Intelligent Autonomous Vehicles on Social Media of International Federation of Automatic Control.



Pál Varga (Senior Member, IEEE) is currently the Head of the Department of Telecommunications and Media Informatics, Budapest University of Technology and Economics. His main research interests include communication systems, cyber-physical systems, the 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, and machine learning techniques. He is active both in the IEEE ComSoc and IEEE IES Communities. He has been a TPC and an OC member in many IEEE conferences, and he was the General Co-Chair of IEEE NOMS 2020 and 2022 editions. He is an editorial board member in many journals, an Associate Editor of IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT, and the Editor-in-Chief of the *Infocommunications Journal*.