To appear in the *International Journal of Production Research*
DOI: 10.1080/00207543.2016.1201604

An Event-Driven Manufacturing Information System Architecture for Industry 4.0

Alfred Theorin[a]; Kristofer Bengtsson[c]; Julien Provost[c]; Michael Lieder[d]; Charlotta Johnsson[a*];Thomas Lundholm[d]; Bengt Lennartson[c]

[a]*Lund University, Lund, Sweden;* [c]*Chalmers University of Technology, Gothenburg, Sweden;* [d]*KTH Royal Institute of Technology, Stockholm, Sweden*

(*2015-11-12*)

Future manufacturing systems need to be more flexible, to embrace tougher and constantly changing market demands. They need to make better use of plant data, ideally utilizing all data from the entire plant. Low-level data should be refined to real-time information for decision making, to facilitate competitiveness through informed and timely decisions.

The *Line Information System Architecture*, LISA, is presented in this paper. It is an event-driven architecture featuring loose coupling, a prototype-oriented information model, and formalized transformation services. LISA is designed to enable flexible factory integration and data utilization. The focus of LISA is on integration of devices and services on all levels, simplifying hardware changes and integration of new smart services as well as supporting continuous improvements on information visualization and control. The architecture has been evaluated on both real industrial data and industrial demonstrators and it is also being installed at a large automotive company.

This article is an extended and revised version of the paper presented at the 2015 IFAC Symposium on Information Control in Manufacturing (INCOM 2015), (Theorin et al. 2015). The paper has been restructured in regards to the order and title of the chapters, and additional information about the integration between devices and services aspects have been added. The introduction and the general structure of the paper now better highlight the contributions of the paper and the uniqueness of the framework.

Keywords: automation, agile manufacturing, manufacturing information systems, service-oriented manufacturing systems, event-driven architecture

1. Introduction

Future industrial manufacturing systems need to make better use of the data (Hill and Smith 2009; Panetto and Molina 2008). Low-level data have to be transformed into useful information and smart services need to be integrated to support decision making. In addition, future manufacturing systems need to be productive, flexible, competitive, sustainable, secure, and safe. They have to be designed to reduce waste of material, capital, energy, and media. Improved control, reconfigurability, optimization, and human interaction in manufacturing processes is also important for future manufacturing (Blanc, Demongodin, and Castagna 2008; Bi, Lang, and Wang 2008).

The challenge to manage data, transform it into knowledge, and make smart automated decisions, has drawn a lot of attention during recent years. The main focus has been on the overall architecture, for example in collaborations like Industry 4.0 (Industrie 4.0 Working Group 2013), Smart Manufacturing Leadership Coalition (Smart Manufacturing Leadership Coalition 2016), Internet of Things (Atzori, Iera, and Morabito 2010), the Industrial Internet (Evans and Annunziata 2012), and cloud

*Corresponding author. Email: charlotta.johnsson@control.lth.se

robotics and automation (Kehoe et al. 2015).

Industries and device manufacturers are, however, reluctant to implement these new technologies due to unclear possible benefits, lack of clear implementation details, and the seemingly large investments required (Dawson 2014). An important driving force for adaptation may therefore be the possibility to integrate a large variety of devices and new services into existing systems. Most automotive companies already use advanced information systems (Dai et al. 2012). However, many of them lack important aspects to simplify integration.

Plants often use a wide range of devices, based on different technologies from different eras. Some devices originate from when the plant was built and devices have then been added as part of continuous improvements. Retrofitting legacy devices is thus a particularly important aspect. It must be possible to integrate them regardless of their capabilities or technology.

This paper presents the *Line Information System Architecture*, LISA, an innovative, yet simple architecture and design pattern for rapid integration of smart services into existing factory infrastructure. LISA is an event- and service-based information system architecture that is used to integrate devices and services, also called the Tweeting Factory (Lennartson et al. 2015). Simple messages (tweets) from all kinds of devices are sent out and transformed into high-level knowledge that is used by smart services for online monitoring, control, optimization, and reconfiguration (Theorin et al. 2015).

Parts of the LISA architecture have been incorporated at one of our partners, an industrial automotive industry, where thousands of devices like PLCs, robots, and scanners have been integrated during the last five years at multiple plants. A large variety of services, from SMS messages for Andon signals (process notifications) to the main MES system, have been connected. The architecture has also been evaluated using historical data from another automotive industry partner. LISA is able to handle layout and structural changes on the plant floor and allows a large diversity of devices and applications. Furthermore, LISA simplifies changes and updates when calculating Key Performance Indicators (KPIs), not only for new, but also for historical data. The importance of using KPIs for manufacturing companies is pointed out in (Cao et al. 2015). Innovative services, like energy optimization of robot motions, CNC-data aggregation, and Grafchart control have also been implemented to prove the simplicity to integrate new services.

In Section 2, the concepts of some common architectures are introduced. In Section 3, LISA is described, and in Section 4, selected sevices are explained. Finally, industrial use of LISA is described in Section 5, and conclusions are presented in Section 6.

2. Architectures

Information and communication architectures have been proposed in various areas in manufacturing research, for example, in planning (Umble, Haft, and Umble 2003), holonic manufacturing (Brussel et al. 1998), control (Dai et al. 2012; Babiceanu, Chen, and Sturges 2004; McFarlanea et al. 2003), service-oriented and cloud based architectures (Shena et al. 2007; Morariu, Borangiu, and Raileanu 2015), and agent systems (Leitao, Marik, and Vrba 2013).

These architectures require information about the real-time performance and behavior of the manufacturing plant. However, few are focusing on how to connect the large variety of devices and how to handle changes over time. Many companies have developed their own solutions, sometimes based on international standards such as ISA95 (ISA 2009). The solutions are typically based on Point-to-Point integration.

2.1 *Point-to-Point Integration*

When new functionality and systems are added, they need to be rapidly integrated with existing systems. The traditional integration approach in manufacturing is to connect applications on a Point-

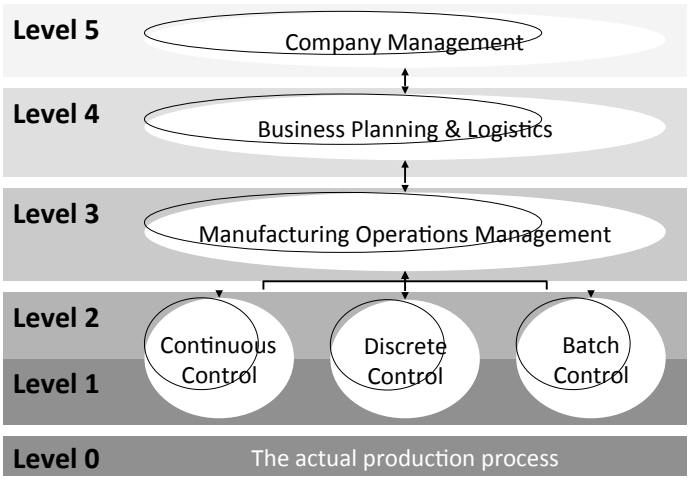


Figure 1. Functional hierarchy as defined by ISA95.

to-Point (PtP) basis using the client/server pattern. The pattern requires that the server and the client know about each other. The number of connections in a fully connected network increases quadratically with the number of applications. This is known as "spaghetti integration" and makes the system rigid and hard to maintain (Boyd et al. 2008). Each time an application is added, all other applications need to be updated to be able to interact with the new application.

It is common that applications can only communicate through proprietary or specific protocols, and applications may require external message translators to communicate with each other. This is, for example, the normal case for communication between Programmable Logic Controllers (PLCs) from different vendors. Another challenge is communication between the different levels of ISA95, see Fig. 1, known as vertical integration.

A common solution in industry is to use OPC to standardize how to access devices over a network. The main problem with only using OPC is that it tends to become a PtP solution, where for example the PLC variable structure must be known in many places. To handle this, many companies include a lot of logic in the devices to aggregate and transform data that is accessed over OPC.

The PtP approach poorly supports business requirements (Ribeiro, Barata, and Mendes 2008). Yet, industry has been slow to migrate to new approaches, mainly due to the cost of replacing their established legacy systems based on PtP (Boyd et al. 2008). However, migration has been significantly accelerated by the advent of Service-Oriented Architectures (SOAs) (He and Xu 2014).

2.2 *Service-Oriented Architecture*

SOA is a distributed software architecture where self-contained applications expose themselves as services, which other applications can connect to and use. To reach its full potential, SOA applications should be self-describing, discoverable, and platform- and language-independent. This leads to loose coupling and high flexibility.

SOA has recently received much attention in both academia and industry. The adoption of SOA in a company typically starts as an IT initiative to improve infrastructure efficiency and can then mature into optimized use for business purposes (Welke, Hirschheim, and Schwarz 2011). SOA is widely used on the business level and is expected to revolutionize manufacturing in a similar fashion (Li and Madnick 2015; Mueller et al. 2010).

The further down the hierarchy in Fig. 1, the shorter the task time frame. On level 1 it is common with hard real-time requirements, with deadlines in the order of milliseconds. The devices which execute on level 1 often have strictly limited memory and computational power. There is a trade-off between flexibility and real-time performance (Theiss, Vasyutynskyy, and Kabitzsch 2009) and

thus, the further down SOA is wanted, the more performant (and hence less flexible) it needs to be. Most SOA tools are tailored for business processes, which do not have strict timing or resource requirements. Thus, these tools cannot be used for manufacturing processes. However, there have been initiatives to bring SOA to level 1 and 2 by customizing the web service technology for resource constrained devices (Cucinotta et al. 2009; Dai et al. 2014).

2.3 *Event-Driven Architecture*

Even though SOA conceptually offers loose coupling and is intended to be distributed, service orchestration is typically done centrally, with the orchestrator taking control of the involved services. SOA 2.0, also known as advanced SOA or event-driven SOA, is the next generation of SOA that focuses on events, inspired by Event-Driven Architecture (EDA). SOA 2.0 enables service choreography, where each service reacts to published events on its own, rather than being requested to do so by a central orchestrator.

EDA is extremely loosely coupled and highly distributed by design. An event creator only needs to know that the event occurred, it does not need to know anything about who is interested in the event or how it will be processed (Michelson 2006). Event data should be immutable since it is then always (thread-)safe to send the events within and between applications. With EDA, applications turn from synchronized and blocking to asynchronous and non-blocking (Kuhn and Allen 2016).

3. Line Information System Architecture

LISA is an EDA that provides loose coupling of applications and devices, as well as a flexible message structure for integration. The core components of LISA are the message bus, the LISA message format, and communication and service endpoints. They enable creation and transformation of events into usable information in a loosely coupled way, and will be described in the following sections.

3.1 *LISA Events*

A common approach for information systems is an object-oriented structure for event types and events (Cheng et al. 1999). LISA on the other hand uses a prototype-based approach (Taivalsaari and Moore 2001). Prototypal inheritance, unlike object-oriented inheritance, is achieved by cloning and refining an object, here an event. This makes event creation, identification, and filtering less rigid, as there is no strict class hierarchy enforcing class relations.

When something happens, for example, when a machine changes state, an event with information about the change is sent. A LISA event is defined as $e = \langle id, t, AV \rangle$, where id is a unique event identifier, t is a timestamp, and $AV = \{attr_1 \colon value_1, \ldots, attr_k \colon value_k\}$ is a set of ordered attribute–value pairs describing the event.

Definition 1 (Attribute pattern)**.** *An attribute pattern $ap = \langle AV_{ap}, A_{ap} \rangle$ is a tuple including a set of ordered attribute–value pairs AV_{ap} and a set of attributes A_{ap}. If $e_1 = \langle id_1, t_1, AV_1 \rangle$ such that $AV_{ap} \subseteq AV_1$ and $A_{ap} \subseteq A_1$, where A_1 denotes all the attributes found in AV_1, then e_1 is matched by ap. This is denoted $e_1 \leftharpoonup ap$.* □

An attribute pattern is used to match, identify, filter, and create events. In this article, a pattern is denoted, for example, $ap = \{attr_1 \colon value_1, attr_2 \colon value_2, attr_3 \colon \_\}$, where $AV_{ap} = \{attr_1 \colon value_1, attr_2 \colon value_2\}$ and $A_{ap} = \{attr_3\}$. When the value is replaced with "_", that attribute can have any value. Values can also be a list of ordered attribute–value pairs or a list of values. Hence, hierarchical data structures can be represented.

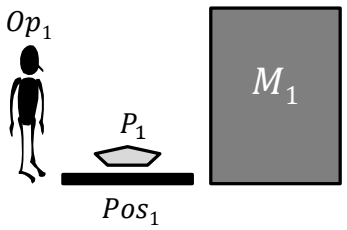


Figure 2. An example workstation.

Table 1. Attribute patterns used for creating and matching the example events.

$O_1^{\downarrow}$	name: $O_1^{\downarrow}$ location: [$Line_1$, WS_1] resources: [Op_1, Pos_1] rfid: _	$O_3^{\uparrow}$	name: $O_3^{\uparrow}$ location: [$Line_1$, WS_1] resources: [M_1, Pos_1] rfid: _
$O_2^{\uparrow}$	name: $O_2^{\uparrow}$ location: [$Line_1$, WS_1] resources: [M_1, Pos_1]	M_1^s	name: M_1^s location: [$Line_1$, WS_1] status: {mode: _, currentTool: _} consumption: {energy: _, duration: _}
$O_2^{\downarrow}$	name: $O_2^{\downarrow}$ location: [$Line_1$, WS_1] resources: [M_1, Pos_1] consumption: {energy: _, duration: _}		

Patterns can be defined freely by the user and are not enforced by LISA. However, the events receivers will match events based on patterns, which makes the definitions important. These patterns cannot be standardized for the lower levels of ISA95 since each plant has a unique system structure with a large diversity of devices.

Example

Consider the workstation WS_1 in Fig. 2. It consists of an operator Op_1, a product instance P_1 with product identifier p_1, a position Pos_1 and a machine M_1. The workstation can perform three operations: O_1 – place a product at Pos_1, O_2 – use M_1 to process the product at Pos_1, and O_3 – move the product at Pos_1 to the next workstation. Each operation is executed once per product instance and can be traced by start and stop events. Often, there are events which are not observable. Here, only $O_{1p_i}^{\downarrow}$, $O_{2p_i}^{\uparrow}$, $O_{2p_i}^{\downarrow}$, and $O_{3p_i}^{\uparrow}$ are observable, where $O_{kp_i}^{\uparrow}$ and $O_{kp_i}^{\downarrow}$ denote the start and stop events, respectively. These events are fired once per product instance P_i.

Events do not have to be related to the execution of an operation, for example, resource alarms, running mode changes, or the start of a lunch break. Here, the machine fires an M_{1i}^s event whenever the machine has changed execution mode (operating, idle, or down) and the events are based on the attribute patterns shown in Table 1. □

3.2 *Message Bus*

It is important with a standardized, structured, and generic concept to describe and implement loosely coupled software applications that are heterogeneous, disparate, and deployed and run independently. Hence, LISA uses an Enterprise Service Bus (ESB), a component that takes care of message routing between distributed applications. To avoid PtP connections and ensure loose coupling, the ESB should support the following Enterprise Integration Patterns (EIPs) (Hohpe and Woolf 2003):

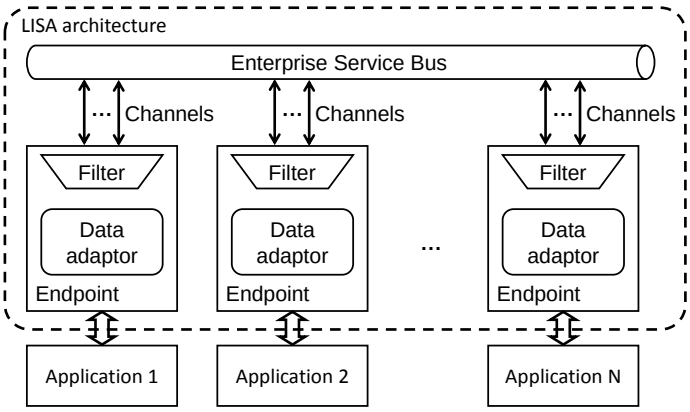


Figure 3. Overview of the LISA communication architecture.

- *Message*: The information or data are packaged into a message that can be transmitted on a message bus.
- *Messaging*: Messages are transferred immediately, frequently, reliably, and asynchronously using customizable formats. Messaging is event-based: when there is a new message, it is sent to the message bus.
- *Publish-subscribe channel*: When a message is sent on a publish-subscribe channel, a copy of the message is delivered to each channel subscriber.
- *Message filter*: If the content of an incoming message does not match the criteria specified by the message filter, the message is discarded. This pattern allows each application to further filter incoming messages.

In the LISA prototype Apache ActiveMQ is used, but it could be replaced by any ESB supporting these patterns.

Fig. 3 shows an overview of the communication architecture of LISA. The connection of applications (devices, services, external applications) to the ESB is through endpoints, which are responsible for 1) adapting the events and information according to the LISA message format, 2) publishing LISA messages on the corresponding channels on the ESB, and 3) filtering incoming LISA messages from the ESB. If an application is modified (for example, due to hardware replacement, variable renaming, or new measurements), only its endpoint needs to be changed. No other endpoints or applications need to be updated.

3.3 *LISA Message Format*

The LISA message format is designed to be simple and to enforce as little structure as possible. It consists of a header and a body. The header contains information related to message sending and routing. The body is an ordered key–value map between attributes (the keys) and their values. Values are usually of primitive data types, but can also be lists or maps. Hence, arbitrary hierarchical structures can be built and sent in LISA messages. Two attributes are mandatory in the body, namely an event *id* and an event *timestamp*, otherwise there are no constraints.

In the LISA prototype JSON is used as data format, but it could be replaced by any data format where maps and lists can be expressed.

Each plant has a unique system structure with different types of devices and LISA should be able integrate any device on level 1 and 2. LISA makes this possible by letting the users define the events. This might be considered a drawback, but it means that it is easier to change or extend events, which indeed makes LISA flexible.

LISA messages sent on the ESB are immutable. To refine a LISA message, a new LISA message

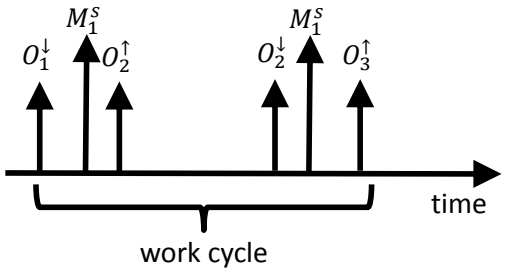


Figure 4. Events fired during one work cycle. Note that the first and second M_1^s are different events with the same name. They have different *id*, *t*, and data.

is created and sent to the ESB. The new message will have the same *id* and *timestamp*, but the content of the message may otherwise change arbitrarily.

3.4 *Communication Endpoints*

The connection between applications (communication devices, services, external applications) over the ESB is done through communication endpoints. Many devices have limited capabilities and knowledge and they communicate with different device specific protocols and interfaces. To replace all production equipment with new devices which all support the same specific protocol and interface is infeasible. Instead, the diversity of devices has been embraced and in LISA, devices are integrated with communication endpoints.

A communication endpoint is an adapter between the ESB and a device. Device event data are converted to the LISA message format and are published on ESB channels. Similarly, a communication endpoint filters events and converts and communicates event data to the device. With a communication endpoint, practically any device can be integrated into LISA. If an application is modified (for example, due to hardware replacement, variable renaming, or new sensors), only the corresponding communication endpoint needs to be updated. Devices could also send data directly on the ESB if they support that.

The messages from devices are usually low level, for example, a few bytes where each bit represents some variable. These bytes should be converted to structured data by transformation services. A key pattern in LISA is to perform data transformation logic outside the devices itself. If new data is needed, considerably more work is required to update all PLCs and robots in a plant than to deploy an updated service.

Example continued

There are three communication endpoints in the workstation: one connected to an RFID reader, one to M_1, and one to a PLC. The events fired during one work cycle are shown in Fig. 4. Event $O_1^\downarrow$ is fired by the RFID reader when it senses that a product is placed at Pos_1. The first M_1^s event is fired by M_1 when changing from idle to operating mode. Then, the PLC fires the start of the processing operation, $O_2^\uparrow$. When the processing is completed, the PLC fires $O_2^\downarrow$ and M_1 fires another event, M_1^s, telling that it is in idle mode. The work cycle is completed with $O_3^\uparrow$, which fires when the RFID reader senses that the product is removed. □

3.5 *Service Endpoints*

When calculating KPIs and controlling a plant with an MES system, most industries have similar structures. Hence, the low-level events should be transformed and updated to a more standardized structure with attribute names and semantics based on international standards like ISO 22400 (ISO

2014). This is managed by the service endpoints.

One challenge is to manage all the different devices. Many devices know little about the manufacturing. In the workstation example, only the RFID reader knows which product is at the workstation, or rather which product identifier. To calculate various KPIs, it is therefore necessary to transform, update, and aggregate events.

LISA classifies three basic types of transformations: Fill, Map, and Fold. Fill and Map add additional data to events and Fold transforms event sequences into new events.

Definition 2 (Fill)**.** *A Fill transformation transforms an event* $e = \langle id, t, AV \rangle$ *by appending a set of attribute–value pairs, that is,* $\langle id, t, AV' \rangle = Fill(e)$*, where* $AV \subset AV'$*.* □

Fill transformations only use static data. If applied to the same event, the result is always the same. A common use case is to add product identity and type based on an RFID tag, or to add information about the original event sender.

Often, an event needs information which depend on the current system state. If we study a system as a DES, a state can be identified based on an initial state and a sequence of events (Cassandras and Lafortune 2008). This is also true in the LISA architecture. Let Σ^* be the set of all finite sequences of events over the set of all LISA events Σ. Then, given a finite sequence $s \in \Sigma^*$ ordered by the timestamp, the state $q \in Q$ of the system is defined by $q = \delta(q^0, s)$, where q^0 is the initial state of the system and δ is the transition function of the system, defined as $\delta : Q \times \Sigma^* \to Q : (q^0, s) \mapsto \delta(q^0, s)$.

The state of a specific part of the system R, such as a product or a resource, can also be identified by an event sequence. If we define R using an attribute pattern ap^R, then the current state of R is $q_R = \delta(q_R^0, s_R)$, where only events that match ap^R are included in the sequence s_R. The Map transformation permits to refine an event according to the current system state.

Definition 3 (Map)**.** *A Map transformation transforms an event* $e = \langle id, t, AV \rangle$ *by appending a set of new attribute–value pairs based on the current state* q*, that is,* $\langle id, t, AV' \rangle = Map(e, q)$*, where* $AV \subset AV'$*.* □

Fill and Map can be used to transform events in multiple steps, to simplify the implementation and to increase the flexibility. However, they do not change the unique identifier *id* or the timestamp t of the event. The transformation history and the event version could be stored as attributes to make it easier to trace the transformation chain.

Definition 4 (Fold)**.** *A Fold transformation is a function that transforms a finite sequence of events,* $s \in \Sigma^*$*, into a single new event,* e*, that is,* $e = Fold(s)$*.* □

Fold can be used to bundle a set of events. It can also implement advanced event pattern identification languages like Complex Event Processing (CEP) (Luckham 2002) or real-time languages (Perez et al. 2014). CEP formalizes how patterns and knowledge are identified from a flow of low-level events, which results in high-level events (Cugola and Margara 2012).

Example continued

A Fill transformation updates RFID reader events with product identifier and product type attributes, that is, $O'^{\downarrow}_{1p_i} = ProductFill(O^{\downarrow}_{1p_i})$. A database that stores RFID tag numbers and their corresponding product identifiers and product types is used.

A Map transformation adds information about which product instance is at the workstation. This is known by listening to $O'^{\downarrow}_{1p_i}$ events.

One Fold transformation tracks when a product first enters the system and when it leaves, resulting in an event with the lead time of each product instance. Another Fold transformation tracks all operation events and combines start and stop events into an operation event which can, for example, include durations and consumptions. There is also a Fold transformation that aggregates the machine events, for each hour and for each day, to an event about operating behavior and energy consumption.

In summary, the following transformations are used:

- $e' = ProductFill(e)$. The product id and product type are added to events, where $e \leftharpoonup \{rfid, location\}$ and $e' \leftharpoonup \{rfid, location, productID, productType\}$.
- $e' = LastPositionFill(e)$. If a location is the last position for this product, it is added to the event. Here $e \leftharpoonup \{location, productID : p_{last}, productType\}$ and $e' \leftharpoonup \{lastPosition : true\}$. Observe that the transformation keeps all attributes, it is only the added key-value pair that is shown.
- $e' = ProductMap(e, q^L)$ is applied to events $e \leftharpoonup \{location, productID, productType\}$, that is, each location is mapped to the product located there (stored in the q^L states).
- $productMessage = ProductFold(\{e \in s | e \leftharpoonup \{productID\colon p_i\}\})$. Collects events related to a specific product identifier p_i and, after the last event, sends a product message. The message includes the time of the first and last events, the sequence of visited positions, and the aggregated operation energy consumption.
- $operationMessage = OperationFold(e_i \in \{O_i^{\uparrow}, O_i^{\downarrow}\})$. Collects operation events, O_i, and sends operation messages.
- $resourceMessage = ResourceFold(\{\forall e \in s | e \leftharpoonup \{resource : rid\}\})$. Collects events that match a specific resource rid and sends a status message every hour and every 24 hours. □

3.6 *LISA Flexibility*

Example continued

The line is extended with two more identical workstations, WS_2 and WS_3. O_3 now means moving the product in WS_1 to WS_2 and after the processing in M_2, the product is moved to WS_3 (O_5). The complete line, $Line_1$, includes four transport operations (O_1, O_3, O_5, O_7), three processing operations (O_2, O_4, O_6), and three machines that send events.

When the new workstations are connected to LISA, the messages will include the new layout without changing the service endpoints. For example, $productMessage$ will include events from the added workstations, including information about the longer lead time and the new processing steps. Also, $ResourceFold$ will automatically detect the new machines and start to send resource messages for them. Since these messages follow a structure understood by the upper level information receivers, these upper services do not have to change either. □

Absence of PtP communication as well as a multitude of event structures and event generators result in loose coupling between information levels. Using Fill, Map, and Fold transformations provides increased flexibility. The example may seem trivial, but this flexibility does typically not exist for automotive manufacturers. Often, a PtP communication approach is used and the upper level systems require detailed understanding about current layout, making the system layout rigid.

3.7 *Persistence*

When an application failure occurs, for example a random application crash, the application should be able to recover gracefully and should behave the same as if it had not failed. Ideally, other applications should not be able to tell if it had crashed or not. Of course, the timeliness will be affected, but the events generated by the crashed application should not be affected.

If all events are persisted, they can be replayed in the restarted application to make it reach the same state as when it crashed. The application can then proceed from there and produce the same events as if it had not crashed. Replay performance can be improved by occasionally persisting a state snapshot, which limits how much history has to be replayed. For practical reasons, there should be logic to avoid event duplicates during replay.

To store the whole history of events with the purpose to persist an application's state is called

event sourcing (Kuhn and Allen 2016). Compared to persisting the state itself, there are some notable differences. With event sourcing, the exact same application behavior can be replayed and analyzed in detail. For example, if an application is found to be in an incorrect state it is possible to step through the replay of events to find out which event processing introduced the error. It might even be possible to go back and correct some such errors retroactively.

Another advantage of event sourcing is that it is possible to apply the event history to new applications. For example, if an application that calculates a new KPI is added, it might be possible to calculate that KPI retroactively for the whole history. Since LISA is based on event sourcing, with all events stored in a journal (Kuc and Rogozinski 2013), it is possible to change and add services and execute them on historical data.

4. Integration of Services Into LISA

Due to its distributed nature, it is simple to integrate new services into LISA. New data can be identified in various devices and extracted via an endpoint. In many cases this can be done without changing the low level device code. In this section, KPI calculation services, energy optimization services, CNC machining data services, and control services are presented. These services have been evaluated either in a demonstrator at a university or in a real industrial situation.

4.1 *KPI Services*

LISA does not enforce calculation of specific KPIs or require that the user follows a specific standard. However, to allow the user of LISA to, in a flexible way, define and calculate KPIs on current and historical data, it is important to use well-defined attributes and values.

Example continued

Product lead time, T_C, is the time between the initiation of operating a product and its final delivery. Here, T_C is calculated for a product P_i with the product identifier p_i using the time difference between the first and the last event. This is the time between placing the product P_i at Pos_1 and removing it from WS_3.

The lead time is calculated in a *ProductFold* and is then added to the product message that the transformation sends out. The events have been transformed in a number of steps before the *ProductFold* creates the product message.

The event $O_7^{\downarrow}$ is part of the following transformations:

- $O_{7prod}^{\downarrow} \leftharpoonup \{productID : p_i\} = ProductFill(O_7^{\downarrow})$
- $O_{7last}^{\downarrow} \leftharpoonup \{lastPosition : true\} = ProductFill(O_{7prod}^{\downarrow})$
- $Product_i \leftharpoonup \{leadTime, ...\} = ProductFold(O_{7last})$

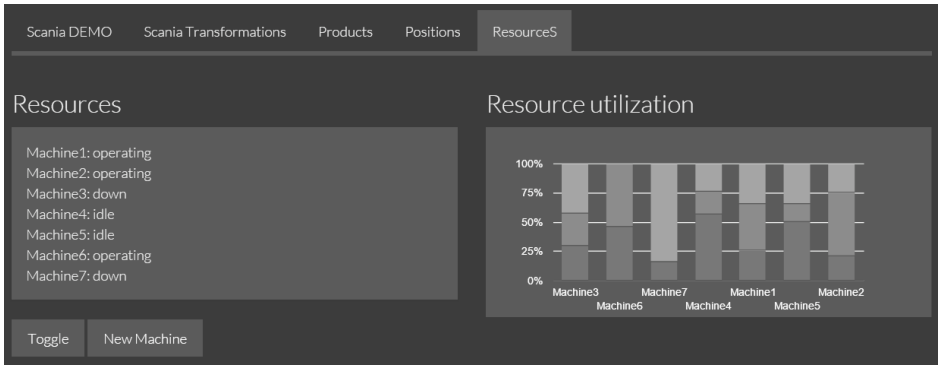
Downtime, T_D, is the time that a machine M_i is unavailable for operation and is defined as the sum of times between event pairs M_i^s that change mode to and from down. This is calculated in the *ResourceFold* transformation. With the same approach, idle time and operation time are calculated, and added to the resource messages.

The duration a particular product P_i stays at a certain position Pos_i is calculated as the time difference between P_i being put on Pos_i and removed from it. Aggregating time durations of all positions in the production line for a single product enables detailed visualization and analysis of time intensive operations, see Fig. 5(b).

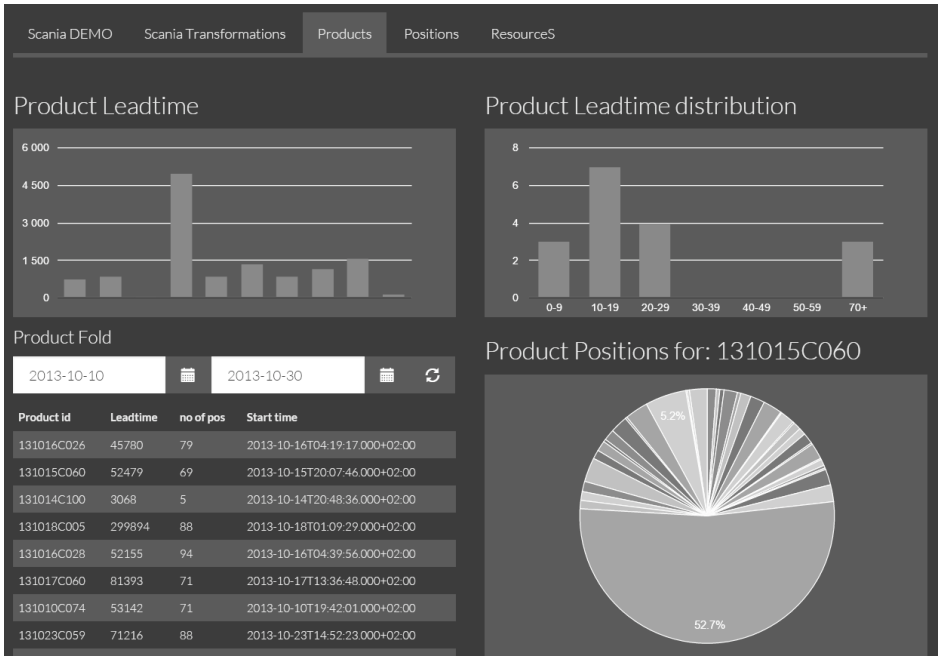
These KPIs are calculated by services and added to the messages. Some examples are shown in Table 2. KPIs for product lead time, availability, and product position times are quantified and

Table 2. KPI attributes for the example events.

O_2	M_1 status	productMessage	...
startTime	operationTime	productID	...
stopTime	downtime	Operations: [...]	
productID	idleTime	Consumption: [...]	
resources	Consumption: [...]	startTime	
consumption	Performance: [...]	stopTime	
...	...	...	



(a) Online KPIs for machine availability.



(b) Online KPIs for product lead time (top) and time spent at each position for a single product (bottom).

Figure 5. KPI visualization

visualized continuously for the LISA demonstrator, see Fig. 5(a) and Fig. 5(b). □

4.2 *Energy Optimization Services*

The number of robots connected directly to a network is constantly increasing. They can send detailed information about their status, motion trajectories, and operation timing. This enables new

and innovative analysis and support.

Currently, a demonstrator is being developed that shows automatic visualization of time and sequence diagrams of the robot work cycles, advanced monitoring of the welding process, error detection and troubleshooting support, and online energy optimization and prediction services (Riazi et al. 2015; Lennartson et al. 2015). The algorithm for energy optimization has been developed in the EU-project AREUS (Pellicciari et al. 2015). The algorithm can reduce the energy consumption of the robots by up to 30% (Vidarsson 2015). The input to the optimization is created by retrieving and transforming events from the robots sent via LISA. The optimization is based on the notion of hybrid operations. It is used as a common notion for activities in a factory, related to both products, manufacturing processes, and automation solutions. The optimization of a robot station using LISA is performed in the following steps:

- All devices send events during execution.
- The events are transformed and aggregated into hybrid operations.
- The hybrid operations are optimized.
- The new operations are sent to the robots and the PLC.
- As long as any of the robots' path is not manually changed, the station runs the optimized operations.

The framework for multi-robot scheduling and optimization focusing on energy consumption is implemented in the tool Sequence Planner (Bengtsson and Lennartson 2014; Lennartson et al. 2010). Sequence Planner is an operation and task-planning tool that enables innovative modeling and algorithms for sequence planning and coordination. Sequence Planner gathers all the hybrid operations from the message bus, identifies their sequential relations, and optimizes the robot trajectories. The result is sent back to the robots via LISA. When minimizing the energy of a robot station, Sequence Planner uses the nonlinear solver Ipopt (Wächter and Biegler 2006).

4.3 *CNC Machining Services*

Based on LISA as well as state-of-the-art technology, in for instance Industrie 4.0 (Industrie 4.0 Working Group 2013; Hermann, Pentek, and Otto 2015), and STEP-NC (ISO 2007; Lanab, Liua, and Zhanga 2008), the new Tweeting machine project develops Internet of Things functionality to provide *rich information from CNC machining*. The availability of such rich information will enable increased productivity and flexibility by improved or new functionality for design, process planning, tooling, operations, and quality control that can be realized and directly implemented in industrial applications.

4.4 *Control Services*

Control using LISA has been implemented on a demonstrator system consisting of a real PLC connected to a physical system, a CNC machine, JGrachart (described below), and an order system, each connected through a separate communication endpoint. The PLC system is connected via OPC, the CNC machine is connected via MTConnect, JGrafchart is connected via SocketIO, and the order system is just a mockup.

Grafchart is a graphical programming language which extends Sequential Function Charts (SFC), the IEC 61131-3 (IEC 2013) PLC standard language for sequential control (Johnsson 1999; Theorin 2014). SFC is supported by most industrial automation systems and is widely used in industrial automation. Grafchart has the same graphical syntax as SFC, with steps and transitions, and adds high-level features for hierarchical structuring, reusable procedures, and exception handling.

JGrafchart is a freely available Grafchart development environment, which supports service orchestration with web service technology (DPWS) (Theorin, Ollinger, and Johnsson 2013) and OPC

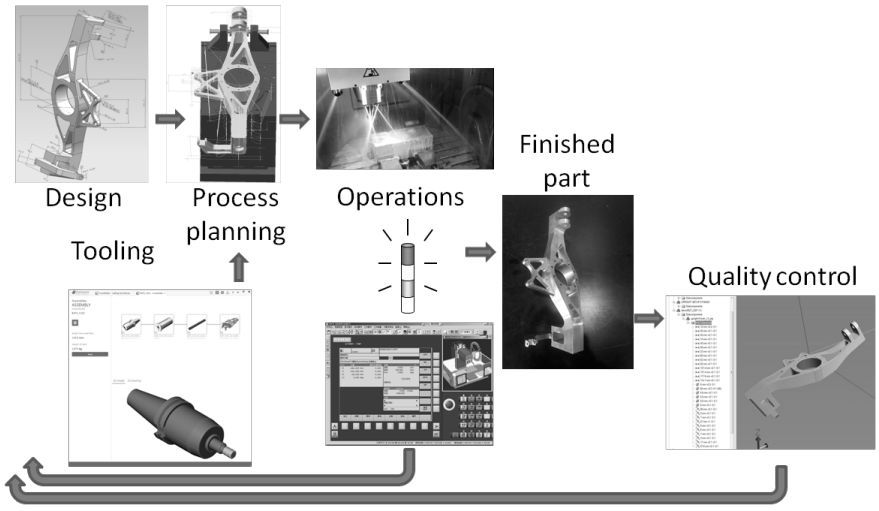


Figure 6. The new Tweeting machine

Unified Architecture (Theorin, Hagsund, and Johnsson 2014). Unlike these technologies, event-driven control does not provide any built-in error handling to detect, for example, invalid requests. Hence, to know if a request was successful, an acknowledgment event is required. In the PLC communication endpoint, all writable variables generate an event when they change, which ensures acknowledgments for write requests.

Both JGrafchart applications and the IEC 61131-3 applications in the PLC are executed periodically. If events are allowed to arrive at any rate to these applications, pulse events might be missed. To avoid this, the JGrafchart and PLC communication endpoints throttle the delivery rate of incoming messages according to the application execution rate.

An overview of production of an order in the demonstrator is shown in Fig. 7. A production request from the order system spawns a procedure call in JGrafchart. The request also triggers the CNC machine to start producing. When the CNCing completes, the product enters the physical system controlled by the PLC. The product is then completed through a collaboration between the PLC, which handles the real-time control, and JGrafchart, which handles high-level coordination and control. JGrafchart reacts to variable change events, such as new sensor values, from the PLC and sends control requests events to the PLC. When the production is completed, an event with the production log is sent.

A discussion about manufacturing views in relation to LISA is available in (Theorin et al. 2015).

5. Industrial Use

The LISA architecture and patterns have been developed by industrial and academic partners with the objective to be industrially applicable. It is inspired by international standards and established off-the-shelf solutions. One core aim of LISA is that it should be usable for any device and application.

One automotive industry partner has partly implemented their own version of LISA and is using it for both data acquisition and control in a new body-in-white plant. The LISA implmentation has resulted in dramatic time savings when upgrading the production system or when introducing new products. Most devices are now connected to the ESB and send out events when their state changes. The main differences in this industrial implementation compared to LISA are that XML is applied as a message format instead of JSON and that a commercial message bus is used instead of Apache ActiveMQ. The robots are directly connected to LISA, which makes it possible to monitor cycle times for robot stations and visualize the results in diagrams in real-time. Previously, a workstation

sent predefined KPIs for each work cycle. With LISA, all communication is event-based on a finer granularity, and devices like PLCs, robots, product carriers, and operators send and receive low-level events which are then aggregated to get the desired KPIs.

The KPI calculation service has also been evaluated on historical data from another automotive industry partner. A large database with production events has been played back into LISA to evaluate services as well as performance. The data did not conform to the LISA message structure, but due to the flexible nature of LISA, events could be identified and generated.

The energy optimization services are currently being evaluated at yet another automotive industry company where a real robot station is being optimized.

Since LISA is programming language independent, it is straightforward to integrate, for example, optimizers implemented in C with services implemented in Scala. This is a significant advantage for software development and maintenance.

6. Conclusions

LISA has been shown to be applicable for discrete manufacturing, for example in the automotive industry, where processes are running asynchronously and the product flow is non-linear. To validate interoperability, various industrial devices, software, and programming languages have been used. Several industrial partners have been involved and have provided valuable feedback on the applicability of the research and permitted evaluation of the architecture. As a result, LISA is an event-based service-oriented architecture which offers flexibility and scalability both for control of

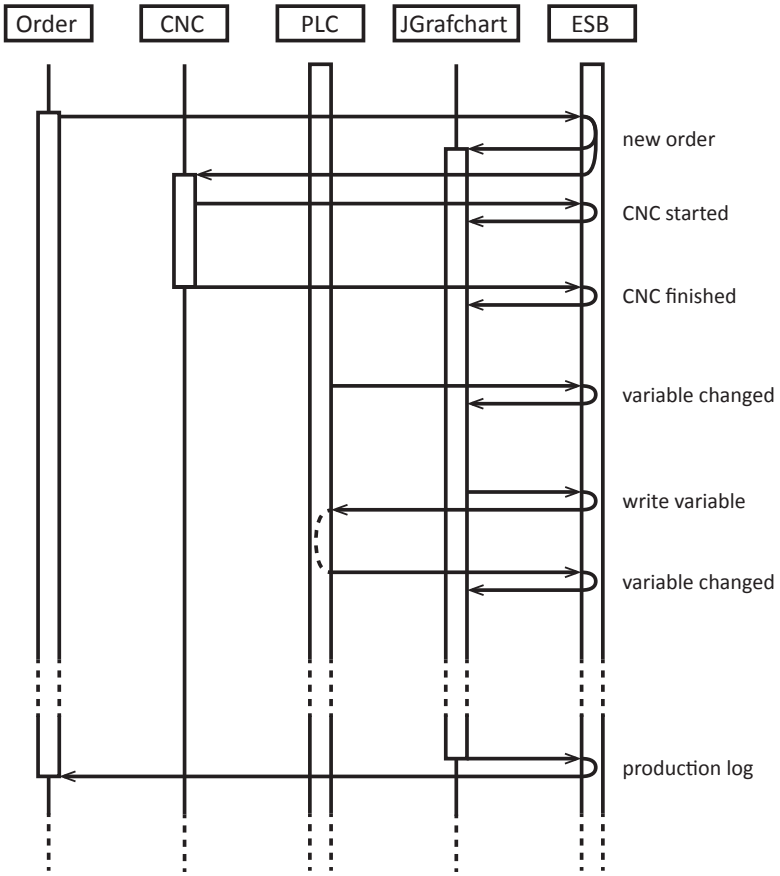


Figure 7. Production of an order in the demonstrator.

low-level applications and aggregation of higher level information, such as KPIs.

For the demonstrator, integration with LISA was straightforward. The advantages of the extreme loose coupling of EDA were also experienced. In particular, applications can be developed and tested in isolation, as other applications are easy to replace by mockups which simply produce events.

Improved visualization for decision support and integration of online optimization are future work.

Acknowledgements

The authors are grateful to the VINNOVA FFI programme as main funder of this project. A. Theorin and C. Johnsson are members of the LCCC Linnaeus Center and the ELLIIT Excellence enter at Lund University. M. Lieder and T. Lundholm are team members of XPRES-Initiative for excellence in production research at KTH Royal Institute of Technology. K. Bengtsson and B. Lennartson are members of the Wingquist Laboratory VINN Excellence Center at Chalmers University of Technology.

References

Atzori, Luigi, Antonio Iera, and Giacomo Morabito. 2010. "The internet of things: A survey." *Computer networks* 54 (15): 2787–2805.

Babiceanu, R. F., F. F. Chen, and R. H. Sturges. 2004. "Framework for the control of automated material-handling systems using the holonic manufacturing approach." *International Journal of Production Research* 42 (17): 3551–3564.

Bengtsson, Kristofer, and Bengt Lennartson. 2014. "Flexible specification of operation behavior using multiple projections." *Automation Science and Engineering, IEEE Transactions on* 11 (2): 504–515.

Bi, Z.M., W. Lang, S. Y. T.and Shen, and L. Wang. 2008. "Reconfigurable manufacturing systems: the state of the art." *International Journal of Production Research* 46 (4): 967–992.

Blanc, Pascal, Isabel Demongodin, and Pierre Castagna. 2008. "A holonic approach for manufacturing execution system design: An industrial application." *Engineering Applications of Artificial Intelligence* 21 (3): 315–330.

Boyd, Alan, D Noller, P Peters, D Salkeld, T Thomasma, C Gifford, S Pike, and A Smith. 2008. "SOA in Manufacturing Guidebook." *MESA International, IBM Corporation and Capgemini Co-Branded White Paper* .

Brussel, Hendrik Van, Jo Wyns, Paul Valckenaers, Luc Bongaerts, and Patrick Peeters. 1998. "Reference architecture for holonic manufacturing systems: {PROSA}." *Computers in Industry* 37 (3): 255 – 274.

Cao, Y., K. Zhao, J. Yang, and W. Xiong. 2015. "Constructing the integrated strategic performance indicator system for manufacturing companies." *International Journal of Production Research* 53 (13): 4102–4116.

Cassandras, Christos G., and Stéphane Lafortune. 2008. *Introduction to Discrete Event Systems, second edition.* Springer.

Cheng, Fan-Tien, Eric Shen, Jun-Yan Deng, and Kevin Nguyen. 1999. "Development of a system framework for the computer-integrated manufacturing execution system: A distributed object-oriented approach." *International Journal of Computer Integrated Manufacturing* 12 (5): 384–402.

Cucinotta, Tommaso, Antonio Mancina, Gaetano F Anastasi, Giuseppe Lipari, Leonardo Mangeruca, Roberto Checcozzo, and Fulvio Rusinà. 2009. "A Real-Time Service-Oriented Architecture for Industrial Automation." *Industrial Informatics, IEEE Transactions on* 5 (3): 267–277.

Cugola, Gianpaolo, and Alessandro Margara. 2012. "Processing Flows of Information: From Data Stream to Complex Event Processing." *ACM Comput. Surv.* 44 (3): 15:1–15:62.

Dai, Qingyun, Runyang Zhong, George Q. Huang, Ting Qu, T. Zhang, and T. Y. Luo. 2012. "Radio frequency identification-enabled real-time manufacturing execution system: a case study in an automotive part manufacturer." *International Journal of Computer Integrated Manufacturing* 25 (1): 51–65.

Dai, W.W., J.H. Christensen, V. Vyatkin, and V. Dubinin. 2014. "Function block implementation of service oriented architecture: Case study." In *Industrial Informatics (INDIN), 2014 12th IEEE International Conference on,* July, 112–117.

Dawson, Tim. 2014. "Industry 4.0 – Opportunities and challenges for smart manufacturing." http://blog.ihs.com/q13-industry-40-opportunities-and-challenges-for-smart-manufacturing.

Evans, Peter C, and Marco Annunziata. 2012. "Industrial internet: Pushing the boundaries of minds and machines." *General Electric* 21.

He, Wu, and Li Da Xu. 2014. "Integration of Distributed Enterprise Applications: A Survey." *Industrial Informatics, IEEE Transactions on* 10 (1): 35–42.

Hermann, M., T. Pentek, and B. Otto. 2015. "Design Principles for Industrie 4.0 Scenarios: A Literature Review." *Technische Universität Dortmund, Fakultät Maschinenbau, Audi Stiftungslehrstuhl Supply Net Order Management* .

Hill and Smith. 2009. Performers at ISA Expo 2009, Enterprise integration track, Reliant Center, Houston, TX, USA.

Hohpe, Gregor, and Bobby Woolf. 2003. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Englewood Cliffs, NJ: Addison-Wesley.

IEC. 2013. "IEC 61131: Programmable controllers - Part 3: Programming languages ed3.0." IEC Standard.

Industrie 4.0 Working Group. 2013. "Recommendations for implementing the strategic initiative INDUSTRIE 4.0." acatech.

ISA. 2009. "ISA95: Enterprise-Control System Integration, Part 1: Models and Terminology." ISA Standard.

ISO. 2007. "ISO 10303:Product data representation and exchange, Part 238: Application protocol, Application interpreted model for computerised numerical controllers." ISO Standard.

ISO. 2014. "ISO 22400: Key performance indicators (KPIs) for manufacturing operations management - Part 1: Overview, Concepts and Terminology." ISO Standard.

Johnsson, Charlotta. 1999. "A Graphical Language for Batch Control." Ph.D. thesis, Department of Automatic Control, Lund University, Sweden.

Kehoe, B., S. Patil, P. Abbeel, and K. Goldberg. 2015. "A Survey of Research on Cloud Robotics and Automation." *IEEE Transactions on Automation Science and Engineering* 12 (2): 398–409.

Kuc, Rafal, and Marek Rogozinski. 2013. *Elasticsearch Server*. Packt Publishing Ltd.

Kuhn, Roland, and Jamie Allen. 2016. *Reactive Design Patterns*. MEAP 10 ed. Manning Publications.

Lanab, H., R. Liua, and C. Zhanga. 2008. "A multi-agent-based intelligent STEP-NC controller for CNC machine tools." *International Journal of Production Research* 46 (14): 3887–3907.

Leitao, P., V. Marik, and P. Vrba. 2013. "Past, Present, and Future of Industrial Agent Applications." *Industrial Informatics, IEEE Transactions on* 9 (4): 2360–2372.

Lennartson, B., K. Bengtsson, O. Wigstrom, and S. Riazi. 2015. "Modeling and Optimization of Hybrid Systems for the Tweeting Factory." *Automation Science and Engineering, IEEE Transactions on* PP (99): 1–15.

Lennartson, Bengt, Kristofer Bengtsson, Chengyin Yuan, Kristin Andersson, Martin Fabian, Petter Falkman, and Knut Åkesson. 2010. "Sequence planning for integrated product, process and automation design." *Automation Science and Engineering, IEEE Transactions on* 7 (4): 791–802.

Li, X., and S.E. Madnick. 2015. "Understanding the Dynamics of Service-Oriented Architecture Implementation." *Journal of Management Information Systems* 32 (2): 104–133.

Luckham, David. 2002. *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison-Wesley Professional.

McFarlanea, Duncan, Sanjay Sarmab, Jin Lung Chirna, C.Y Wonga, and Kevin Ashtonb. 2003. "Auto {ID} systems and intelligent manufacturing control." *Engineering Applications of Artificial Intelligence* 16 (4): 365 – 376.

Michelson, Brenda M. 2006. "Event-driven architecture overview." *Patricia Seybold Group* 2.

Morariu, O, T Borangiu, and S Raileanu. 2015. "vMES: Virtualization aware manufacturing execution system." *Computers in Industry* 67: 27 – 37.

Mueller, B., G. Viering, C. Legner, and G. Riempp. 2010. "Understanding the Economic Potential of Service-Oriented Architecture." *Journal of Management Information Systems* 26 (4): 145–180.

Panetto, Hervé, and Arturo Molina. 2008. "Enterprise integration and interoperability in manufacturing systems: Trends and issues." *Computers in industry* 59 (7): 641–646.

Pellicciari, M., A. Avotins, K. Bengtsson, G. Berselli, N. Bey, B. Lennartson, and D. Meike. 2015. "AREUS – Innovative hardware and software for sustainable industrial robotics." In *Automation Science and Engineering (CASE), 2015 IEEE International Conference on,* Aug., 1325–1332. IEEE.

Perez, J., J. Jimenez, A. Rabanal, A. Astarloa, and J. Lazaro. 2014. "FTL-CFree: A Fuzzy Real-Time

Language for Runtime Verification." *Industrial Informatics, IEEE Transactions on* .

Riazi, Sarmad, Kristofer Bengtsson, Oskar Wigstrom, Emma Vidarsson, and Bengt Lennartson. 2015. "Energy optimization of multi-robot systems." In *Automation Science and Engineering (CASE), 2015 IEEE International Conference on,* 1345–1350. IEEE.

Ribeiro, Luís, José Barata, and Pedro Mendes. 2008. "MAS and SOA: complementary automation paradigms." In *Innovation in manufacturing networks,* 259–268. Springer.

Shena, Weiming, Qi Haoa, Shuying Wanga, Yinsheng Lia, and Hamada Ghenniwac. 2007. "An agent-based service-oriented integration architecture for collaborative intelligent manufacturing." *Robotics and Computer-Integrated Manufacturing* 23 (3): 315 – 325. International Manufacturing Leaders Forum 2005: Global Competitive Manufacturing.

Smart Manufacturing Leadership Coalition. 2016. "Smart Manufacturing Leadership Coalition." *visited on 2016-04-05*, https://smartmanufacturingcoalition.org.

Taivalsaari, A., and I. Moore. 2001. *Prototype-Based Object-Oriented Programming: Concepts, Languages, and Applications.* Springer-Verlag New York, Inc.

Theiss, Sebastian, Volodymyr Vasyutynskyy, and Klaus Kabitzsch. 2009. "Software agents in industry: A customized framework in theory and praxis." *Industrial Informatics, IEEE Transactions on* 5 (2): 147–156.

Theorin, Alfred. 2014. "A Sequential Control Language for Industrial Automation." Ph.D. thesis, Department of Automatic Control, Lund University, Sweden.

Theorin, Alfred, Kristofer Bengtsson, Julien Provost, Michael Lieder, Charlotta Johnsson, Thomas Lundholm, and Bengt Lennartson. 2015. "An Event-Driven Manufacturing Information System Architecture." *IFAC-PapersOnLine* 48 (3): 547 – 554. 15th {IFAC} Symposium on Information Control Problems in Manufacturing (INCOM'2015), http://www.sciencedirect.com/science/article/pii/S2405896315003778.

Theorin, Alfred, Johan Hagsund, and Charlotta Johnsson. 2014. "Service Orchestration with OPC UA in a Graphical Control Language." In *19th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA'2014),* Barcelona, Spain, Sep.

Theorin, Alfred, Lisa Ollinger, and Charlotta Johnsson. 2013. "Service-oriented Process Control with Grafchart and the Devices Profile for Web Services." In *Service Orientation in Holonic and Multi Agent Manufacturing and Robotics,* Vol. 472 of *Studies in Computational Intelligence*213–228. Springer Berlin Heidelberg.

Umble, Elisabeth J, Ronald R Haft, and M Michael Umble. 2003. "Enterprise resource planning: Implementation procedures and critical success factors." *European Journal of Operational Research* 146 (2): 241–257.

Vidarsson, Emma. 2015. *Experimental evaluation of energy optimized trajectories for industrial robots.* Master's Thesis EX025/2015. Signals and Systems, Chalmers University of Technology.

Wächter, Andreas, and Lorenz T Biegler. 2006. "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming." *Mathematical programming* 106 (1): 25–57.

Welke, Richard, Rudy Hirschheim, and Andrew Schwarz. 2011. "Service-Oriented Architecture Maturity." *Computer* 44.