Data-driven Workflow Management by utilising BPMN and CPN in IIoT Systems with the Arrowhead Framework

Dániel Kozma, Pál Varga
Dept. of Telecommunications and Media Informatics
Budapest University of Technology and Economics
2 Magyar Tudósok krt., Budapest, Hungary, H-1117
Email: {kozma, pvarga}@tmit.bme.hu

Felix Larrinaga
Mondragon Unibertsitatea
Goiru kalea 2, Arrasate-Mondragon,
Gipuzkoa, Spain, E-20500
Email: flarrinaga@mondragon.edu

***Abstract*—Workflow management is realised in manufacturing at the Enterprise- and Production (workstation) levels. The characteristics of business processes in these levels differ enough to prevent the adoption of a conventional modelling or implementation solution. The conceptualisation of an adequate and adaptable workflow management model that over-arches the heterogeneous business process levels is challenging. However, one of the encouragements of Industry 4.0 is to provide easy-to-use models and solutions that enable the effective implementation of production goals.**

The current paper addresses this challenge and – as a proof-of-concept – it demonstrates how the goals mentioned above can be achieved by combining the different manufacturing process models. To accomplish this, a workflow control engine needs to be designed and standardised respectively. Arrowhead is an IIoT (Industrial IoT) framework that dynamically and flexibly supports automated manufacturing processes following Industry 4.0 expectations. This paper describes how its workflow management system, i.e., the Workflow Choreographer implements automated production. Furthermore, it details what the functions of this system are and how it applies and combines BPMN and CPN in practice to provide a solution for the given challenge.

To verify its feasibility, the paper showcases a demo application of the concept as well.

***Keywords*—Digital Production, Coloured Petri Nets, BPMN, Productive 4.0, Industry 4.0**

I. Introduction

Workflow management has always played a vital role in all areas of production. Modelling, optimising processes and automation are the driving forces of the next generation production. Industry 4.0 sets requirements for current industrial solutions and targets full automation. The task to be solved is complex. The implementation of Industry 4.0 in many cases is not just a one-generation industrial change, but very often, the manufacturers are in the Industry 2.0 phase. Managing and integrating legacy systems in the new Internet of Things (IoT) era is, on the one hand, a challenge, and on the other hand, it raises many questions.

To clarify these issues, the Productive 4.0 [1] research project was created in response to the Industry 4.0 problem-setting. Productive 4.0 focuses on three main domains where current industrial solutions need to evolve: Digital Production (DP), Supply Chain Networks (SCN) and Product Lifecycle Management (PLM). This article describes the DP advances in the project which are built upon the Arrowhead framework, hence addressing other Industrial IoT (IIoT) requirements, such as real-time operation, security, and adopting the principles of Service Oriented Architectures (SOA). The paper also describes the concept of the multi-level workflow execution and demonstrates the draft implementation of the Arrowhead's workflow engine, the Workflow Choreographer. The draft implementation only serves as a proof-of-concept, a verification of the final application. The development and operation of the workflow engine presented here could be completely vendor-specific, but its Arrowhead-compatibility helps fitting it into the complete (enterprise- or production level) workflow-management picture.

The structure of the article is as follows. Section II discusses the related work. Section III briefly summarises the architecture of the Arrowhead framework. Section IV describes in detail the automated workflow management engine of the Arrowhead framework, namely the Workflow Choreographer. Section V demonstrates the reference implementation of the concept. Finally, Section VI concludes the paper.

II. Related Work

Due to the Industry 4.0 movement, modelling business processes is a current "hot" topic that has been a trigger for many kinds of research in recent years. Based on the multi-level workflow concept [2], processes must be divided into two parts for industrial production: Enterprise (EL) and Production level (PL). EL workflows overcome high-level processes that are easy-to-understand and use. For this purpose, the Business Process Modelling and Notation (BPMN) was selected, which is the most widely used business process modelling language. However, BPMN is not able to provide a comprehensive solution that covers all possible cases at the PL. So at this level, we use Coloured Petri Net (CPN) as the modelling language. The following subsections give a brief description of these languages and describe the most important aspects of them.

A. Brief overview of BPMN

BPMN provides an intuitive and straightforward method for non-expert users to model business processes. It has smooth semantics that helps in understanding enterprise-level processes and reduces misunderstandings from communication and modelling. BPMN is designed to model a wide range of areas and enables end-to-end business processes to be created. The structural elements of BPMN allow the user to distinguish easily between various aspects of the BPMN diagram. With BPMN, a company can gain many benefits through a clear description and understanding of its processes. Due to the prevalence of the language, BPMN models can be created by many tools. Using the language and the corresponding tools provide the following benefits:

- BPMN is suitable for a variety of process modelling, and it can be flexibly used for current purposes and process visualisation.
- Graphical elements can be expanded with variables, making process analysis and execution more efficient.
- The process models are easy to understand because BPMN has simple but powerful semantics.
- The BPMN model language is expandable with company-specific notations.
- It also supports the modelling of event and error handling.

The graphical description provided by the BPMN specification has the following components:

- Events: start, intermediate, end;
- Activities:
  - Tasks: service, user, script, mail, receive, business;
  - Multiple instances;
  - Sub-processes;
  - Loop.
- Gateways: exclusive, inclusive, parallel, event-based and complex;
- Data and Flows: data object, association, sequence-, default- and message flow.

As it was mentioned above, there are several possibilities for creating and executing BPMN workflows. The most common process engines currently are Activiti, jBPM and Bonita BPM [3].

B. Brief overview of Coloured Petri Nets

CPN [4] has been developed mainly for systems where communication, synchronisation or resource sharing play an essential role. The CPN includes the strengths of traditional Petri Net, which provides primitives for modelling process interaction, and it is extended by functional programming language which offers primitives to define data types and manipulate data values. The CPN model consists of modules (pages), and it can contain – such its predecessor, the Petri Net – places, transitions and arcs. The modules can connect through well-defined interfaces, similar to many modern programming languages. The graphical representation is also available in case of CPN, which makes the basic structure of the complex CPN model easier to edit and to assess the impact of processes as well. There are many tools for designing CPN models. The most current CPN modelling program is the CPN Tools [5]. The used functional programming language in CPN Tools is the so-called Standard Meta Language (SML).

C. Concept of BPMN and CPN together

Many solutions have been introduced during the years about how to translate BPMN into high-level Petri Nets and vice-versa [6]–[9]. However, the purpose of this article is not to present yet another solution, 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 modelling complex manufacturing processes [10] – among other reasons because of incompleteness on the performance for [11]:

- multiple, parallel starting events;
- exception handling for concurrent sub-process instances;
- more complex gateways like OR-Join;
- process instance completion.

BPMN is developed for modelling business processes, hence satisfies the requirements at EL but not at the PL. CPN is suitable for production modelling, but it is harder to understand. 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 EL and PL at the same time. Our approach proposes [2], [12] to use two modelling techniques; BPMN at the EL and CPN at PL with a single workflow engine provided by an Enterprise Service Bus (ESB). The architecture is complemented using the Arrowhead framework, which enables the connection of loosely coupled systems at both levels in an SOA. The elements of the architecture and the methodology to model and implement processes are described in the following sections.

III. Arrowhead framework

The Arrowhead framework [13] has originally come to life to cover interoperability and integration issues for the IIoT world. It supports the collaboration of newly built as well as legacy CPS architectures based on the principles of SOA through applying the System-of-Systems (SoS) approach. One part of the above-listed issues are tackled through the Local Cloud concept [14] empowered by inter-cloud communication capabilities [15]. Each stakeholder has their local cloud(s), working as an SoS, their systems implement either intra- or inter-cloud information sharing, as well as security- and other policies. The Arrowhead framework defines mandatory core systems for the local clouds, which provide the necessary functionality. Further, supportive core systems provide general services that are often needed in System of Systems, so integrators do not have to implement their solutions for such common services. The Application Systems are distinct elements of the SoS, these provide (and in fact, consume)

the various application services – in a discoverable, late-bound, loosely coupled way that is defined by the SOA. Figure 1. describes the current core systems of the Arrowhead framework.

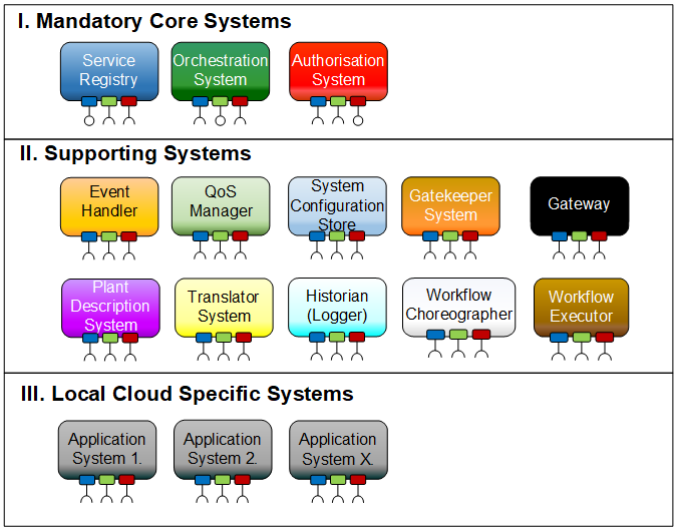

Fig. 1. The core systems of the Arrowhead framework

In brief, the mandatory core systems [16] are the Orchestration System (mainly for service discovery and late binding), Service Registry (so services providers can announce their active services), and Authorisation System (to provide Authorisation and Authentication). Further, supporting systems are provided by the Gateway [17] and Gatekeeper Systems [15] for inter-cloud communication (data and control plane, respectively), the Workflow Choreographer [12] (to trigger the next step in the process execution), the Event Handler [18] (to circulate status and event information), and the Plant Description System [19] (to keep track of SoS- or Plant-related meta-data), among others. The local cloud-specific systems can be the local systems; from the smallest sensor up to the biggest Cyber-Physical Systems (CPS). In this paper, the Workflow Choreographer and its features are described in detail.

IV. The Workflow Choreographer and its Environment

In order to manage industrial workflows (from Enterprise Level to Production Level; from manufacturing to logistics), not only models, semantics and description languages are needed, but actual entities that handle, manage, execute the workflow. Depending on the application area, the dynamics of the production plant, and even external constraints, the management and execution of the workflows can face various challenges – which can be described as requirements and restraints.

Based on the analysis of current Industry 4.0 requirements at actual plants, Derhamy et al. [20] suggested the separation of ”workflow management” tasks (retrieving process steps and processing results) and ”workflow execution” (issuing job, progress tracking) tasks on the factory floor. As part of a research parallel to this, a Workflow Choreographer [12] has been suggested that enables simultaneous, dynamic workflow processing controlled by recipes described in Coloured Petri Nets. The current paper merges these concepts with the Enterprise Level workflow handling – by ESB.

The Workflow Choreographer [12] is an engine controlling automated production within the Arrowhead framework. From the production point of view, the features presented by Figure 2 are provided by the engine. It processes the Production Order (PO) and accordingly implements the Production Recipe (PR) [20]. There are extensive requirements within the various industrial domains; therefore, it is challenging to define one universal workflow manager. Taking this into account, the proposal is that the Workflow Choreographer must execute workflows based on predefined templates according to the specific procedures. The PR uses these templates, and based on the production steps; the Workflow Choreographer instantiates the workstation specific Workflow Executors (WE), which are the real production accomplishments of the workstations. Each WEs execute the related activities and tasks according to the PR, which will be given to workstations for performing modifications on goods. Based on the PR, the devices and systems of a workstation will be coordinated to complement the related workflow step. From the AF point of view, the Workflow Choreographer pushes 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.

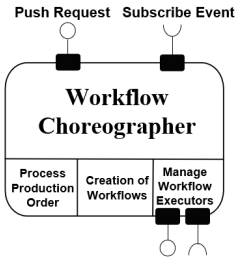

Fig. 2. Main services and features of the Workflow Choreographer

A. Process the Production Order

The Workflow Choreographer gets the Production Order from the Enterprise Resource Planning (ERP) system – and based on that, it creates the Production Recipe. The Production Order is company-specific; therefore, the Workflow Choreographer must implement the PR accordingly. The established PR contains only the tasks to be executed with the regarding services and products specific parameters.

B. Creation of the Workflows

Based on the PR, the Workflow Choreographer creates the EL and PL workflows, respectively, as Figure 3 shows. The EL workflow contains the main steps of the production written in BPMN and these steps are filled with the pre-written CPN template, i.e. PL logic according to the task to be executed.

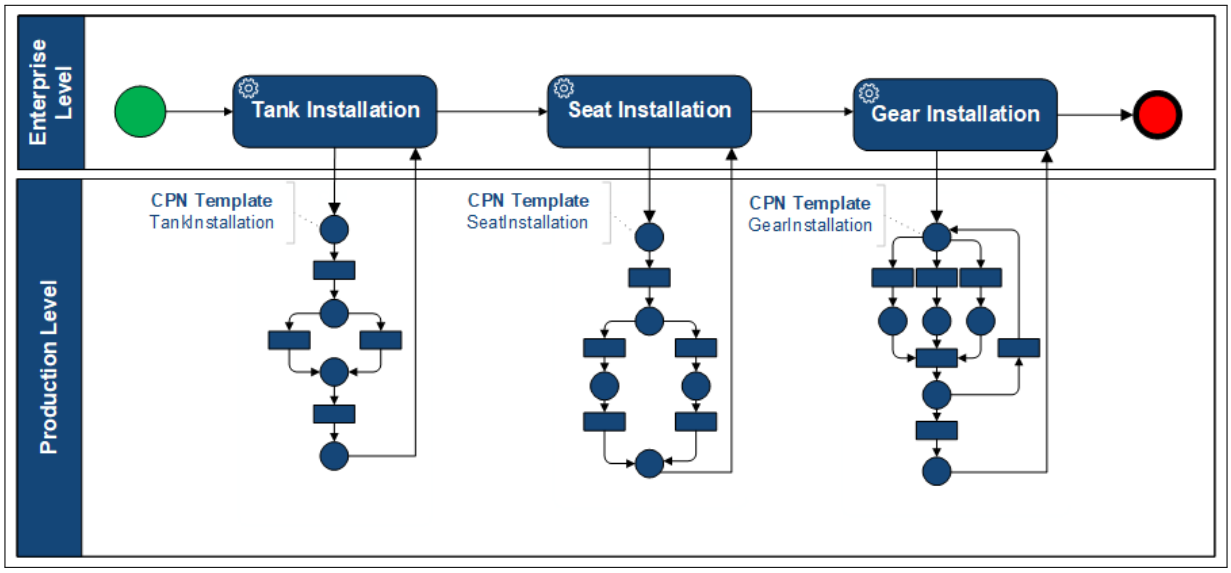

Fig. 3. Concept of the workflow levels – and their model description

C. Communication within the Arrowhead cloud

The Workflow Choreographer needs to oversee how the current workflow is executed, what is the status of the processes. This engine is preferably data-driven and must be capable of error detection and handling. As it can be seen on Figure 4., the engine has to rely on the other core systems of Arrowhead – mostly on the Orchestration System and the Event Handler [18]. The latter is there to facilitate event notification propagation and monitoring over the network, while the former is to re-assign interactions between application systems. The Orchestration System interface can be used in a ”push” manner, as an option. This means that the application systems might receive new, unsolicited orchestration information. This enables for strong central government and allows for the execution of common goals. During its operation, the Workflow Choreographer exploits this kind of ”Orchestration Push” service. When the services are reserved, the Workflow Choreographer will be informed by the Event Handler, and it will prepare the related WEs. Then the WEs communicate with the allocated application systems and informs the Workflow Choreographer about the associated events. A simple message sequence is shown in Figure 5.

D. Managing the Workflow Executors

Based on the PR, the Workflow Choreographer allocates the appropriate Workflow Executors. The basis of the executors' functional description is the pre-written template (our suggestion: in CPN) and contains the detailed workflow steps for that executor. The pre-written template is filled out (i.e., configured) based on the Production Recipe, for the given step.

In general, the workstations are strictly defined points within a factory, where mostly the same tasks are executed. However, in the next generation of manufacturing, the allocation of robots at the workstations can be changed dynamically. For example, if more resource is needed at a workstation because

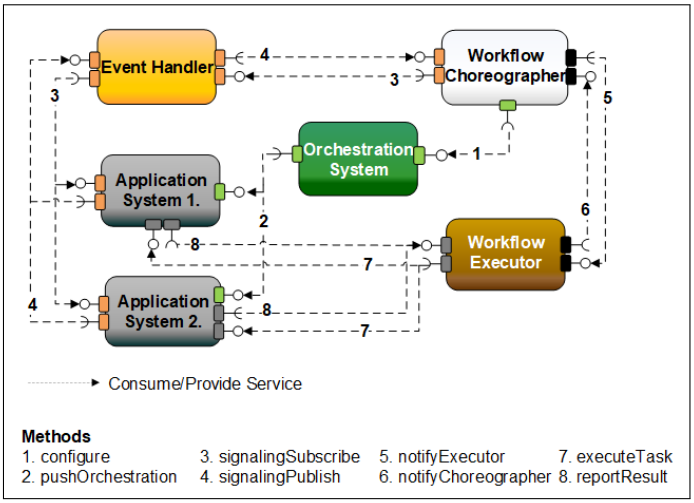

Fig. 4. Workflow Choreographer and its environment within Arrowhead

of unexpected events such as break down, another robot moves there to take over the task. Thus, it is a must to ensure that the Workflow Executors can always be allocated for a given job based on the PR.

The PR contains the detailed PL logic at the workstation (based on the CPN template) – which task will be executed, which robots will be used and how long, etc. Then the Workflow Executors are connected to the allocated robots – in the Arrowhead context it is an Application System providing a service –, and they will control the robots respectively. The Workflow Executors continuously report the events of the production. Based on the results, the Workflow Choreographer makes smart decisions, and if it is necessary, it also re-configures the existing Workflow Executors. When the production is finished at the workstation, the Workflow Choreographer releases the related, reserved services, as well.

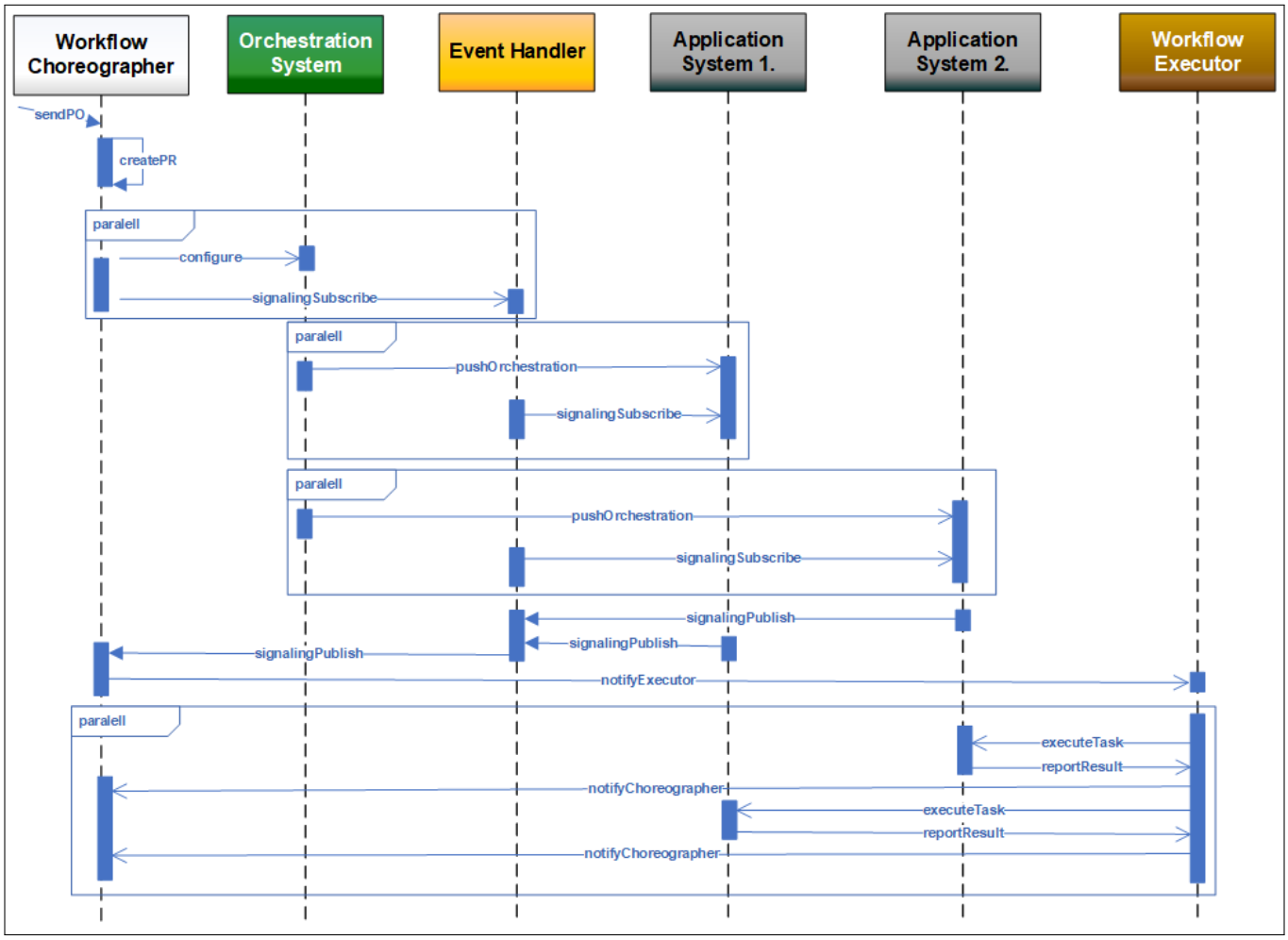

Fig. 5. This figure presents a general workflow execution message flow within the local Arrowhead cloud. If a Production Order arrives,
1) The Workflow Choreographer interprets it and use its createPR() feature.
2) It consumes (in parallel) a *configure()* service from the Orchestration System for the services to be used and it consumes a *signalingSubscribe()* service from the Event handler for the reserved services respectively.
3) In parallel, the Orchestration System sends *pushOrchestration()* requests to the appropriate Application Systems (1-2.) while the Event Handler subscribes on the related events accordingly – *signalingSubscribe()*.
4) The reserved Application Systems send *signalingPublish()* messages to the Event Handler when the orchestration is done, indicating that they are ready to be used.
5) The Event Handler provides *signalingPublish()* service about the reserved Application Systems to the Workflow Choreographer.
6) The Workflow Choreographer provides *notifyExecutor()* service to the Workflow Executor, and allocates the reserved Application Systems and the related workflow logic as well.
7) The Workflow Executor sends *executeTask()* service request to the Application Systems which provide *reportResult()* service to the Workflow Executor about the events. Then the Workflow Executor provides *notifyChoreographer()* to the Workflow Choreographer about the performance.

V. Reference Implementation of the Multi-Level Workflow Model

The implementation of the features described in the previous sections is divided into several parts. This paper, however, focusing on presenting the most significant part of them:

- Creation of Workflows: both at PL and EL;
- Managing the Workflow Executors: instantiate the executors based on the PR;
- Communication within the Arrowhead cloud: service reservation and subscription on the related events.

To model the processes, we have created a demo PR that can be produced by the Workflow Choreographer from the PO. The PR template consists of the following elements:

- PriorityId: This number identifies the order of priority.
- DocumentVersion: Optional. This applies to specific production schemes, for example, productions which often occurs with minor differences.
- Plant: This attribute recognises the plant where the workflow will be executed.
- Steps: Each step identifies a Workflow Executor.
  - BpmnType: Referring to the BPMN task type.
  - FlowId: The number id representation of the task.
  - ProductionStep: The name of the Workflow Executor.

- PreviousStepId: Identifier of the previous Workflow Executor – which is equal to the previous step's FlowId.
- NextStepId: Identifier of the next Workflow Executor – which is equal to the next step's FlowId.
- Services: The list of the services to be used by the current Workflow Executor.

To put this work into an automotive-industrial use case: let us focus on the final car assembly shop and some of its workstations. This involves many devices or systems (different types of robots), executing specific activities with a common goal. The complete use-case is described within [2] – where it was put into the Arrowhead context, as well. The related PR documentation is the following:

```
{
  "PriorityId": 1,
  "DocumentVersion": 1,
  "Plant": "AssemblyShop",
  "Steps": [
    {
      "BpmnType": "servicetask",
      "FlowId": 1,
      "ProductionStep": "TankInstallation",
      "PreviousStepId": "start",
      "NextStepId": "2",
      "Services": {
         "Service1": "Transport",
         "Service2": "Welding",
         "Service3": "Assembling"
      }
    },
       {
      "BpmnType": "servicetask",
      "FlowId": 2,
      "ProductionStep": "SeatInstallation",
      "PreviousStepId": "1",
      "NextStepId": "3",
      "Services": {
         "Service1": "Transport",
         "Service2": "Welding",
         "Service3": "Welding",
         "Service4": "Assembling"
      }
    },
    {
      "BpmnType": "servicetask",
      "FlowId": 3,
      "ProductionStep": "GearInstallation",
      "PreviousStepId": "2",
      "NextStepId": "end",
      "Services": {
         "Service1": "Transport",
         "Service2": "Welding",
         "Service3": "Assembling",
         "Service4": "Trimming"
      }
  }]
}
```

A. Workflow model implementation at EL and PL

The models at Enterprise and Production Levels are developed by using Eclipse Java EE IDE for Web Developers including the necessary plugins to model and deploy processes, i.e. Activiti, Developer Studio and WSO2. The development of the processes is represented in the following subsections.

1) Workflow model at Enterprise level: Based on the PR, the Workflow Choreographer will generate the respective BPMN model automatically. To achieve this, we use the Activiti process engine developed by Alfresco [21], 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 PR, and it creates the EL workflow dynamically with the steps. These are the high-level workflow steps which define the functional description of Workflow Executors at the same time. In the current implementation, we use the Activiti engine Service Tasks function where – amongst others – Java classes can be passed as input of the Service Tasks. This is the main pillar of the implementation because the Production Level logic will be implemented into these Java classes.

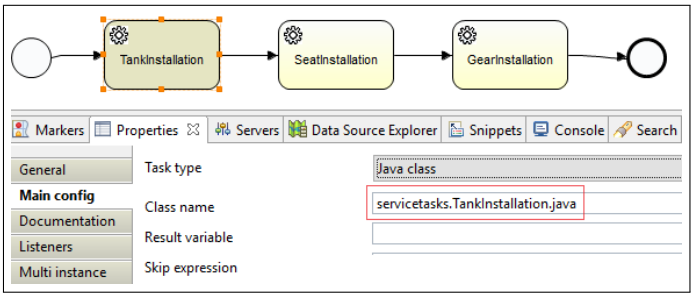

Fig. 6. BPMN model by Workflow Choreographer using the Activiti engine

2) Workflow model at Production level: The created steps of the EL workflow must be filled with the appropriate production-specific logic based on the pre-written 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 into Java code – however, the presentation of the mapping algorithm and the detailed description of the CPN templates are not the subjects of this paper.

For automated code generation, we use the JavaPoet [22] – developed by Square –, an open source library that provides APIs for creating a Java source code. It allows the generation of primitive types, reference types, and their variants, e.g. classes, interfaces, enumerated types, inner classes. Moreover methods, parameters, fields, comments, and even Java doc. JavaPoet automatically manages the import of dependent classes, and its builder can interpret the logic of generating Java code.

The exemplary code using JavaPoet:

```
// WorkflowUtils.java
public void createJavaDelegateClass(String
    packageName, String className)
        throws IOException {

    File directory = new File("src/main/java");
    MethodSpec execute = MethodSpec
          .methodBuilder("execute")
          .addModifiers(Modifier.PUBLIC)
          .returns(void.class)
          .addParameter(DelegateExecution.class,
              "execution")
          /* The CPN logic is implemented here */
```

```
            .build();

        TypeSpec processName =
            TypeSpec.classBuilder(className)
              .addModifiers(Modifier.PUBLIC)
              .addMethod(execute)
              .addSuperinterface(JavaDelegate.class)
              .build();

        JavaFile javaFile =
              JavaFile
              .builder(packageName, processName)
              .build();

        javaFile.writeTo(directory);
    }
}
```

These results are the following, automatically generated Java class regarding the first step (i.e., the Tank Installation) and another Java class, which shows how the instantiated WE can reach and control one of the welding robots.

```
// TankInstallation.java
public class TankInstallation implements
     JavaDelegate {
  public void execute(DelegateExecution execution) {
  WeldingRobot weldingRobot = new WeldingRobot();
   /* The CPN logic is implemented here */
    }
}

// WeldingRobot.java
public class WeldingRobot {
    private WebTarget webTarget;
    private Client client;
    private static final String BASE_URI =
    "http://10.1.2.2:8080/ApplicationSystem1/robots";
    public WeldingRobot() {
        client = ClientBuilder.newClient();
        webTarget = client.target(BASE_URI)
            .path("welding");
    }
    public String getText() throws
         ClientErrorException {
        WebTarget resource = webTarget;
        return resource.request(MediaType.TEXT_PLAIN)
            .get(String.class);
    }
    public void close() {
        client.close();
    }
}
```

B. Deployment of the artefacts

Our solution uses an Enterprise Service Bus (ESB) named Web Services Oxygenated 2 (WSO2) [23]. WSO2 Enterprise Integrator (WSO2 EI) is an open source generic 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 following modules of WSO2 EI are used in the reference implementation:

- Enterprise Service Bus: The enterprise integration capacities allow the creation of service endpoints, sequences of services and software artefacts to completely deploy a service architecture and integrate other SOA based frameworks such as the Arrowhead.
- Business Process Server: Enables to deploy business processes written in BPMN amongst others and also operates as the business process management and hosting environment for SOA systems. It is supported by the Activiti BPMN Engine and BPEL Apache Orchestration Director Engine (ODE).
- Tooling: This module is an Eclipse plugin that facilitates the modelling of processes using BPMN. Models are built using the graphical user interfaces provided by the tool and generate XML files. The plugin enables to compile procedures for Activiti [21].

After the implementation of WEs, the next step is the creation of a Composite Application Project (CAR). The CAR includes the Service Task artefacts of WEs as dependencies and enables the selection of the server where the classes will run (WSO2 Business Process Server in our case). The result after implementing the software leaves to main archives:

- A *.bar* zip file that holds the Enterprise level model implemented in BPMN. This process is represented in XML.
- A *.car* zip file that holds the Workflow Executor artefacts. This file holds an XML file (artefacts.xml) indicating that the archive holds a Carbon type application, runs on a Business Process Server and indicates which is the artefact that holds the jar files and its version. The artefact is also included in the *.car* zip file. This artefact includes the .jar files holding the classes used in the Service Tasks build with BPMN.

These two files are deployed in the WSO2 infrastructure available in our experiment environment. The environment uses the WSO2 Business Process Server where both level models are run. For the delivery, two kinds of solution can be used.

1) Manually: Uploading the *.bar* and *.car* files using the Management Console utility provided in the Business Process Server web interface;

2) Automatically: There are several ways [23] to deploy CAR files in the environment, e.g., using Maven Deploy feature of WSO2 Developer Studio, by which it will be sent directly to the server.

After the delivery, the BPMN process can be run and monitored using the BPMN Explorer utility, as shown by Figure 7.

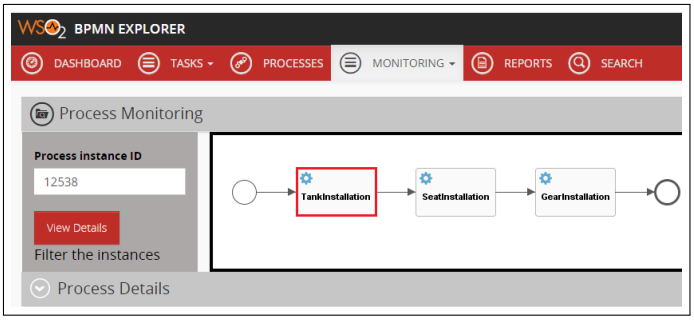

Fig. 7. Monitoring the workflow with BPMN Explorer

VI. Conclusion

Managing workflows has different requirements, toolset, and mindset on the Enterprise and the Production Levels since data is starting to flow into various management systems due to Industry 4.0 efforts. In order to effectively use the production and business data (for feedback and control), this paper proposes to utilise the BPMN-based business process descriptions and CPN-based production-level descriptions. Moreover, this paper demonstrates the feasibility of this merged model through actual practical tools: Enterprise Service Bus (ESB) and CPN tools for the two workflow levels. Furthermore, the whole concept has been fitted into the Arrowhead framework, which provides support for the IIoT System of Systems through its Service Oriented Architecture-based principles, and its core systems.

To provide native support for workflow management both at the Enterprise and the Production Levels, this paper details a newly formed element of the Arrowhead framework, the Workflow Choreographer. This system turns Production Orders into Production Recipes and instantiates Workflow Executors. These latter elements are suggested to execute the (CPN-defined) steps simultaneously, as the incoming production data allows. The actual Application Systems available for providing a given service (e.g. positioning, welding, painting, etc.) are loosely coupled – by the Orchestrator – to the service consumers, as usual in the Arrowhead framework.

As a proof-of-concept, this paper describes a reference implementation for the overall model, including the BPMN-based ESB implementation, the CPN-based production recipe template-elements, and the automatically generated Java execution codes for the Workflow Executors. Altogether, the paper provides validated results that the combined BPMN and CPN workflow description models can provide with a solution in practice for the given challenge by utilising the Arrowhead framework.

Acknowledgement

The project has been developed by the Embedded System Group of MGEP together with the Dept. of Telecommunications and Media Informatics of BME. This work has been supported by the Department of Education, Universities and Research of the Basque Government under the projects Ikerketa Taldeak (Grupo de Sistemas Embebidos) and TEK-INTZE (Elkartek 2018) and the European H2020 research and innovation programme, ECSEL Joint Undertaking, and National Funding Authorities from 19 involved countries under the project Productive 4.0 with grant agreement no. GAP-737459 - 999978918.

References

[1] Productive4.0, "Electronics and ict as enabler for digital industry and optimized supply chain management covering the entire product lifecycle," 2017. [Online]. Available: https://productive40.eu

[2] D. Kozma, P. Varga, and F. Larrinaga, "Multi-level Workflow Management for Automated Production: Integrating BPMN-driven ESB with Coloured Petri Net powered by Arrowhead," *Transactions on Industrial Informatics*, 2019, submitted.

[3] K. Baïna and S. Baïna, "User experience-based evaluation of open source workflow systems: The cases of bonita, activiti, jbpm, and intalio," in *3rd International Symposium ISKO-Maghreb*. IEEE, 2013, pp. 1–8.

[4] K. Jensen and G. Rozenberg, *High-level Petri nets: theory and application*. Springer Science & Business Media, 2012.

[5] Cpn tools 4.0. [Online]. Available: http://cpntools.org/

[6] C. Dechsupa, W. Vatanawood, and A. Thongtak, "Hierarchical verification of the bpmn design model using the state space analysis," *IEEE Access*, vol. PP, pp. 1–1, 01 2019.

[7] M. Ibrahim, "Formal semantics of bpmn process models using cpn," *IREIT. J*, vol. 5, no. 3, 2017.

[8] 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. PP, pp. 1–1, 07 2018.

[9] A. Kheldoun, K. Barkaoui, and M. Ioualalen, "Formal verification of complex business processes based on high-level petri nets," *Information Sciences*, vol. 385, pp. 39–54, 2017.

[10] J. Lenhard, V. Ferme, S. Harrer, M. Geiger, and C. Pautasso, "Lessons learned from evaluating workflow management systems," in *International Conference on Service-Oriented Computing*. Springer, 2017, pp. 215–227.

[11] R. M. Dijkman, M. Dumas, and C. Ouyang, "Formal semantics and analysis of bpmn process models using petri nets," *Queensland University of Technology, Tech. Rep*, pp. 1–30, 2007.

[12] P. Varga, D. Kozma, and C. Hegedús, "Data-driven workflow execution in service oriented iot architectures," in *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, vol. 1. IEEE, 2018, pp. 203–210.

[13] P. Varga, F. Blomstedt, L. L. Ferreira, J. Eliasson, M. Johansson, J. Delsing, and I. M. de Soria, "Making System of Systems Interoperable - the Core Components of the Arrowhead Framework," *Journal of Network and Computer Applications, Special Issue on Engineering Future Interoperable and Open IoT Systems*, 2016.

[14] J. Delsing, J. Eliasson, J. van Deventer, H. Derhamy, and P. Varga, "Enabling IoT Automation using Local Clouds," *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)*, pp. 502–507, 2016.

[15] C. Hegedűs, D. Kozma, G. Soós, and P. Varga, "Enhancements of the arrowhead framework to refine intercloud service interactions," in *42nd Annual Conference of IEEE Industrial Electronics Society (IECON)*, Florence, Italy, 2016.

[16] J. Delsing, *Towards industrial and societal automation and digitisation*. In book: IoT Automation – Chapter 1 – published by CRC Press, 2016.

[17] C. Hegedűs, P. Varga, and A. Franko, "Secure and Trusted Intercloud Communications in the Arrowhead Framework," in *2018 IEEE Industrial Cyber-Physical Systems (ICPS)*, May 2018, pp. 755–760.

[18] M. Albano, L. Ferreira, and J. Sousa, "Event Handler System: Publish / Subscribe communication for the Arrowhead world," in *12th IEEE World Conference on Factory Communication Systems (WFCS)*, 2016.

[19] O. Carlsson, D. Vera, J. Delsing, and B. Ahmad, "Plant Descriptions for Engineering Tool Interoperability," in *14th International Conference on Industrial Informatics*, 2016.

[20] H. Derhamy, M. Andersson, J. Eliasson, and J. Delsing, "Workflow management for edge driven manufacturing systems," in *2018 IEEE Industrial Cyber-Physical Systems (ICPS)*. IEEE, 2018, pp. 774–779.

[21] T. Rademakers, *Activiti in Action: Executable business processes in BPMN 2.0*. Manning Publications Co., 2012.

[22] Square, *JavaPoet*. [Online]. Available: https://github.com/square/javapoet

[23] K. Indrasiri, *Beginning WSO2 ESB*. Apress, 2016.