# Dynamic Multilevel Workflow Management Concept for Industrial IoT Systems

Dániel Kozma, Pál Varga, *Member, IEEE*, and Felix Larrinaga

***Abstract*— Workflow management is implemented in manufacturing at many levels. The nature of processes varies at each level, hindering the use of a standard modeling or implementation solution. The creation of a flexible workflow management framework that overarches the heterogeneous business process levels is challenging. Still, one of the promises of the Industry 4.0 initiative is precisely this: to provide easy-to-use models and solutions that enable efficient execution of enterprise targets. By addressing this challenge, this article proposes a workflow execution model that integrates information and control flows of these levels while keeping their hierarchy. The overall model builds on the business process model and notation (BPMN) for modeling at the enterprise level and recipe modeling based on colored Petri net (CPN) at the production level. Models produced with both alternatives are implemented and executed in a framework supported by an enterprise service bus (ESB). Loosely coupled, late-bound system elements are connected through the arrowhead framework, which is built upon the service-oriented architecture (SOA) concept. To prove its feasibility, this article presents the practical application of the model via an automotive production scenario.**

***Note to Practitioners*—The methodology detailed in this article can serve as a basis for experts who are dealing with industrial workflows. Reacting to the requirements of Industry 4.0, i.e., the virtualization, decentralization, modularity, real-time capability, and service orientation, this article provides a concept that can answer all the defined criteria. First, it adopts a new two-level approach to workflow management, which makes the understanding and control of workflows easier, enhancing transparency. Furthermore, it demonstrates how—even completely different—applications and modeling languages can be integrated into a service-oriented architecture (SOA). The presented composition and the used tools are all tried and tested. Behind the solution described in this article, there is a genuine, working code wherewith the presented end-to-end workflow management can be achieved. Following the methodology detailed in this article, the readers can construct their workflow management composition. In order to report on the performance of the created solution, this article presents different measurement compositions that allow the investigation of the essential components separately, demonstrating the scalability and temporal parameters.**

***Index Terms*— Business process management, industrial engineering, Internet of Things (IoT), manufacturing automation, Petri nets, service-oriented systems engineering, systems modeling.**

Manuscript received December 2, 2019; revised March 6, 2020; accepted June 9, 2020. This article was recommended for publication by Editor B. Vogel-Heuser upon evaluation of the reviewers' comments. This work was supported in part by the Department of Education, Universities, and Research of the Basque Government under the projects Ikerketa Taldeak (Grupo de Sistemas Embebidos), in part by the European H2020 Research and Innovation Program, ECSEL Joint Undertaking, and National Funding Authorities from 19 involved countries under the project Productive 4.0 under Grant GAP737459 and 999978918, in part by the Grant EU ECSEL JU through the H2020 Framework Programme under JU Grant 826452 (Arrowhead Tools project), and in part by the Partners' National Funding Authorities. *(Corresponding author: Dániel Kozma.)*

Dániel Kozma and Pál Varga are with the Department of Telecommunications and Media Informatics, Budapest University of Technology and Economics, 1111 Budapest, Hungary (e-mail: kozma@tmit.bme.hu; pvarga@tmit.bme.hu).

Felix Larrinaga is with the Embedded System Group, Mondragon University, 20500 Mondragon, Spain (e-mail: flarrinaga@mondragon.edu).

This article has supplementary downloadable material available at http://ieeexplore.ieee.org, provided by the authors. It includes one multimedia AVI format movie clip, which shows the built prototype based on the presented concept and toolset by this article. This material is 51.6 MB in size.

Color versions of one or more of the figures in this article are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TASE.2020.3004313

## I. Introduction

ONE of the promises of the Industrial Internet of Things (IIoT) concept regarding production is to gain better control over complete processes through dynamic information capture, processing, and feedback. While current production systems function well, their efficiency can be further improved through fast reconfiguration according to production orders' timing and corresponding resource availability. In order to enable dynamic reconfiguration, the system elements (from the enterprise level to the production workstations (WSs) on the shop floor) need to share status information and commands.

Information sharing is limited due to heterogeneous infrastructures and the incompatibility of legacy protocols. Cyber-physical system (CPS) and service-oriented architecture (SOA) approaches enhance interoperability among systems and open the door to the Industry 4.0 initiative. This initiative [1] raises expectations on methods [2], procedures [3], safety and security compliance [4], and tools to support flexible interoperable and smart manufacturing [5] throughout the organization—or even among organizations. The SOA [6] approach enables the automation of distributed processes supported by services provided through product lines. It is essential to have efficient process management in this hierarchical environment with different actuation levels.

This article addresses the dynamic construction of models for business processes, considering the different requirements at two levels. The proposal integrates business process model and notation (BPMN) modeling [7] at the enterprise level and colored Petri net (CPN) [8] at the production level. The solution automatically deploys those models over an enterprise service bus (ESB) [9] where the processes are implemented and monitored. An arrowhead system, the workflow choreographer [10], builds the models from the production order. In general, the arrowhead framework [11] provides

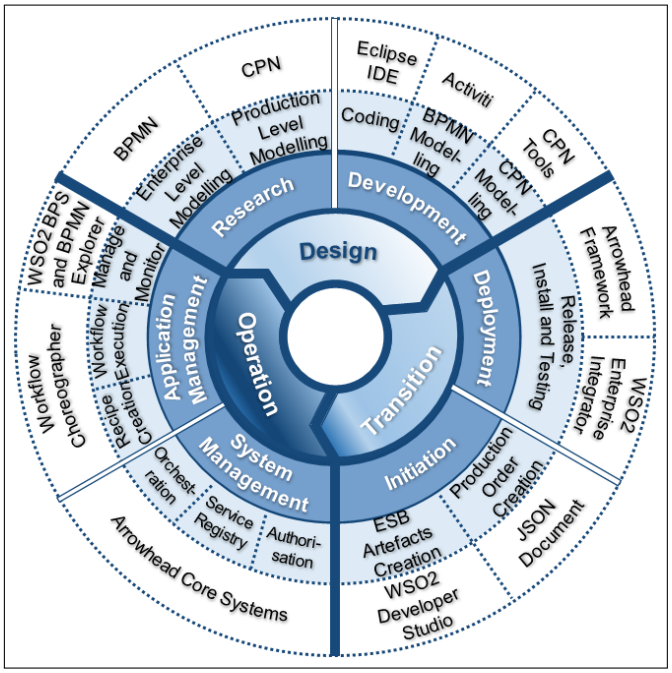


Fig. 1. Development lifecycle phases, and the processes, tasks, and tools used within the proposed dynamic, multilevel workflow management concept.

interoperability and integrability for heterogeneous systems in an IIoT architecture.

This article proposes the integration of two different modeling techniques for workflow design at enterprise and production levels in heterogeneous and dynamically changing environments. The novelty of this article strives as follows:

1) the dynamic construction of those workflows using BPMN and CPN from a production recipe;
2) the automatic deployment of those models over an ESB where the processes are implemented and monitored.

Fig. 1 shows the elements of the integrated model. The internal rings of this model are a merge of the ITIL service lifecycle [12] and other product lifecycle models [13]. They describe the central development and operation stages of the lifecycle, i.e., (from inside to outside) the design, transition and operation phases, and their subprocesses, tasks, and related tools, which is the outer ring of Fig. 1, and implementation specific to our current proposal.

The structure of this article is as follows. Section II discusses the related work. Sections III–V present our approach, including the architecture and frameworks to be used. Section VI demonstrates a use case for the proposal. Section VII discusses the implementation, Section VIII provides the results on its performance, and Section IX concludes this article.

## II. Related Work

This section presents related work on the field of business processes, the modeling techniques employed for their representation, the capability to adapt to process changes, and the ability to share resources among processes. The results expounded here describe the research process of the design phase in Fig. 1.

### A. Modeling Business Processes

In recent years, modeling business processes has attracted the interest of various research communities. Workflow-related concerns—such as the management of resources [14], optimization [15], scalability [16], adaptability [17], and efficiency [18]—are continuously evolving and changing. Workflow processes have become a popular and essential topic partially due to the Industry 4.0 initiatives. This section summarizes the recent advancements that are most relevant to our approach. According to the survey of Krishna *et al.* [19], business processes describe the production of goods or services as a set of local tasks and interorganization exchanges. They found that the current primary business process modeling notations have a workflow perspective of business processes. Brouns *et al.* [20] presented a thorough state-of-the-art study that identifies the gap between the digital world and IoT about business process design and implementation. The study reinforces the importance of creating a uniform way of representing IoT in business process models. Modeling languages do not fully cover concepts, such us availability and mobility of resources or process context information. Among the alternatives for modeling analyzed, such as the BPMN 2.0, event-driven process chains (EPCs), and unified modeling language (UML), their article pinpoints BPMN as the best position to represent IoT although some improvements are necessary. BPMN 2.0 is an ISO standardized notation for modeling business processes that can be made executable either using process engines (e.g., Activiti, Bonita BPM, or jBPM) or using model transformations into executable languages [e.g., Business Process Execution Language (BPEL)].

Enhancing the processing capacity of IoT devices makes them active participants in business processes by executing parts of the business logic. IoT devices can manage data and make decisions locally, reducing the amount of data exchanged and the need for central processing. This decentralization of business processes requires support at both design and execution time. Domingos and Martins [21] proposed a solution, which uses BPMN to model both centralized and IoT decentralized business processes with the same level of abstraction.

Mass *et al.* [22] presented a comparison between two business process workflow execution approaches—an embedded workflow engine and a coding program representing the workflow. They conclude that executing business processes on a workflow engine cannot be justified in cases where system resources are sparse, and the model is not reused. In cases where the same process is executed multiple times and lot of memory is available, a workflow engine is a viable solution.

### B. Workflows at Enterprise Level

Several business process modeling tools and languages have been proposed to describe, analyze, and evaluate business processes [23]. The BPMN [7] is one of the most widely used and standardized modeling languages that can be used to describe workflow structure, organizational level tasks, helping to manage process-related resources effectively. Consequently, BPMN serves as a universal language that bridges

the communication gap that often occurs between business process planning and implementation. The primary goal with BPMN is to support business process management and provide a structure that is easy to understand for every participant during the production. The BPMN specification also provides a graphical description and has the following components.

1) *Events:* Start, intermediate, and end.
2) *Activities:* Tasks: service, user, script, mail, receive, and business; multiple instances; subprocesses; and Loop.
3) *Gateways:* Exclusive, inclusive, parallel, event-based, and complex.
4) *Data and Flows:* Data object, association, and sequence, default, and message flow.

There are several possibilities for creating and executing BPMN workflows, depending on the solution preferred by the designer. The most common process engines are [24] Activiti, jBPM, or Bonita BPM. Model transformation to some executable language, such as BPEL, is also possible [19].

### C. Modeling Parallel Workflows at Production Level

Modeling languages, such as BPMN or BPEL, are relatively easy to interpret, but they are not entirely suited to describe very complex production cases. In contrast, based on a previous comparison [10], Petri nets are useful for describing complex logic that can represent distributed systems and production flows as well. The Petri net [8], [25] is a widely known and used graphical-mathematical model-description language. Briefly described, it is a directed bipartite graph consisting of places and transitions. Directed arcs carry "firing" conditions between places and transitions. Although the language satisfied the industrial expectations in the past, the requirements of Industry 4.0 already set higher demands on modeling languages in general, in which traditional Petri net can no longer satisfy in the sense of, e.g., timing or hierarchy. However, along with the recent digital industrial revolution, the Petri net has also undergone many iterations—resulting among others the CPN [26]. This extension adds the ability to carry more complex information in the tokens to Petri nets, becoming the token "colored." It also allows the use of time as a parameter and supports a hierarchical composition. For the efficient modeling of CPN, several tools have been developed, and the best known is the CPN Tools [27]. CPN Tools is based on the Standard Meta Language (SML)—which is a functional programming language—but the CPN Tools also extends SML with functions such as color sets and constructs for declaring variables, multisets, and related operators and functions. SML enables simulation, state-space analysis, and performance analysis [28]. Section III puts our proposal in context with the related work.

## III. Concept of Adaptable Workflow Management in Dynamically Changing Environments

### A. Motivation and Requirements

On the one hand, the literature related to workflow process changes concentrates on ensuring the correctness of the changed workflow processes [29]–[31], where the focus is mainly on the preciseness of task sequences or structural errors. On the other hand, the preservation of workflow process consistency is the goal in [32] and [33], where the purpose is to examine the persistence of changed workflow processes considering availability and reliability. However, nowadays, the IIoT-supported business processes are characterized by frequent changes during their life cycle, which have an impact on the management of workflows. Du *et al.* [34] highlighted that fast-changing business environments and their workflows bring many challenges for the workflow execution and proposed an off-line approach using temporal constraints that impacts during the design stage. They identified that systems must adapt and respond dynamically to changes throughout the life cycle. Therefore, the ultimate goal is to provide a solution that can operate in real-time circumstances and can adapt to changes on-the-fly, such as the substitution of a given resource (occupied) by another available. The approach presented in this article especially emphasizes this expectation, demonstrating how the workflow can dynamically adapt to unexpected events. The availability of resources (devices, machines, assembly lines, and so on) and the ability to share them in real-time are one of the most important requirements to support IIoT business processes based on services. Therefore, a framework is needed to manage these processes and resources according to the following requirements.

1) *Requirements at the Enterprise Level:*
   a) To provide a high-level graphical vision of workflows for easy understanding and monitoring.
   b) To provide a universal language for easy communication.
   c) The technique modeling processes must be easily implemented and deployed in real scenarios.
2) *Requirements at Production Level:*
   a) To represent complex processes involving distributed systems and production flows.
   b) To accommodate timing constraints and enable hierarchical composition.
3) *Requirements at the Framework Level:*
   a) To accommodate standard modeling techniques.
   b) To provide a single framework for the implementation, execution, and monitoring of processes.
   c) To maintain the hierarchy for enterprise and production levels.
   d) To provide tools for the agile and dynamic construction of workflows.
   e) To enable the automatic deployment of workflows.
   f) To enable business process changes during the whole lifecycle.
   g) To enable the sharing of resources (reallocation and substitution) in real time.
   h) To enable interoperability and integrability for heterogeneous systems.
   i) To enable a service-oriented-driven architecture guaranteeing adaptable, loosely coupled, and late-bound services.

### B. Overall Architecture

To address these requirements during the whole life cycle of the workflow process, we propose an architecture that supports the usage of the previously described modeling languages, namely BPMN and CPN, in a common infrastructure. Many solutions have been introduced during the years about how to translate BPMN into CPN and vice versa [35]–[37], or even how CPN can verify new BPMN-based approaches [38].

However, the purpose of this article is not to introduce yet another solution, such as the mentioned ones, but to show how the two languages can be used together on a common platform. On the enterprise level, BPMN is easy to understand and enables monitoring of processes. Furthermore, it is easy to create and edit new processes. On the contrary, it is not suitable for modeling complex manufacturing processes [39]—among other reasons because of incompleteness on the performance for: 1) multiple, parallel starting events; 2) exception handling for concurrent subprocess instances; 3) more complex gateways such as OR-Join; and 4) process instance completion [40]. BPMN is developed for modeling business processes and satisfies the requirements at the enterprise level but not at the production level. In contrast, CPN is suitable for production modeling, but it is more difficult to understand and use at the enterprise level. Although there are solutions for translation between CPN and BPMN, a single solution does not satisfy business processes at both workflow levels. Therefore, the task is to find a solution that can combine these two approaches and offers a satisfying option for enterprise and production levels at the same time. Our solution provides an arrowhead support system (workflow choreographer) that builds models at both levels from a recipe. This design document includes the sequence of tasks for the enterprise process and the details (e.g., services involved, the sequence of resources, and so on) to build the production-level processes. The workflow choreographer converts the recipe in a BPMN model (for the first part) and a CPN code (in the form of classes) for each of the production-level classes. At the transition phase (deployment), our approach proposes to deploy both modeling techniques, BPMN at the enterprise level and CPN at the production line level over a single workflow engine provided by an ESB. The hierarchical relation between both levels is maintained during execution. The architecture is complemented using the arrowhead framework, which enables the connection of loosely coupled systems at both levels in an SOA. Plant-independent resources and services are discovered and accessed through arrowhead at the production level. The enterprise-level services are also available using arrowhead.

The operation phase is supported by the arrowhead framework providing service management through its core systems and workflow management using a new element, the workflow choreographer and furthermore monitoring the established workflows through the WSO2 Business Process Server (BPS) and Explorer platforms. The elements of the architecture and the methodology to model and implement processes are described in Sections V and VI.

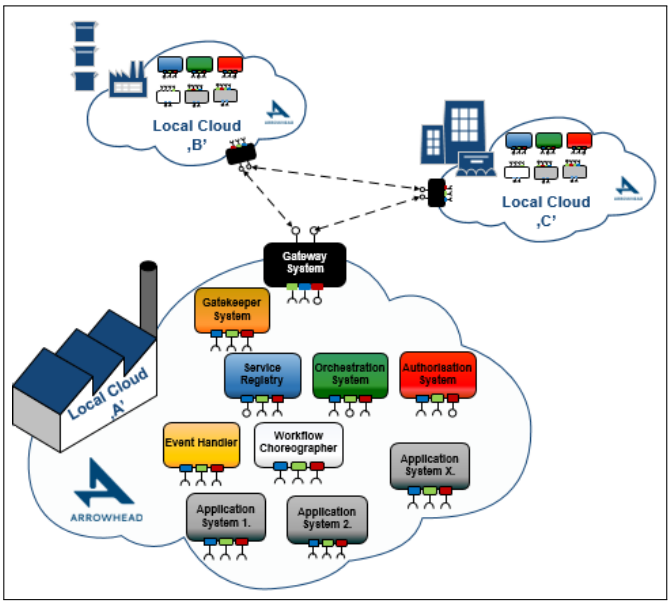


Fig. 2. Generic example of using intercloud servicing within arrowhead.

## IV. Arrowhead Framework

System architects of the IIoT have been looking for methods and frameworks that enable interoperability and integrability of heterogeneous systems in the last decade. The arrowhead framework [11] has been designed and developed for satisfying precisely those needs, supporting effective and secure service sharing through the SOA approach.

The arrowhead framework introduced the concept of local automation clouds [41], providing security, engineering simplicity, and real-time communication for application systems (see Fig. 2). Each local cloud has three mandatory core systems (service registry, orchestration system, and authorization system) to enable essential SOA and security functions. Arrowhead provides supporting core systems as well for intercloud servicing (through the gatekeeper and gateway systems) [42], homogeneous event handling (event handler), different protocol mapping (translator system) [43], and many more [44]. One of the main principles of arrowhead is technology independence. Nevertheless, the current reference implementation of the mentioned components is already widely used [45], which is open source, written in Java.

Workflow management is also supported through data-driven [10] concepts as well as separated workflow manager and executor entities [46]. Arrowhead allows to implement and maintain local cloud [41] specific systems as well, which are the custom applications, CPSs of the cloud. To increase its applicability, the homogeneous integration concept of how the framework can support enterprise and production levels is presented in this article.

The overall model benefits arrowhead due to its SOA approach—it supports service lookup, together with loose coupling and late binding of the service provider and consumer systems, resulting in dynamic service interactions. Since the presented approach also integrates the BPMN and CPN-based task execution using an ESB, the overall model also serves as a proof of concept for arrowhead flexibility.

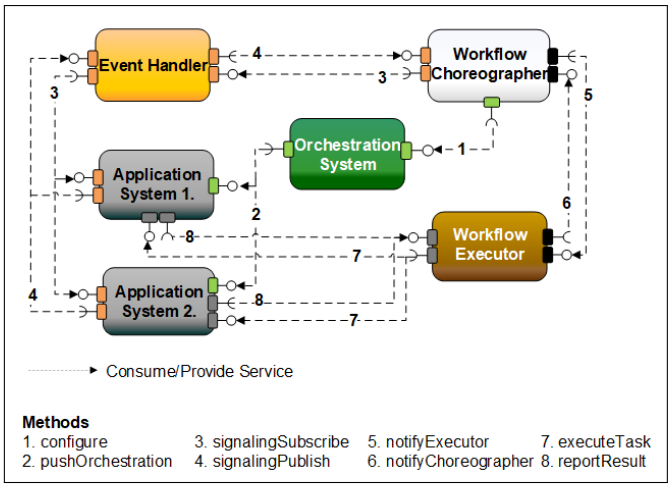


Fig. 3. Simplified service exchange for workflow choreography [10].

### A. Requirements of Digital Production in the Arrowhead

Now, the development of the arrowhead framework is in that stage where the requirements of the automated production must be satisfied. While addressing automated production issues through the arrowhead framework, several alternatives came alive. These approaches defined the critical elements of the automated production, i.e., implemented following an SOA [44], using distributed systems and resources [46] and based on predefined but dynamically changing state machines [10], [47]. Considering the proposals, Section IV-B introduces the proposed new supporting core system, namely the workflow choreographer, which combines the mentioned concepts.

### B. Workflow Choreographer for Supporting Automated Production Within the Arrowhead

The workflow choreographer [10] is the main engine of automated production within the arrowhead framework. From the production point of view, it processes the production order and accordingly implements the production recipe [46]. The industrial world has diverse scenarios; therefore, it is not possible to define a universal workflow manager. Considering that, the proposal is that the workflow choreographer must execute workflows based on predefined templates according to the specific procedures. The production recipe uses these templates, and based on the production steps, the workflow choreographer creates the WS specific workflow executors (WEs), which are the real production accomplishments of the WSs. It executes the related activities and tasks according to the production recipe, which will be given to WSs for performing modifications on goods. Based on the production recipe, the devices and systems of a WS will be coordinated to complement the related workflow step. From the arrowhead point of view, the workflow choreographer sends requests to the orchestration system regarding the services to be used, and it subscribes to the related events, which are reported by the event handler, and it manages the allocated application systems of WEs accordingly (see Fig. 3).

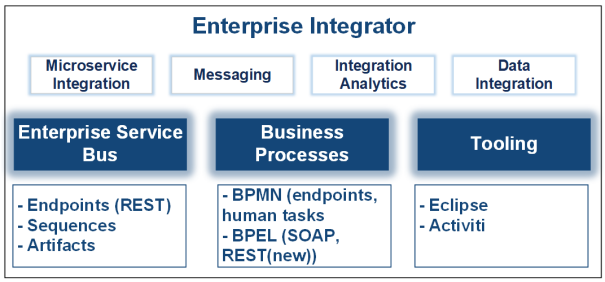


Fig. 4. Architecture of the WSO2 enterprise integrator.

## V. Enterprise Service Bus and Its Integration

The ESB [9] is a software architecture model used for designing and implementing the collaboration and communication between mutually interacting software applications and components in an SOA [48]. The ESB motivation comes from the need to find a standard, structured and general concept for describing the implementation of loosely coupled services that are expected to be independently deployed, running, heterogeneous and disparate within a network. The main functional areas for an ESB are as follows.

1) *Architecture:* Including error tolerance, scalability, performance, as well as hierarchical composition and extensibility and the ability to communicate with other ESBs.
2) *Change and Control:* Service monitoring, lifecycle management, and ESB supporting tools.
3) *Connection:* It provides a wide range of support for most messaging standards and communication protocols.
4) *Mediation:* It dynamic provision, discovery, and management of resources.
5) *Orchestration:* Lightweight orchestration and support for BPEL and BPMN.

### A. WSO2 Enterprise Integrator Architecture

Our solution proposes the usage of an ESB named WSO2 (Web Services Oxygenated 2) [49].

WSO2 EI (WSO2 Enterprise Integrator) is a generic open-source platform that makes possible the integration of different applications, systems, or even data. It enables enterprise services to collaborate dynamically between SOA-based systems. The WSO2 EI block architecture is outlined in Fig. 4.

Out of the modules offered by WSO2 EI, we propose to use the following ones.

1) *Enterprise Service Bus*: The enterprise integration capabilities enable the creation of service endpoints, sequences of services, and software artifacts to fully deploy a service architecture and integrate other SOA-based frameworks such as arrowhead.
2) *Business Process and BPMN Explorer Server:* It enables us to easily deploy business processes written in BPMN or BPEL and also serves as the business process management and hosting environment for SOA systems. It is supported by the Activiti engine [50] and Apache Orchestration Director Engine (ODE). On the BPMN Explorer server, the current status of workflows can be monitored.

3) *Tooling:* The WSO2 Developer Studio is an Eclipse plugin that enables the modeling of processes using BPMN or BPEL. Models are created using the graphical interfaces provided by the tool and generate XML files. The plugin enables to compile processes for Activiti.

### B. Using ESB Together With the Arrowhead Framework

Both the arrowhead framework and the ESB are SOA-based, and the features offered by ESB meet arrowhead's expectations. Thus, the following components of ESB have been used during integration.

*1) Providing APIs for Arrowhead Service Consumption:* Artifacts and endpoints have been constructed to consume services offered by the arrowhead framework. These services are related to service registration and orchestration.

*2) Running Business Processes:* The business process engine has been used to deploy and run processes designed in BPMN or BPEL. Embed the recipes for business process parts at enterprise and production levels.

*3) Supporting Container-Based Solutions:* There are already directions suggesting container-based workflow management [51], which is also possible with the WSO2 EI provided infrastructure.

The ESB can provide a lightweight orchestration of services and BPEL, BPMN support. For creating BPMN diagrams, Activiti has been used. Activiti also makes possible the definition of BPMN service tasks using Java classes. WSO2 workflow engines can interpret and execute such models. In order to be able to run the two modeling alternatives at different levels over the same platform, we need to perform some adaptations. WSO2 ESB can implement and run BPMN models, but it cannot process CPN. The CPN model needs to be compiled and converted into a form that can be handled by the WSO2 ESB. As a starting point, the CPN models can be exported as XML code. Besides, the BPMN processes can be converted into XML format as well, which is an essential basis for integration [52]. Since the Activiti engine allows defining service tasks using Java classes and embeds them into the BPMN model as service tasks, we propose to transform the production-level processes written in CPN into Java code that includes the tasks to be performed in the Java class. Then, the obtained Java classes have been deployed into the BPMN processes using WSO2 ESB. In general, the production steps are different, which means that the CPN models should be designed accordingly. Manufacturing processes are varied and always company-specific; therefore, production recipes need to be created in advance. The CPN models represent those production recipes at the production level, and consequently, they must be prepared before the construction of enterprise workflows. These models can be used as templates when the workflow choreographer creates the production-specific workflows based on the production order.

## VI. Use Case: Automotive Assembly Plant

This section presents a use case example for an automotive production system—considered as system of systems powered by the arrowhead framework. This production system runs business processes, and the workflow choreographer manages their execution.

### A. Company-Specific Production Systems

The company-specific production system is called XPS [53], where "X" denotes the company. XPS is a strategic development program, which ensures consistency and stability of development. It creates a common development language that facilitates communication between plants and helps the sharing of information and resources efficiently.

The XPS includes the assembly plants [54], which can usually be divided into four major units, so-called shops, shown in Fig. 5.

1) *Press Shop:* Here, the various body elements are manufactured, e.g., doors, pillars, roofs, and many more elements.
2) *Body Shop:* The first stage in car assembling, where body parts are mounted, using various techniques, such as welding, laser brazing, riveting, clinching, or bonding.
3) *Paint Shop:* Where the car body elements will be painted, including processes from the chemically clean parts to the coating and required repairs.
   item *Final Assembly Shop:* The automobile will be finished here. Chassis subassemblies and trim, e.g., glass, doors, seats, electronics, and other optional parts, are applied to the body.

Except for the press shop, the shops should be in one plant, but the paint shop must be separated from the others because of the purity, processing, and airing problems. Besides, there is another department where cars are repaired. Because manufacturing is usually faster in certain sections, high-capacity storage buffers are located between the shops; however, it is not shown explicitly in Fig. 5.

### B. Assembly Flow Used as an Example

In this use case, the focus is on the final assembly shop and its WSs, which involves many devices or systems (different types of robots), executing specific activities with a common goal. From the standard processes, three production steps are taken and described.

The workflow consists of the following tasks.

1) *Tank Installation:* The conveyor belt moves to WS14 with the smart product (Transport service), where welding and assembling services are consumed, and the tasks are assigned to the robots of this WS. The services are available, and the tasks will be executed. At the end of the process, the workflow choreographer will be notified (notify service) (this is the default assumption for the next steps as well).
2) *Seat Installation:* The conveyor belt moves to the WS15 (transport service), where the welding, assembling, and trimming services are consumed, and the tasks are assigned to WS robots. Meanwhile, the welding robot of this WS crashes; therefore, another welding robot is needed to take over the job. The WS17 welding robot is reconfigured and redirected (transport service) to do the job. At the end of the process, the redirected welding

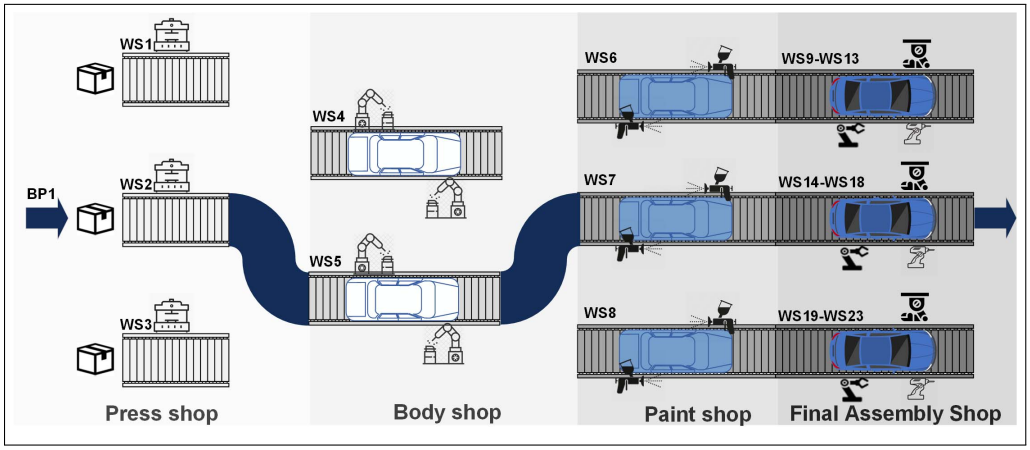


Fig. 5. Typical automotive assembly plant: the example use case of Section VI-B refers to the production steps of final assembly shop.

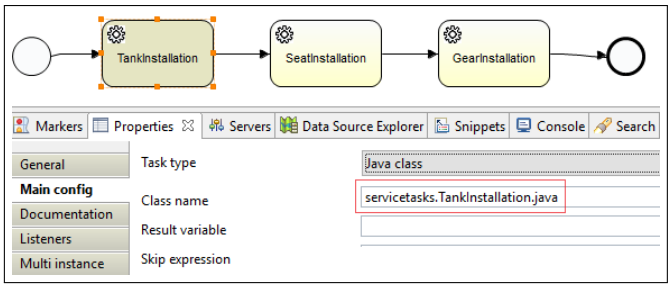


Fig. 6. Modeling the use case in BPMN with Activiti.

robot will be moved back to its original place, and it will be reconfigured again for WS17 tasks.

3) *Gear Installation:* The conveyor belt moves to the WS16 (transport service), where welding, assembling, and nutrunner services are consumed, and the tasks are assigned to the WS robots.

### C. Workflow Choreographer at Enterprise Level

The input of the workflow choreographer is the production order that should be first preprocessed. It is complex and bound documentation that describes the product specification. Based on this, the workflow choreographer creates the production recipe, which will precisely contain the related workflow steps and the appointed systems. The production recipe can be dynamically changed by the workflow choreographer during the production, according to the available resources and unexpected events. Using this, the workflow choreographer allocates the tasks to the appropriate WSs with the necessary production configuration.

Fig. 6 shows a part of the production steps in the final assembly shop. Unlike the production-level execution shown in Fig. 7, the BPMN model merely shows the high-level workflow steps that will be executed, although it provides enough information about the current state of production on the enterprise level. According to the use case, the production will start only when the smart product arrives at WS14. The production-specific workflows are defined and coded in the visible BPMN Service Task objects whose logic is described in Section VI-D.

### D. Workflow Choreographer at Production Level

When the workflow choreographer has defined the enterprise -level workflow, the created workflow steps will be filled with the related production logic based on the production recipe. From the workflow choreographer point of view, these production steps are considered as WEs, i.e., the WE is encapsulated in the workflow choreographer. Implementationwise, the workflow choreographer sees the WE as a simple service task (BPMN notation) that implements a Java class. This class holds the production recipe represented as a CPN that interacts with the arrowhead framework for resource allocation, service provisioning, and other production-related activities. When the task is executed, the WE reports it to the workflow choreographer. Fig. 7 shows the production-level logic.

Using the opportunities provided by CPN tools, the hierarchical composition can be seen on top. For simulating the interaction between the mentioned elements, three main parts are defined from the hierarchical point of view: arrowhead mandatory core systems, workflow choreographer, and WEs.

The arrowhead mandatory core systems are the authorization system, service registry, and orchestration system (through service discovery). The workflow choreographer manages the sequence of service consumption for the WEs, and it can reconfigure the reserved devices for the WEs according to the current step. To keep it simple, the WE contains only the basic logic of the current WS where a service is used from an application system. Here, the workflow steps of the specific business processes can be found (e.g., tank installation)—together with the corresponding actions such as when, how long, and which robots should work on the smart product during this process.

In Fig. 7, it holds the following.

1) The workflow choreographer sends a *serviceRequest()* to the orchestration system, which will ideally offer a provider who can give the requested service.
2) The allocated provider will be reported by the event handler. Until the workflow choreographer receives the notification about the allocated service, it will wait.
3) Then, the workflow choreographer examines the offered provider, which can be any systems, robots, and so on.

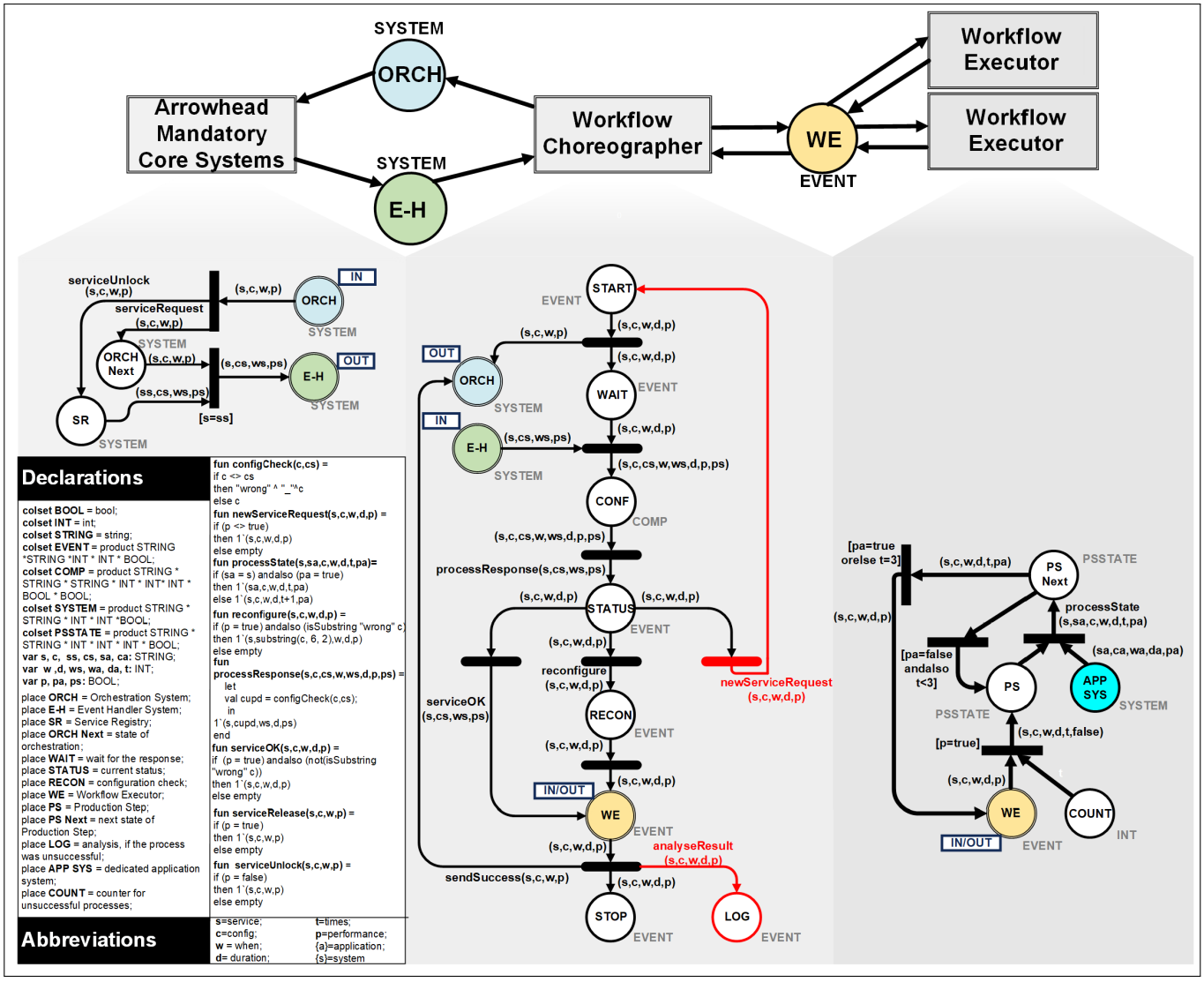


Fig. 7. Model represents the initiative CPN model of workflow choreographer (in the middle) and shows roughly the interaction with mandatory Arrowhead systems (left) and a possible example WE (right).

In the first step, the workflow choreographer checks the configuration of the allocated provider. Here, three scenarios can be considered.

a) If the provider is currently configured for another task, then the workflow choreographer first makes a software update on the robot, which is basically a preconfigured patch with the regarding task execution logic.
b) For some reason, the provider is not able to complete the task; therefore, the workflow choreographer must send a *newServiceRequest()* to the orchestration system and 1–3 points will be executed again.
c) The provider has the appropriate configuration.

4) Then, the provider will be passed to the regarding WE, where the production level workflow is executed synchronized with other possible systems, robots, and so on at the current WS.
5) According to the prewritten CPN logic of the production level, the task will be executed by the WE, where the current states of the production is continuously evaluated *(processState())*.
6) The WEs will report the statuses according to the events, which can be a successor, in case of unexpected events, can be a failure.
7) The workflow choreographer analyses the results *(analyseResult())*, and based on, it brings the proper decisions. Here are two relevant possibilities.
   a) If the result is a failure, then this has to be analyzed. The current model does not represent the possible scenarios.
   b) If the result is a success, then the workflow choreographer will give back the reserved service, i.e., *sendSuccess()* message to the orchestration system, which can interpret this and will make the used provider free *(serviceUnlock())*.

## VII. Discussion on Implementation

During the creation of the workflow engine, various approaches for solution integration were considered, and as

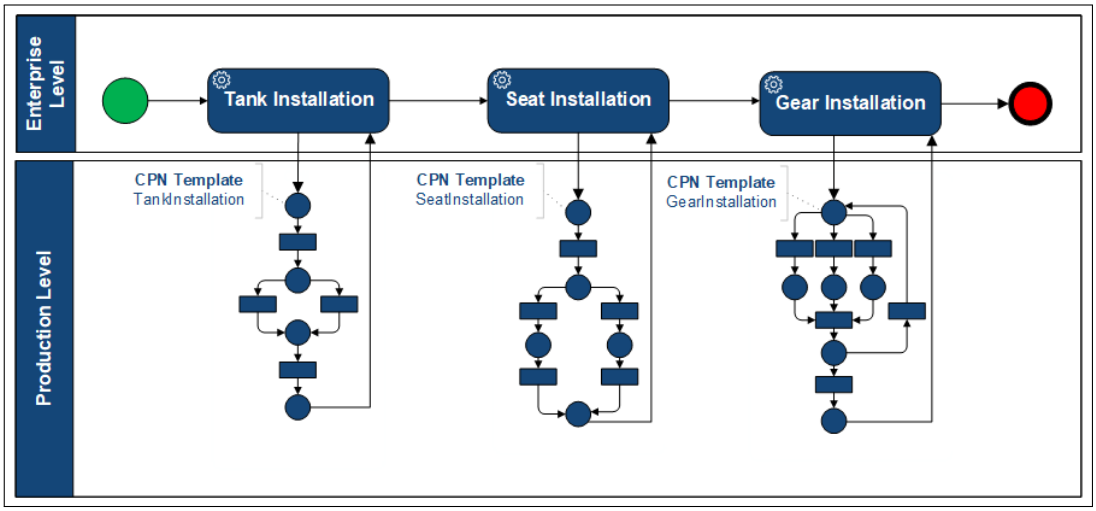


Fig. 8. Integration of the BPMN model on enterprise level and the CPN templates on production level. The execution order is managed by the workflow choreographer.

always, finding the optimal solution depends on the application area. Considering the requirements of Industry 4.0 concerning production, we aimed at providing a feasible solution both for business and production participants [47].

Combining the two modeling languages (BPMN and CPN) at the two levels (enterprise and production) is shown in Fig. 8. Here, transparent business workflow elements include verified and CPN-based production WEs.

Behind the BPMN tasks, a template-based and verified CPN model can be found, which ensures the correctness of the particular production step. These templates are simple although company- and production-specific, hence, have to be created by the competent expert for the given product.

### A. Reference Implementation of the Multilevel Workflow

In our reference implementation of the workflow model, as it can be seen in Fig. 1 in the transition and operation phases and their subprocesses, the models at enterprise and production levels are developed by using Eclipse Java EE IDE, including the necessary plugins to model and deploy processes. In order to provide dynamically adaptable workflows, automated code generation is required. We used the JavaPoet [55], which is an open-source library that provides APIs for creating the Java source code. Section VII-B describes in detail the main steps of the workflow generation.

*1) Workflow Model at Enterprise Level:* Based on the production order, which is a JSON document [47], the workflow choreographer generates the respective production recipe and, based on it, the BPMN model automatically. To achieve this, we use the eclipse plugin version of the Activiti process engine developed by Alfresco [50], which is the leading lightweight, Java-centric open-source BPMN engine supporting real-world process automation needs. The workflow choreographer identifies the steps and services from the production recipe, and it creates the enterprise-level workflow dynamically with the appropriate steps. These are the high-level workflow steps that define the functional description of WEs at the same time. In the current implementation, we use the Activiti engine service tasks function where—among others—Java classes can be passed as input of the service tasks. This is the central pillar of the implementation because the production-level logic has then got implemented into these Java classes.

*2) Workflow Model at Production Level:* The created steps of the enterprise-level workflow must be filled with the appropriate production-specific logic based on the prewritten CPN Templates. These steps are the WEs from the production point of view. To use the features provided by Activiti, the CPN logic must be converted automatically *(with JavaPoet)* into Java code—however, the presentation of the mapping algorithm is one of the further works. The following options can be considered for compiling CPN into running (in our implementation: Java) code.

1) Export CPN logic as XML and create a compiler that will generate BPMN service tasks [37] or BPEL model [56].
2) Automatically generated pseudocode based on the SML logic from CPN Tools that will be converted into Java-code, which will be implemented as BPMN service tasks or BPEL model;
3) Combining the two solutions and based on the generated XML of CPN Tools, creating pseudocode that will be converted into Java-code, which will be used in the BPMN Service Task or BPEL model.

### B. Deployment of the Artifacts

When the workflow is generated, it is ready to be delivered. For this, first, we identified the mandatory artifacts of WSO2. Here, the first step is the creation of a composite application project (CAR). The CAR includes the service task artifacts of WEs as dependences and enables the selection of the WSO2 BPS where the workflow will run. The second step is the creation of the BPMN graph based on the logic of the production recipe. The result after implementing the software leaves to main archives.

1) A *.bar* zip file that holds the enterprise-level model implemented in BPMN. This process is represented in XML.

```
AssemblyShopTask_CAR_1.0.0.car
├── AssemblyShopTask_Artifact_1.0.0
│   ├── artifact.xml
│   └── AssemblyShopTask_Artifact-1.0.0.jar
│       ├── servicetasks
│       │   ├── TankInstallation.class
│       │   ├── SeatInstallation.class
│       │   └── GearInstallation.class
│       └── META-INF
│           └── MANIFEST.MF
└── artifacts.xml
```

Fig. 9. Tree structure of CAR artifact.

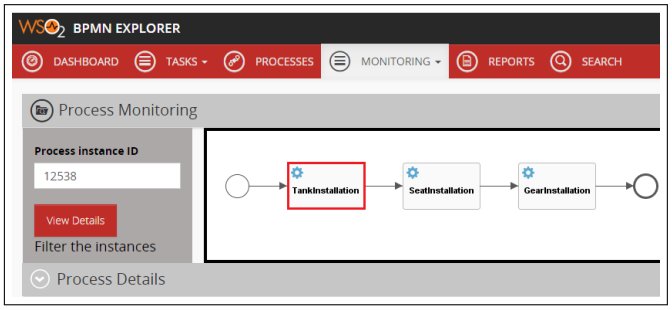


Fig. 10. Monitoring the workflow with BPMN explorer.

2) A *.car* zip file that holds the WE artifacts. This file holds an XML file (artifacts.xml) indicating that the archive holds a carbon-type application, runs on a BPS, and indicates which is the artifact that holds the jar files and its version. The artifact is also included in the *.car* zip file. This artifact includes the.jar files holding the classes used in the service tasks built with BPMN. The structure of the *.car* file of the current case is shown in Fig. 9.

These two files can be deployed onto the WSO2 infrastructure, available in our experiment environment. However, regarding the delivery, two kinds of solutions can be used.

1) *Manually:* Uploading the *.bar* and *.car* files using the management console utility provided in the WSO2 BPS web interface.
2) *Automatically:* There are several ways [49] to deploy CAR files onto the environment, e.g., using the Maven Deploy feature of WSO2 Developer Studio, by which it will be sent directly to the server.

Our approach adds further functionality to the workflow choreographer by enabling the automatic deployment of the zip files. The workflow choreographer collects the *.bar* and *.car* files produced after the design stage (development) and transfers them to the regarding folders of the WSO2 BPS where the workflow process will be executed.

1) The path of the *.car* file: "$CARBON_HOME/repository/deployment/server/carbonapps."
2) The path of the *.bar* file: "$CARBON_HOME/repository/deployment/server/bpmn."

After the delivery, the WSO2 ESB runs and monitors the processes by using the BPMN explorer utility, as it is shown in Fig. 10.

## VIII. Performance Measurement of the Proof of Concept

In order to get an objective view of the previously presented automated deployment, it is crucial to measure the efficiency of workflow generation based on specific parameters. In this case, two factors are worth examining. First, how the number of BPMN objects (in our case, the WEs) affects the generating times of a workflow, and second, how long the service consumption and provision processes will last.

TABLE I
Deployment Times Depending on the BPMN Objects

| BPMN Objects [number] | Slowest [ms] | Fastest [ms] | Average [ms] |
|---|---|---|---|
| **1** | 2046 | 165 | **255** |
| **5** | 2311 | 525 | **770** |
| **10** | 3078 | 962 | **1175** |
| **25** | 4379 | 2217 | **2520** |
| **50** | 6794 | 4459 | **4779** |
| **75** | 9078 | 6369 | **6785** |
| **100** | 11543 | 8527 | **8964** |
| **200** | 20234 | 16832 | **17384** |
| **300** | 28865 | 25256 | **25873** |
| **400** | 41154 | 37407 | **38095** |
| **500** | 50184 | 46455 | **47468** |
| **600** | 60722 | 56955 | **57873** |
| **700** | 69496 | 66101 | **67348** |
| **800** | 79126 | 75608 | **76660** |
| **900** | 116212 | 83616 | **88426** |
| **1000** | 207579 | 95602 | **119130** |

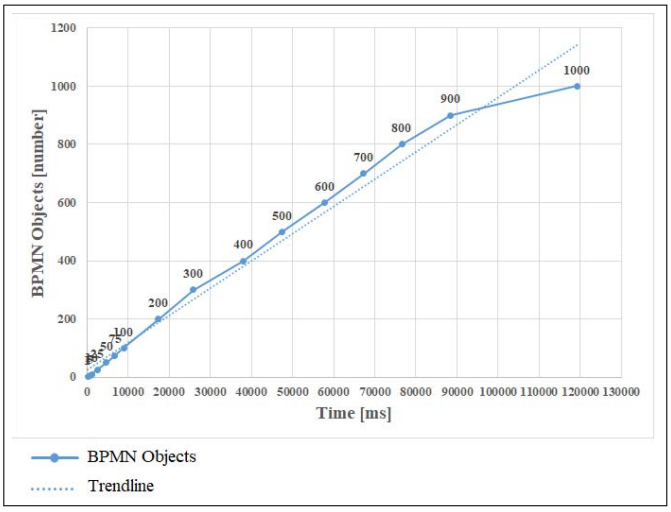


Fig. 11. Average deployment time relative to the number of BPMN objects.

### A. Performance Test of Workflow Generation

Regarding the first factor, Table I presents the information about the slowest, the fastest, and the average deployment time (given in ms) to produce BPMN objects (given in number). One thousand runs were made for each studied BPMN object number. The tests were run on a machine with Intel Core i7-8650U CPU 1.90 GHz (4 cores), 16 GB of memory, and Microsoft Windows 10 x64.

Based on the results, the diagram of the average deployment time is shown in Fig. 11, where it can be shown that the increasing number of BPMN objects is linear with time. This is as expected. However, this measurement was needed to demonstrate how long a possible update will take compared with the number of BPMN objects under real-time circumstances.

TABLE II
SERVICE CONSUMPTION TIMES

| | Slowest [ms] | Fastest [ms] | Median [ms] | Average [ms] |
|---|---|---|---|---|
| **WSO2 BPS + Arrowhead** | 344 | 32 | 54 | **61** |
| **Java Client + Arrowhead** | 252 | 26 | 47 | **52** |

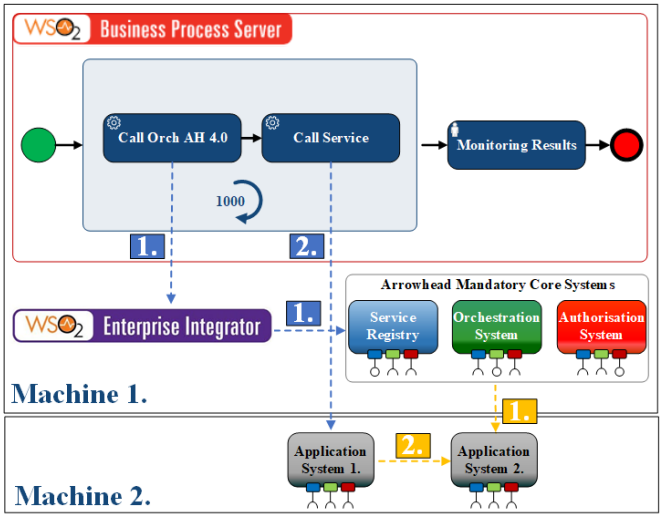


Fig. 12. Architecture of service-related performance tests.

### B. Performance Test of Service-Related Processes

Regarding the second factor, the goal is to estimate the additional load introduced by the WSO2 BPS in comparison with a simple service consumption through the arrowhead framework. The objective is to measure the increase in latency introduced when the processes generated by the workflow choreographer consume services. Note that it is using the APIs constructed in the ESB to access the arrowhead framework. The test environment composition is shown in Fig. 12 with the following subparts.

1) *Machine 1:* VMWare ESXi Virtual Machine with 2 GB memory, 16 GB disk, and Ubuntu Server 16.04.5. The following components run on this machine:
   a) WSO2 Enterprise Integrator 6.5.0 (Docker);
   b) WSO2 BPS 6.4.0 (Docker);
   c) arrowhead framework 4.0 (Core Systems).
2) *Machine 2:* Intel Core i5-4200M CPU 2.50 GHz, 8 GB Memory, and Microsoft Windows 10 x64. The following components run on this machine:
   a) service application provider written in Java;
   b) service application consumer, written in Java.

Using this architecture, two types of tests were executed—for which the results are summarized in Table II.

The first one measures the time incurred by several serial calls from the Java Application Client to the Java Application Provider using the arrowhead framework (orange arrows in Fig. 12). The requests are placed after each other; the total time for all calls is measured. Fig. 12 shows the time needed for 1000 serial calls. The performance of Java Client and arrowhead (orange) is shown in Fig. 13. Each call implies a call to the orchestration system to obtain the endpoint for the final service offered by the Service Application Provider 1) and the actual consumption of that service 2).

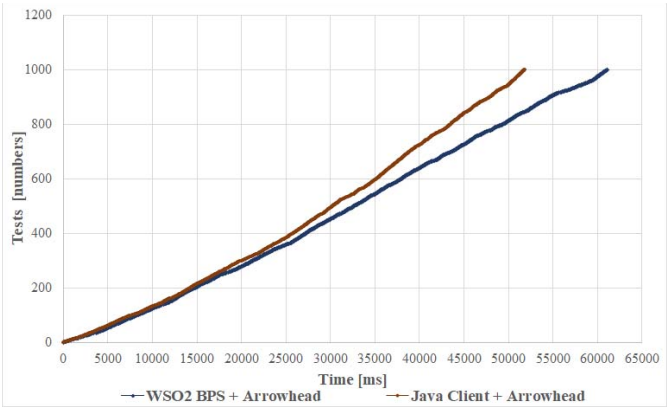


Fig. 13. Results of the service-related performance tests.

The second test considers a simple business process [see Fig. 12 (top)] that invokes the arrowhead adaptation API deployed in the enterprise integrator, which in turn redirects the call to the arrowhead orchestrator service to obtain the final service endpoint (arrows in blue). Once the endpoint is available, the process calls on to the service offered by the service application provider.

Fig. 13 shows the comparison between 1000 serial calls using the BPS, the WSO2 EI API, and the arrowhead framework to those of just directly using the arrowhead framework. On average, the WSO2 introduces only 9-ms delay in a regular service discovery consumption sequence. The results also show a total delay of approximately 9 s at the end of the 1000 iterations, which agrees with the previous average value, and bearable for the users.

### C. Performance Evaluation

In the case of the presented scenario in Section VI-B, where in the second step (seat installation), the welding robot is crashed, the other welding robot can be allocated approximately in 368 ms, summing up the measured average values (see Tables I and II). In this case, one WE (BPMN object) of the three needs to be updated with one new service. The performance test results provide a reasonable basis for designing automated workflows using the described technologies. Furthermore, even the response time to unexpected events can be easily estimated by considering the identified metrics. Noticeably, the delay from the WSO2 architecture cannot be reduced because it is a particular and noncontrollable component of this composition. However, in the final implementation, the WSO2 architecture will be used at the enterprise level and not at the production level where those unexpected events take place; therefore, the additional time introduced by WSO2 will not impact the performance, which will allow for faster service consumption and end-to-end deployment time.

## IX. CONCLUSION

The novelty of this article is the integration of dynamic task executions at the enterprise and production levels through actual workflow management models and tools (such as ESB, BPMN, CPN, and arrowhead), enabling efficient and flexible workflow management even in heterogeneous and dynamically

changing environments. The overall method translates to operative, executable code, due to the available toolset. While there are many well-known methodologies for workflow modeling in the industrial fields, Industry 4.0 brought new requirements, which must be considered and fulfilled in the next generation of manufacturing.

Accepting the challenge, this article proposes a workflow execution model that integrates information and control flows at different workflow levels. The goal is to provide a solution that is transparent and easy to manage at the enterprise level and can also verify and execute the workflow at the production level. To achieve this, in the first step, the requirements of the related workflow levels were identified. For the enterprise level, the recommended modeling language is BPMN, and for the production level, it is CPN. This article provides a generic overview of the two chosen modeling languages, identifying their advantages and disadvantages. The presented combination of the models satisfies the requirements of both levels. The further novelty of this article is to show how the arrowhead framework can assist in automated production while fitting in the abovementioned approach. In line with the identified requirements of workflow levels, expectations for the framework level have also been defined. Based on the described criteria, we introduce a new workflow-engine supporting system, namely the workflow choreographer, managing the parallel execution processes at the WSs. In order to show an alternative to a fully arrowhead-driven implementation, as proof of concept, another SOA compatible open-source solution, the WSO2 ESB is integrated with the arrowhead framework. Using these together, we verified and validated the workflow choreographer through an example from the automotive industry.

Regarding future work, the prototype of the arrowhead-specific workflow choreographer has already been released based on the presented proof of concept. Therefore, the goal is to test, develop, and fine-tune the system until it is suitable for operation in a real industrial environment.

## REFERENCES


[1] S. Jeschke, C. Brecher, T. Meisen, D. Özdemir, and T. Eschert, "Industrial Internet of Things and cyber manufacturing systems," in *Industrial Internet of Things*. Cham, Switzerland: Springer, 2017, pp. 3–19.

[2] J. Lee, B. Bagheri, and H.-A. Kao, "A cyber-physical systems architecture for industry 4.0-based manufacturing systems," *Manuf. Lett.*, vol. 3, pp. 18–23, Jan. 2015.

[3] Y. Lu, "Industry 4.0: A survey on technologies, applications and open research issues," *J. Ind. Inf. Integr.*, vol. 6, pp. 1–10, Jun. 2017.

[4] A. Bicaku, C. Schmittner, P. Rottmann, M. Tauber, and J. Delsing, "Security safety and organizational standard compliance in cyber physical systems," *Infocommun. J.*, vol. 11, p. 2, Mar. 2019.

[5] S. Mittal, M. A. Khan, D. Romero, and T. Wuest, "Smart manufacturing: Characteristics, technologies and enabling factors," *Proc. Inst. Mech. Eng., B, J. Eng. Manuf.*, vol. 233, no. 5, pp. 1342–1361, Apr. 2019.

[6] T. Erl, *Service-Oriented Architecture*. London, U.K.: Pearson, 2005.

[7] S. A. White, *Introduction to BPMN*. Armonk, NY, USA: IBM, 2004.

[8] T. Murata, "Petri nets: Properties, analysis and applications," *Proc. IEEE*, vol. 77, no. 4, pp. 541–580, Apr. 1989.

[9] D. A. Chappell, *Enterprise Service Bus*. Newton, MA, USA: O'Reilly Media, 2004.

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

[11] 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.

[12] A. Nabiollahi and S. bin Sahibuddin, "Considering service strategy in ITIL v3 as a framework for IT governance," in *Proc. Int. Symp. Inf. Technol.*, Aug. 2008, pp. 1–6.

[13] 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. Inform. (INDIN)*, Jul. 2019, pp. 126–131.

[14] W. Du, J. W. Davis, and M.-C. Shan, "System and method for enterprise workflow resource management," U.S. Patent 6 308 163, Oct. 23, 2001.

[15] J. Saives, G. Faraut, and J.-J. Lesage, "Automated partitioning of concurrent discrete-event systems for distributed behavioral identification," *IEEE Trans. Autom. Sci. Eng.*, vol. 15, no. 2, pp. 832–841, Apr. 2018.

[16] R. Ferreira da Silva, R. Filgueira, I. Pietri, M. Jiang, R. Sakellariou, and E. Deelman, "A characterization of workflow management systems for extreme-scale applications," *Future Gener. Comput. Syst.*, vol. 75, pp. 228–238, Oct. 2017.

[17] M. Reichert, "Enabling flexible and robust business process automation for the agile enterprise," in *The Essence of Software Engineering*. Cham, Switzerland: Springer, 2018, pp. 203–220.

[18] H. A. Reijers, I. Vanderfeesten, and W. M. P. van der Aalst, "The effectiveness of workflow management systems: A longitudinal study," *Int. J. Inf. Manage.*, vol. 36, no. 1, pp. 126–141, Feb. 2016.

[19] A. Krishna, P. Poizat, and G. Salaün, "Checking business process evolution," *Sci. Comput. Program.*, vol. 170, pp. 1–26, Jan. 2019.

[20] N. Brouns, S. Tata, H. Ludwig, E. S. Asensio, and P. Grefen, "Modeling iot-aware business processes-a state of the art report," IBM Res. Division, San Jose, CA, USA, Tech. Rep. RJ 10540, 2018.

[21] D. Domingos and F. Martins, "Using BPMN to model Internet of Things behavior within business process," *Int. J. Inf. Syst. Project Manage.*, vol. 5, no. 4, pp. 39–51, 2017.

[22] J. Mass, C. Chang, and S. N. Srirama, "Workflow model distribution or code distribution? Ideal approach for service composition of the Internet of Things," in *Proc. IEEE Int. Conf. Services Comput. (SCC)*, Jun. 2016, pp. 649–656.

[23] M. Havey, *Essential Business Process Modeling*. Newton, MA, USA: O'Reilly Media, 2005.

[24] K. Baina and S. Baina, "User experience-based evaluation of open source workflow systems: The cases of Bonita, Activiti, jBPM, and Intalio," in *Proc. 3rd Int. Symp. ISKO-Maghreb*, Nov. 2013, pp. 1–8.

[25] K. Jensen, W. M. Van der Aalst, G. Balbo, M. Koutny, and K. Wolf, *Transactions on Petri Nets and Other Models of Concurrency VII*, vol. 7480. Berlin, Germany: Springer-Verlag, 2013.

[26] S. Christensen, L. M. Kristensen, and T. Mailund, "Condensed state spaces for timed Petri nets," in *Proc. Int. Conf. Appl. Theory Petri Nets*. Berlin, Germany: Springer-Verlag, 2001, pp. 101–120.

[27] K. Jensen, L. M. Kristensen, and L. Wells, "Coloured Petri nets and CPN tools for modelling and validation of concurrent systems," *Int. J. Softw. Tools for Technol. Transf.*, vol. 9, nos. 3–4, pp. 213–254, May 2007.

[28] K. Jensen and G. Rozenberg, *High-Level Petri Nets: Theory and Application*. Berlin, Germany: Springer-Verlag, 2012.

[29] D. Habhouba, S. Cherkaoui, and A. Desrochers, "Decision-making assistance in engineering-change management process," *IEEE Trans. Syst., Man, Cybern., C (Appl. Rev.)*, vol. 41, no. 3, pp. 344–349, May 2011.

[30] N. Ahmad, D. C. Wynn, and P. J. Clarkson, "Change impact on a product and its redesign process: A tool for knowledge capture and reuse," *Res. Eng. Des.*, vol. 24, no. 3, pp. 219–244, Jul. 2013.

[31] P. Sun and C. Jiang, "Analysis of workflow dynamic changes based on Petri net," *Inf. Softw. Technol.*, vol. 51, no. 2, pp. 284–292, Feb. 2009.

[32] M. L. Rosa, W. M. Van Der Aalst, M. Dumas, and F. P. Milani, "Business process variability modeling: A survey," *ACM Comput. Surv.*, vol. 50, no. 1, p. 2, 2017.

[33] W. Song, X. Ma, S. C. Cheung, H. Hu, and J. Lü, "Preserving data flow correctness in process adaptation," in *Proc. IEEE Int. Conf. Services Comput.*, Jul. 2010, pp. 9–16.

[34] Y. Du, B. Yang, and H. Hu, "Incremental analysis of temporal constraints for concurrent workflow processes with dynamic changes," *IEEE Trans. Ind. Informat.*, vol. 15, no. 5, pp. 2617–2627, May 2019.

[35] C. Dechsupa, W. Vatanawood, and A. Thongtak, "Hierarchical verification for the BPMN design model using state space analysis," *IEEE Access*, vol. 7, pp. 16795–16815, 2019.

[36] M. Ibrahim, "Formal semantics of BPMN process models using CPN," *IREIT. J*, vol. 5, no. 3, pp. 1–7, 2017.

[37] C. Dechsupa, W. Vatanawood, and A. Thongtak, "Transformation of the BPMN design model into a colored Petri net using the partitioning approach," *IEEE Access*, vol. 6, pp. 38421–38436, 2018.
[38] Y. Ji, H. Sun, X. Liu, J. Zeng, and S. Bai, "A decentralized framework for executing composite services based on BPMN," in *Proc. Comput. World: Future Comput., Service Comput., Cognit., Adapt., Content, Patterns*, Nov. 2009, pp. 332–338.
[39] J. Lenhard, V. Ferme, S. Harrer, M. Geiger, and C. Pautasso, "Lessons learned from evaluating workflow management systems," in *Proc. Int. Conf. Service-Oriented Comput.* Málaga, Spain: Springer, 2017, pp. 215–227.
[40] R. M. Dijkman, M. Dumas, and C. Ouyang, "Formal semantics and analysis of BPMN process models using Petri nets," Queensland Univ. Technol., Brisbane, QLD, Australia, Tech. Rep. 7115, 2007, pp. 1–30.
[41] J. Delsing, J. Eliasson, J. van Deventer, H. Derhamy, and P. Varga, "Enabling IoT automation using local clouds," in *Proc. IEEE 3rd World Forum Internet Things (WF-IoT)*, Dec. 2016, pp. 502–507.
[42] C. Hegedus, D. Kozma, G. Soos, and P. Varga, "Enhancements of the arrowhead framework to refine inter-cloud service interactions," in *Proc. 42nd Annu. Conf. IEEE Ind. Electron. Soc. (IECON)*, Florence, Italy, Oct. 2016, pp. 5259–5264.
[43] H. Derhamy, J. Eliasson, J. Delsing, P. P. Pereira, and P. Varga, "Translation error handling for multi-protocol SOA systems," in *Proc. IEEE 20th Conf. Emerg. Technol. Factory Autom. (ETFA)*, Sep. 2015, pp. 1–8.
[44] J. Delsing *et al.*, "Arrowhead framework core systems and services," in *IoT Automation: Arrowhead Framework*. Boca Raton, FL, USA: CRC Press, 2017, ch. 4.
[45] *Arrowhead Consortia*. Accessed: Jun. 16, 2020. [Online]. Available: https://github.com/arrowhead-f/core-java-spring
[46] H. Derhamy, M. Andersson, J. Eliasson, and J. Delsing, "Workflow management for edge driven manufacturing systems," in *Proc. IEEE Ind. Cyber-Physical Syst. (ICPS)*, May 2018, pp. 774–779.
[47] D. Kozma, P. Varga, and F. Larrinaga, "Data-driven workflow management by utilising BPMN and CPN in IIoT systems with the arrowhead framework," in *Proc. 24th IEEE Int. Conf. Emerg. Technol. Factory Autom. (ETFA)*, Sep. 2019, pp. 385–392.
[48] M. Keen *et al.*, "Patterns: Implementing an SOA using an enterprise service bus," *IBM Redbooks*, vol. 336, pp. 20–28, Jul. 2004.
[49] K. Indrasiri, *Beginning WSO2 ESB*. New York, NY, USA: Apress, 2016.
[50] T. Rademakers, *Activiti Action: Executable Bus. Processes BPMN 2.0*. Shelter Island, NY, USA: Manning Publications, 2012.
[51] T. Goldschmidt, S. Hauck-Stattelmann, S. Malakuti, and S. Grüner, "Container-based architecture for flexible industrial control applications," *J. Syst. Archit.*, vol. 84, pp. 28–36, Mar. 2018.
[52] R. Shiraki and Y. Shinkawa, "Verification of business processes with time constraints," in *Proc. 6th IIAI Int. Congr. Adv. Appl. Informat. (IIAI-AAI)*, Jul. 2017, pp. 72–75.
[53] T. H. Netland, "Company-specific production systems: Managing production improvement in global firms," Ph.D. dissertation, NTNU, Trondheim, Norway, 2013, p. 340.
[54] G. Michalos, S. Makris, N. Papakostas, D. Mourtzis, and G. Chryssolouris, "Automotive assembly technologies review: Challenges and outlook for a flexible and adaptive approach," *CIRP J. Manuf. Sci. Technol.*, vol. 2, no. 2, pp. 81–91, Jan. 2010.
[55] *Introduction to JavaPoet*, Baeldung, Bucharest, Romania, 2019.
[56] K. B. Lassen and W. M. Van der Aalst, "WorkflowNet2BPEL4WS: A tool for translating unstructured workflow processes to readable BPEL," in *Proc. OTM Confederated Int. Conf. 'Move Meaningful Internet Syst.'* Berlin, Germany: Springer-Verlag, 2006, pp. 127–144.

**Dániel Kozma** received the M.Sc. degree in electrical engineering from the Budapest University of Technology and Economics (BME), Budapest, Hungary, in 2015, where he is currently pursuing the Ph.D. degree.

He was an experienced database software engineer with a demonstrated history of working in telecommunications. He is working currently as an Information Security Officer. In parallel, he is also a Researcher with BME within the Productive 4.0 project. His research focuses on the different areas of Industry 4.0, such as automated production, supply chain and lifecycle management, cybersecurity, and information security.

**Pál Varga** (Member, IEEE) received the M.Sc. and Ph.D. degrees from the Budapest University of Technology and Economics (BME), Budapest, Hungary, in 1997 and 2011, respectively.

He is currently an Associate Professor with BME. He is also the Director of the Telecommunications Division, AITIA International Inc., Budapest. His main research interests include communication systems, network performance measurements, root cause analysis, fault localization, traffic classification, end-to-end QoS and SLA issues, as well as hardware acceleration and Industrial Internet of Things.

**Felix Larrinaga** received the B.Sc. degree (Hons.) in industrial electronics and the M.Sc. degree in computing engineering from Mondragon University, Mondragon, Spain, in 1995 and 2010, respectively, and the Ph.D. degree in mobile communications from Staffordshire University, Stoke-on-Trent, U.K., in 1999.

He is currently a Lecturer and a Researcher with Mondragon University, Basque, Spain. Since 2000, he has been lecturing and researching at Mondragon University with extensive experience in industrial work as a Project Leader. His research interests include web engineering, interoperability, social web technologies, and semantic web.