Real-time Visualization of Robot Operation Sequences

Real-time Visualization of Robot Operation Sequences ⋆

Ashfaq Farooqui * **Kristofer Bengtsson** * **Petter Falkman** *
Martin Fabian *

* *Department of Electrical Engineering, Chalmers University of Technology, Göteborg, Sweden 412 96 (e-mail: {ashfaqf, kristofer.bengtsson, petter.falkman, fabian}@chalmers.se)*

Abstract: Evaluation of manufacturing systems requires large amounts of accurate data from the factory floor. This data is then processed to calculate Key Performance Indicators (KPIs), evaluation metrics used within the manufacturing industry by engineers and managers in order to make data-driven decisions. Mechanisms to capture large scales of usable data, which is both reliable and scalable is, more often than not, scarce. In this paper, we provide an approach to capture data from robot actions, which can be applied to both legacy and current state-of-the-art manufacturing systems. By exploiting the robot code structure, robot actions are converted to event streams that are transformed into a higher usable abstraction of data. Applicability of this data is demonstrated, primarily, by visualizations. The described approach is developed in Sequence Planner – a tool for modeling and analyzing production systems – and is currently implemented at an automotive company as a pilot project to visualize and examine what goes on on the factory floor.

Keywords: Performance evaluation, Manufacturing systems, Data acquisition, Industrial robots, Data streams, Sequence estimation

1 Introduction

Access to data from the factory floor is crucial in order to evaluate the performance of any manufacturing station. This data is used to calculate Key Performance Indicators (KPIs) which, in turn, are used by engineers and managers to evaluate manufacturing systems. These KPIs include, for example: cycle times, machine downtime, resource utilization, productivity etc. To be able to use these KPIs to make reliable decisions, accuracy of the collected data is vital.

Often, to enable the process of collecting data and calculating KPIs, additional resources and software programs are added to the existing systems. However, additional tools used are typically rigid and, more often than not, hard-coded. This hard-coding has several challenging effects, especially as it relates to maintenance.

One major challenge within the automotive manufacturing industry has been to, conveniently, access and extract data from operating robots that could contribute to KPI calculations. One reason for this has been restrictions to access data imposed by robot vendors. Today, based on requirements, robots are programmed to send data to the PLC. This data usually includes alarms, warnings, and measurements made during execution. The PLC further aggregates the data and saves it in a central database. KPIs are then calculated using data stored in the database off-line at distinct time instances. To make this possible each robot needs additional code, in the robot and PLC, that has to be manually added and maintained. Therefore, it is currently not possible to visualize real-time performance for robots. These, real-time visualizations, help operators and engineers tune and improve performance of the system; and also enable data-driven approaches for preemptive maintenance.

The main contribution of this paper, is a method to gather useful data from robots to enable real-time visualization of robot operations within manufacturing systems. In order to do this, this paper provides a data processing pipeline that starts with a generic and non-intrusive approach to capture low level data from an existing robot station, followed by, transforming this data into a higher abstraction which can be used to further analyze the station. The end result, in this paper, is the real-time visualization of robot sequences by processing the abstracted data. The transformed data can also be used, in an on-line or off-line manner, to further calculate desired KPIs.

This paper also provides an insight into the tool *Sequence Planner* (Dahl et al., 2017) used to capture and process data from a demo station at an automotive manufacturing plant. However, for practical details of the application setup and configuration the interested reader is referred to Nord and Wahlqvist (2016).

The rest of the paper is organized as follows: The ideas and definitions used throughout the paper are discussed in Section 2. Section 3 takes a live robot station and provides implementation details for it. Section 4 provides insight

⋆ This project was supported by ITEA3 VINNOVA ENTOC (2016-02716) and VINNOVA LISA 2 (2014-06258).

into the use of live data; specifically, visualization of robot sequences. Finally, the paper concludes in Section 5.

2 Background

Within the manufacturing community different production paradigms exist, like: Reconfigurable, Flexible, Adaptive, Multi-agent, Holonic, manufacturing systems, to name a few (Ribeiro and Barata, 2011). At the core of these systems lies the idea that each subsystem performs one specific function. Similarly, in the computer science community the last few years have seen the development of Service Oriented Architectures (SOA), where each service performs one specific task. This parallelism in the two communities allows for easy integration of ideas between them.

Theorin et al. (2015) provide a broader discussion on various architectures available, and introduce the Line Information System Architecture (LISA) that uses an Event-driven Service Architecture (EDA). We use the ideas from LISA to build a streaming data pipeline; which is basically, a series of steps that transform real-time data between different formats and abstraction levels.

2.1 Pipeline components

A data pipeline broadly consists of two types of components, namely a *message bus*, responsible for communication and *endpoints*, data processing modules connected to the message bus that process the streaming data. In this method, data is passed through a pipeline in the following order for the desired result:

(1) *Virtual Device Endpoints*: VD endpoints provide an entry point to generate event streams from the physical hardware such as robots, PLCs, scanners etc., onto the message bus. Implementing communication details inside each low level systems with the message bus is a hard task, and not always feasible. Instead, the VD is a wrapper that provides a message-based interface; it thereby simplifies the architecture and allows plug-and-play design for hardware components, providing seamless integration for new devices.
The output from a VD endpoint is a stream of simple, low level messages that can be interpreted and used by all other endpoints.

(2) *Transformation Endpoints*: Low level systems communicate with simple messages which are sent by the VD onto the bus. Transformation endpoints convert such low level data, and generally data on a low abstraction level, into higher abstractions, thereby making the data more usable for other endpoints in the pipeline.

(3) *Service Endpoints*: Service Endpoints provide a service – one specific function – with the incoming data as input and may or may not send processed data on the bus. That is to say, services consume data from one or more transformations; then use this data to compute a result. Since services are based on user needs, the pipeline allows integration of new services without hampering the existing process. For example, services may include aggregation services, prediction services, calculation of KPIs, or visualization services.

2.2 Message bus

The message bus forms the communication layer allowing interaction between all endpoints. There exist, in literature and in practice, a number of possible configurations to create a message bus. While configuring a bus, the main objective is to aim for low complexity and high scalability. Ideally, it must be possible to change or upgrade the message bus without major changes to the code.

Services, in the pipeline, are triggered to execute when a message arrives for it. To make this possible, messages on the bus are structured into *topics*, and each service subscribes to one or more topics. When a message arrives on a specific topic subscribed by the service, that particular service is executed once.

To keep the implementation simple, the Apache ActiveMQ (AMQ, 2017) message bus was chosen. The bus was tested with a total of about 20 interacting resources and several services running online; no issues related to performance were noticed. However, if higher throughput and a distributed nature is required, Apache Kafka (kaf, 2017) or any other compatible bus can be used with minimal changes to the existing system.

Figure 1 shows an overview of the general computational pipeline presented in this section, along with their subfunctions. Section 3 elaborates the functions with an example applied to a real station.

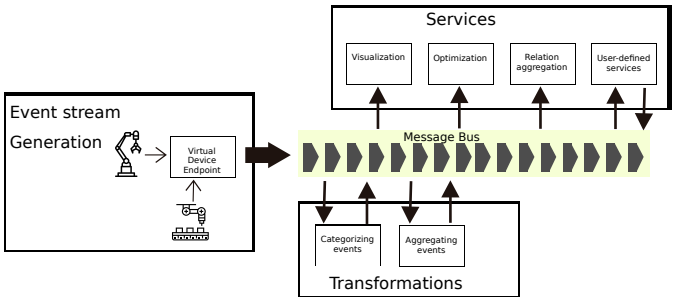

Fig. 1. Image of the complete pipeline

2.3 Example station

The approach presented in this paper has been implemented at a body-in-white production station at an automotive company. The station performs spot welding using four robots on a variety of different car models produced by the company. Once the car body enters the station and is in position, the robots choose the appropriate predefined program depending on the car model; and then, by moving in the workspace between pre-programmed spatial positions where they perform operations such as welding; during these movements they might need to wait for each other when entering common shared zones. Apart from this, tip dressing operations – the process of cleaning the welding tip – are also performed regularly by the robots to maintain weld quality.

ABB robots are used in this station which are programmed using a high-level programming language – *RAPID* – developed by ABB for their industrial robots. This language supports modular programming, that is, these programs

are divided into, smaller, self-contained, modules that contain a set of instructions corresponding to some physical division in the system. Each module is further divided into *routines* which correspond to a unit task performed by the robot. Each line in the program corresponds to an action by the robot, which is performed sequentially. That is, there are entry and exit points defined for the robot program. The line being executed is referred to by the *program pointer*. Apart from robot movements, the program also interfaces with the *signals* which provide input/output functionality. This code structure will prove beneficial when transforming the generated data into a different abstraction.

3 Event pipeline

Having defined the general framework let us now look at a real implementation on the example station. To simplify and understand the system, this paper will focus on connection and data from robots, specifically ABB industrial robots. The following section will elaborate on Figure 1.

The implementation is done using Sequence Planner (SP) (Dahl et al., 2017), a tool developed at Chalmers University for modeling and analyzing automation systems. SP is developed as a micro-service architecture, where services interact with each other by passing messages. This, micro-service architecture of SP, is exploited to implement the discussed pipeline.

3.1 ABB endpoint

A virtual device is required to interface with the ABB robots. Communication to the robots is done using the Robot SDK (ABB, 2017) provided by the manufacturer. This software provides a mechanism to subscribe to both the program pointer and the signals. The virtual device endpoint connected to the robot uses this SDK to subscribe and get notified for changes in either the program pointer or signals; which is followed by sending a message – after appending a header – on the message bus with the appropriate contents. An example robot message for a *program pointer position* is shown in Listing 1.

The message consists of a program pointer position, an address, and a header. The *programPointerPosition* block contains information regarding the position of the program pointer. It contains the names of the module and routine currently executed by the robot. Additional information pointing to the exact line number is available under `range`. The value of `time` contained here is the time stamp when this event was created. The `address` block differentiates the message as either a signal or program pointer.

The header block (the bottom three lines) contains information that helps identify the source of each message. It consists of a `workCellId`, a unique number to represent the manufacturing station where the message was generated; a time stamp for when the endpoint received the pointer change information is defined by `time`; and a `robotId` to identify the robot that generated the event.

If the event was generated when a signal value changes a signal message is created. The signal message, similar to a program pointer position message, consists of the header and a `Signal` block with the name of the signal and its corresponding updated value. Correspondingly, the `address` block refers to the signal address.

Listing 1. Program pointer message sent for every change in pointer position

```
{
  "programPointerPosition": {
    "position": {
      "module": "LD930R8119",
      "routine": "D931SchDefault",
      "range": {
        "begin": {
          "column": 5,
          "row": 250
        },
        "end": {
          "column": 28,
          "row": 250
        }
      }
    },
    "time": "2017-01-12T17:03:54.942+01:00",
    "task": "T_ROB1"
  },
  "address": {
    "kind": "programPointer",
    "path": [
      "T_ROB1"
    ],
    "domain": "rapid"
  },
  "robotId": "r8119",
  "time": "2017-01-12T17:03:54.942+01:00",
  "workCellId": "1736070"
}
```

3.2 Transformation endpoints

As mentioned earlier, transformation endpoints transform data from one abstraction to another. The incoming raw data from a robot as seen in Listing 1 needs to be refined into something more understandable. One sufficiently good abstraction to use is that of an *operation* (Bengtsson et al., 2009). The robot structure defined in Section 2.3 is made use of here. Routines in the program represent the tasks performed by the robot, these constitute "move from a to b", "grip", or "weld", and hence they are abstracted as operations for us to understand. Waiting for shared resources is accomplished using the keyword "WaitSignal" in the robot programs. Using this information, the following subsections provide a step by step approach to transforming the raw data into operations.

3.2.1 Naming and Categorizing events The first step in the transformation is to be able to differentiate between the incoming events. We do this by naming them according to the routine they are generated from. Furthermore, an event due to a wait must be differentiated from other tasks. For every robot message that arrives, text corresponding to the range specified by the program pointer is extracted from the robot program; this text is the instruction the robot currently performs. If the extracted instruction reads "WaitSignal" then the event is categorized as a "wait" event, else it belongs to a "routine". A new message is then sent out appending the original message with tags "instruction", containing the instruction extracted from the

program, and "isWaiting" with a value true if the event is categorized as a "wait" else a value false.

Listing 2. Three different operation events


```
{
  "activityId" : "80b1a21e-4535-4797-ad65-
      d0ec47e2fc99",
  "isStart" : true,
  "name" : "WaitSignal ReleaseStation2;",
  "robotId" : "r8121",
  "time" : "2017-01-13T12:37:01.907+01:00",
  "type" : "wait",
  "workCellId" : "1736070"
 },
 {
  "activityId" : "80b1a21e-4535-4797-ad65-
      d0ec47e2fc99",
  "isStart" : false,
  "name" : "WaitSignal ReleaseStation2;",
  "robotId" : "r8121",
  "time" : "2017-01-13T12:37:02.311+01:00",
  "type" : "wait",
  "workCellId" : "1736070"
 },
 {
  "activityId" : "048266f1-a071-4f1e-b77e-0
      e5de57a8c23",
  "isStart" : true,
  "name" : "B940ToPutFixt071_3",
  "robotId" : "r8119",
  "time" : "2017-01-13T12:36:34.951+01:00",
  "type" : "routines",
  "workCellId" : "1736070"
 }
```


3.2.2 Aggregate events to operations The named and categorized messages need to be further processed to make them usable. The next step in the pipeline is to aggregate the different messages into operations. This endpoint listens to named and categorized events generated as described in the previous section. It also keeps track of all available resources and the routines being currently executed by each of them. The output from this endpoint generates *operation events* – i.e. the start and stop events for operations. An operation event has a name, a time stamp, a resource where it is executed, and a flag that defines if this is a start or stop event. Listing 2 shows three different operation events, two events for a wait operation and a start event for a routine, running on two different robots. The start and stop operation events are also shown with their respective time stamps. Furthermore, operation events can be aggregated to view a complete operation as seen in listing 3. However, there is an advantage to preserve operation events instead of merging them into operations, since start-stop events can be processed to understand underlying relations between operations.

Listing 3. An aggregated operation


```
{
  "activityId" : "80b1a21e-4535-4797-ad65-
      d0ec47e2fc99",
  "name" : "WaitSignal ReleaseStation2;",
  "robotId" : "r8121",
  "startTime" : "2017-01-13T12:37:01.907+01:00",
  "stopTime" : "2017-01-13T12:37:02.311+01:00",
  "type" : "wait",
  "workCellId" : "1736070"
}
```


4 Services

Having real-time data from factory floors abstracted into operations, one can run various algorithms – in the form of services – to understand and visualize the tasks on the factory floor. The following section describes some service endpoints that uses the generated data.

4.1 Real-time resource based operation visualization

Information such as, execution time for each cycle, execution time for each operation, total waiting time, operations where robots wait for each other, are, among others, important to maintain and develop the station. An overview of ongoing operations can be visualized using real-time Gantt charts. Operators and engineers responsible for maintaining and developing the station use these charts to keep track of on-going processes at the station. Figure 2 shows a snapshot of a real-time Gantt chart of ongoing operations in all resources, appended with additional information such as execution time for each operation. Similarly, it is also possible to visualize historical cycles, as shown in Figure 3. In both figures, each robot is identified by its name and has two rows corresponding to it. The first row traces "routines" while the second traces "wait" operations.

4.2 Real-time operation-centered visualization

Relations between various operations are of interest; certain combinations might not be desired during execution. These relations between operations also serve as an input during analysis and optimization of the system. Instead of using a resource-centered view, one can visualize operations and their relations in a Gantt view as seen in Figure 4.

Take as an example robot `r8255` highlighted within a box in the same figure. We see that `r8255_ArcSoftServoTrap`, which is the operation to 'close' or 'open' the welding tip, runs after operations, `r8255_D910WeldDefault2` and `r8255_D910WeldDefault3`. Also, operations `r8255_D910WeldDefault2`, `r8255_D910WeldDefault3` and `r8255_D911WeldDefault1` run in a sequence, this relation can be deduced by a quick visual inspection of the Gantt chart. Though visual inspections, such as these, are possible, they are highly time-consuming and inaccurate. As number of interacting resources increase, so does the complexity in finding relations. Computational methods, in this scenario, will be helpful in performing the same analysis, but, with a higher degree of accuracy; this is performed as part of *sequence identification.*

4.3 Sequence identification

Visualization of robot operations are useful when interpreted by a human operator with sufficient understanding of the system. It would be of interest if a service could create, from available data, a model that captures the process behavior, and then to apply model-based algorithms that analyze the system and its performance. Building a general model – that defines the nominal operation of the station – can be done by Process Mining (van der Aalst, 2016) or Grammar Inference (de la Higuera, 2005; Bugalho

Fig. 2. A real-time Gantt view of operations running in the station.

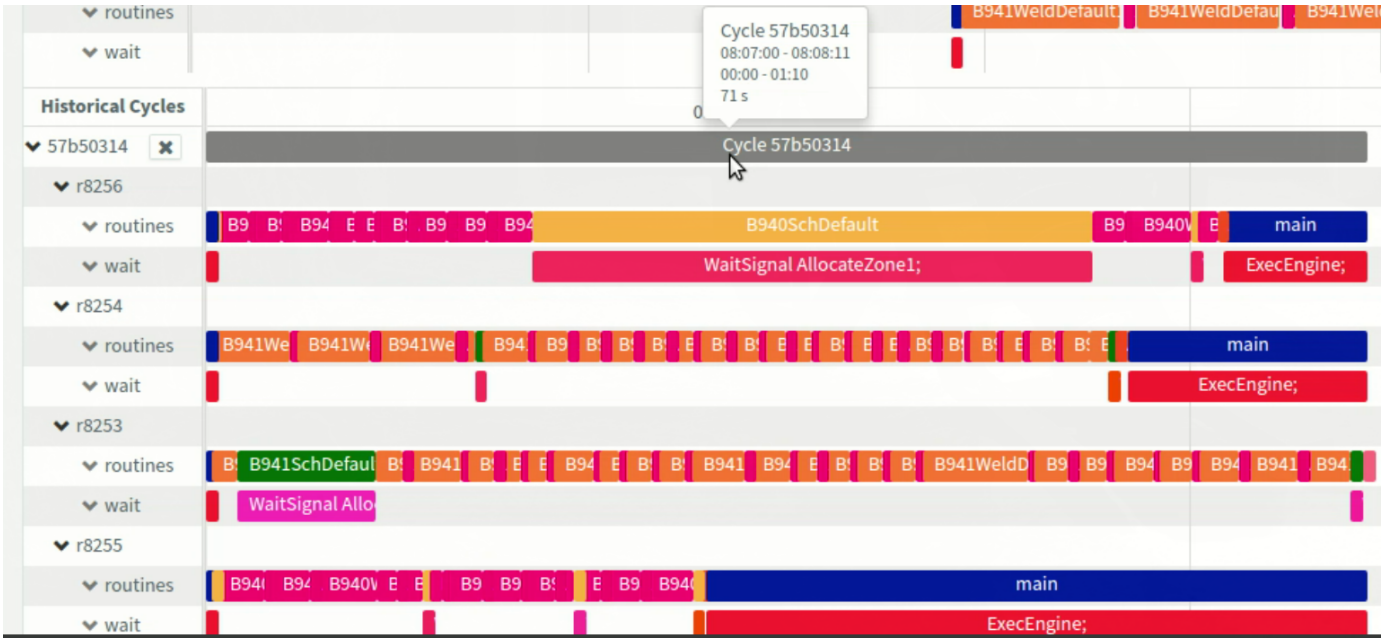

Fig. 3. Historical cycles with additional details

and Oliveira, 2005; Parekh and Honavar, 2000) from gathered data. The service employing such algorithms takes a stream of operations with start and stop times as inputs. By analyzing operation streams over several cycles, generic models describing the different operation sequences performed by each robot and of the complete station are generated; which can then be visualized using the language of Sequence of Operations (SOPs) introduced by Bengtsson et al. (2010). Farooqui et al. (2018) use the data from robots to infer the sequence of operations performed by the robots. These models are then used to further analyze, improve, and optimize the robot station. Furthermore, the models can be used to trace the behavior of the robots to detect errors.

4.4 Additional Services

The data generated and the messaging structure can be used to create different services which are briefly elucidated in this subsection.

Optimization services can use the models generated to optimize and find optimal sequences based on certain constraints. Dahl et al. (2017) discuss a method to optimize the sequence of operations with the aim of reducing cycle time. On the other hand, Riazi et al. (2017) and Sundström et al. (2017) focus on optimizing energy consumed by the robots by finding optimal operation sequences, such that the overall cycle time is constant while specific operations run at lower speeds; thereby saving energy. As these methods rely on the idea of operations they can be integrated in the defined architecture to use the messaging structure.

Prediction services can use past and present data to make certain predictions regarding the factory floor. Predicting potential errors and failures can help prepare operators early thereby mitigating downtime. One such example is to predict the replacement of the welding tip by analyzing tipdress operations which are responsible for cleaning the welding tip elucidated by Nord and Wahlqvist (2016). Similarly, access to several cycles of historic data opens up possibilities to predict a variety of KPIs such as cycle times, delays, maintenance requirement etc, that help in evaluation and maintenance of the station.

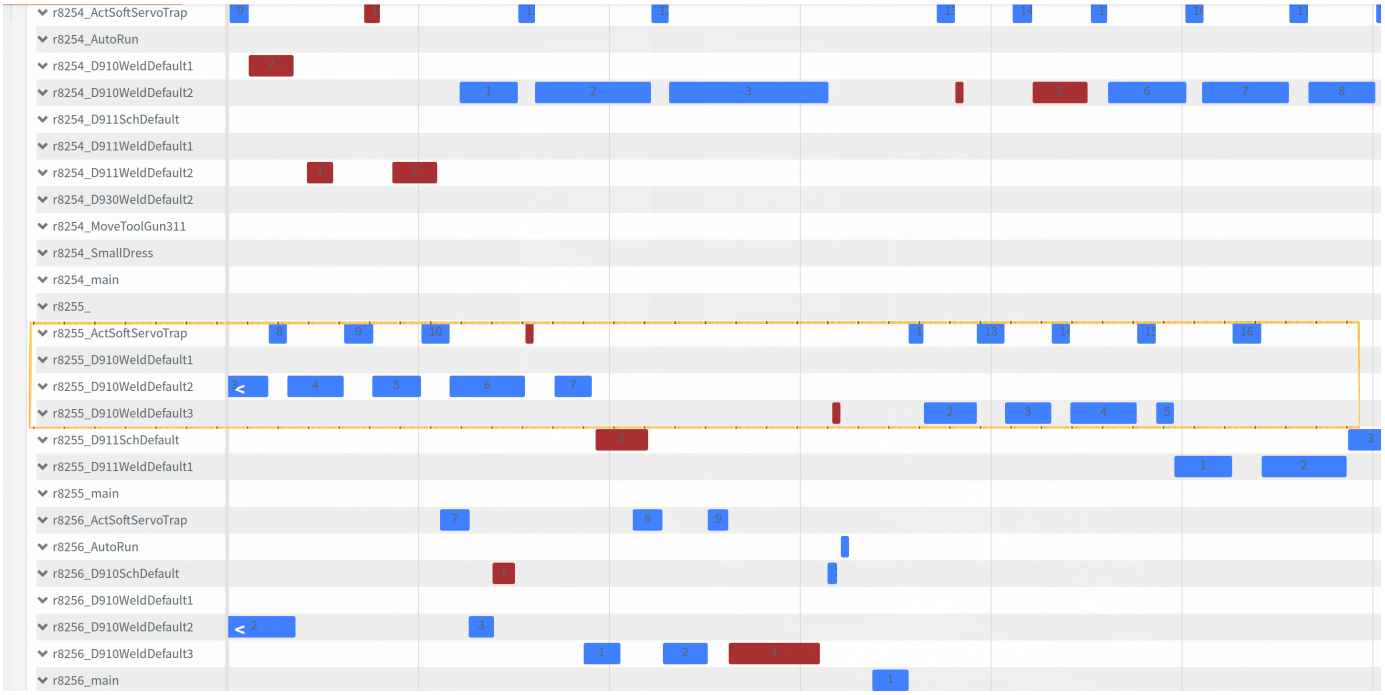

Fig. 4. Real-time view of ongoing operations and the number of times they have run

5 Conclusion

In conclusion, a robot agnostic approach that enables generation of data from, new and existing, robot operations is provided. This data, is streamed as events to a data processing pipeline transforming raw data to a usable abstraction of operations. These operations are then used to visualize the manufacturing system in different views that equip operators and engineers with a better understanding of the system. Additionally, access to more detailed data from robots has also paved way for advancement in the ways to calculate KPIs in a live production environment. Furthermore, access to detailed operation level data has made it possible to analyze the system by calculating a generic operating model for the station and then trace its behavior in real-time.

References

(2017). ABB Robot SDK. URL `http://developercenter.robotstudio.com`.

(2017). Apacha Kafka. URL `https://kafka.apache.org/`.

(2017). Apache Active MQ. URL `http://activemq.apache.org`.

Bengtsson, K., Lennartson, B., and Yuan, C. (2009). The origin of operations: Interactions between the product and the manufacturing automation control system. *IFAC Proceedings Volumes*, 42.

Bengtsson, K., Torstensson, C., Lennartson, B., Åkesson, K., Yuan, C., Miremadi, S., and Falkman, P. (2010). Relations identification and visualization for sequence planning and automation design.

Bugalho, M. and Oliveira, A.L. (2005). Inference of regular languages using state merging algorithms with search. *Pattern Recogn.*, 38(9).

Dahl, M., Bengtsson, K., Bergagård, P., Fabian, M., and Falkman, P. (2017). Sequence planner: Supporting integrated virtual preparation and commissioning. In *The 20th World Congress of the International Federation of Automatic Control, 9-14 July 2017*.

de la Higuera, C. (2005). A bibliographical study of grammatical inference. *Pattern Recognition*, 38(9).

Farooqui, A., Bengtsson, K., Falkman, P., and Fabian, M. (2018). From factory floor to process models: A data gathering approach to generate, transform, and visualize manufacturing processes. To be published.

Nord, D. and Wahlqvist, H. (2016). *The tweeting robot - Collection and processing of data from industrial robots.* Master's thesis.

Parekh, R. and Honavar, V. (2000). Grammar inference, automata induction, and language acquisition. *Handbook of natural language processing*, 727–764.

Riazi, S., Wigström, O., Bengtsson, K., and Lennartson, B. (2017). Energy and peak power optimization of time-bounded robot trajectories. *IEEE Transactions on Automation Science and Engineering*.

Ribeiro, L. and Barata, J. (2011). Survey paper: Rethinking diagnosis for future automation systems: An analysis of current diagnostic practices and their applicability in emerging IT based production paradigms. *Comput. Ind.*, 62(7).

Sundström, N., Wigström, O., Riazi, S., and Lennartson, B. (2017). Conflict between energy, stability, and robustness in production schedules. *IEEE Transactions on Automation Science and Engineering*, 14, 658–668.

Theorin, A., Bengtsson, K., Provost, J., Lieder, M., Johnsson, C., Lundholm, T., and Lennartson, B. (2015). An event-driven manufacturing information system architecture. *IFAC-PapersOnLine*, 48(3), 547–554.

van der Aalst, W. (2016). *Process Mining.* Springer Nature.