An Event-Driven Manufacturing Information System Architecture

An Event-Driven Manufacturing Information System Architecture

Alfred Theorin [*] **Kristofer Bengtsson** [**] **Julien Provost** [***]
Michael Lieder [****] **Charlotta Johnsson** [*]
Thomas Lundholm [****] **Bengt Lennartson** [†]

[*] *Lund University, Lund, Sweden (alfred.theorin@control.lth.se, charlotta.johnsson@control.lth.se).*
[**] *Sekvensa AB, Gothenburg, Sweden (kristofer@sekvensa.se).*
[***] *Technische Universität München, Munich, Germany (provost@ses.mw.tum.de).*
[****] *KTH Royal Institute of Technology, Stockholm, Sweden (lieder@kth.se, thomas.lundholm@iip.kth.se).*
[†] *Chalmers University of Technology, Gothenburg, Sweden (bengt.lennartson@chalmers.se).*

Abstract: Future manufacturing systems need to be more flexible, to embrace tougher and constantly changing market demands. They also 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 designed to enable flexible factory integration and data utilization. In LISA, international standards and established off-the-shelf technologies have been combined with the main objective to be industrially applicable. LISA is an event-driven architecture with a prototype-oriented information model and formalized transformation services. It features loose coupling, flexibility, and ease of retrofitting legacy devices. The architecture has been evaluated on both real industrial data and industrial demonstrators and is also being installed at a large automotive company.

Keywords: industry automation, agile manufacturing, flexible manufacturing systems, architectures, events, decision support systems, automobile industry

1. INTRODUCTION

A hidden asset in manufacturing industry is data. Investigations estimate that 85% of the data are unstructured, and 42% of all transactions (sending and receiving information) are paper-based (IBM (2007)). CEOs in the manufacturing industry say that "we need to do a better job to capture and understand information rapidly in order to make sound business decisions" (Hill and Smith (2009)).

Future industrial manufacturing systems need to make better use of the data (Panetto and Molina (2008)). The low-level data has to be transformed into information that can be used for decision making. In addition, future manufacturing systems need to be productive, flexible, competitive, sustainable, secure, and safe and must reduce waste of material, capital, energy, and media. Most automotive companies use advanced information systems (Dai et al. (2012)). However, most of these systems lack many key features. Improved control, optimization, and human interaction in manufacturing processes is also important for future manufacturing (Blanc et al. (2008)).

There are several types of manufacturing information systems, such as Manufacturing Execution Systems (MES) (Dai et al. (2012)), Enterprise Resource Planning (ERP) (Umble et al. (2003)), or Multi-Agent Systems (MAS) (Leitao et al. (2013)). These systems require information about the real-time performance and behavior of the manufacturing plant. However, there is no vendor-independent integration architecture for such information management and many companies have developed their own solutions. With an increasing demand to launch new vehicle models faster, automotive companies require flexible and scalable information systems.

To enable access to the data, all devices and software must first be integrated. To accomplish this, a flexible architecture is needed which facilitates integration of any application or device. 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.

The contribution of this paper is a new information system architecture, called Line Information System Architecture (LISA), that enables flexibility and scalability. The architecture is event-based, has formalized transformation patterns, and uses stream-based aggregation and prototype-oriented information models. LISA is able to handle layout and structural changes on the plant floor and allows a

large diversity of devices and applications. Furthermore, LISA enables new Key Performance Indicators (KPIs) to be calculated, not only for new, but also for historical data. LISA has been implemented and evaluated on industrial data and demonstrators, and is being installed at a large automotive company.

In Section 2, the concepts of service-oriented and event-driven architectures are introduced. In Section 3, LISA is described. How LISA can be used for KPI calculation is explained in Section 4. Finally, event-based control with LISA is presented in Section 5.

2. SERVICE-ORIENTED ARCHITECTURES

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-to-Point (PtP) basis using the client/server pattern. The pattern mandates 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.

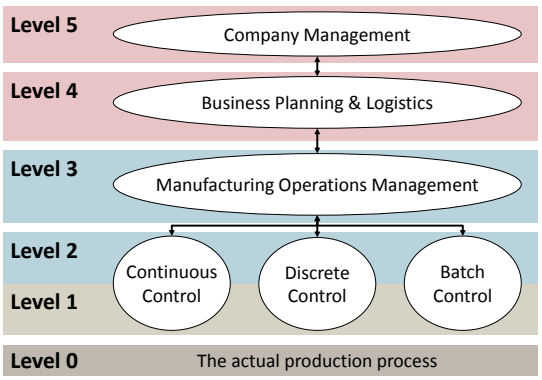

Fig. 1. Functional hierarchy as defined by ISA95.

The PtP approach poorly supports business requirements (Ribeiro et al. (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)).

2.1 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 et al. (2011)). SOA is widely used on the business level and is expected to revolutionize manufacturing in a similar fashion.

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 et al. (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)).

2.2 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 (2014)).

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 $e_1 \leftharpoondown ap$. □

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.

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 $O^{\downarrow}_{1p_i}$, $O^{\uparrow}_{2p_i}$, $O^{\downarrow}_{2p_i}$, and $O^{\uparrow}_{3p_i}$ are observable, where $O^{\uparrow}_{kp_i}$ and $O^{\downarrow}_{kp_i}$ 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^s_{1i} 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.

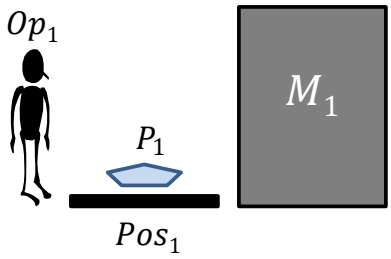

Fig. 2. An example workstation.

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

$O^{\downarrow}_1$	name: $O^{\downarrow}_1$ location: [$Line_1$, WS_1] resources: [Op_1, Pos_1] rfid: _
$O^{\uparrow}_2$	name: $O^{\uparrow}_2$ location: [$Line_1$, WS_1] resources: [M_1, Pos_1]
$O^{\downarrow}_2$	name: $O^{\downarrow}_2$ location: [$Line_1$, WS_1] resources: [M_1, Pos_1] consumption: {energy: _, duration: _}
$O^{\uparrow}_3$	name: $O^{\uparrow}_3$ location: [$Line_1$, WS_1] resources: [M_1, Pos_1] rfid: _
M^s_1	name: M^s_1 location: [$Line_1$, WS_1] status: {mode: _, currentTool: _} consumption: {energy: _, duration: _}

To avoid PtP connections and ensure loose coupling, the ESB should support the following Enterprise Integration Patterns (EIPs) (Hohpe and Woolf (2003)):

- *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 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

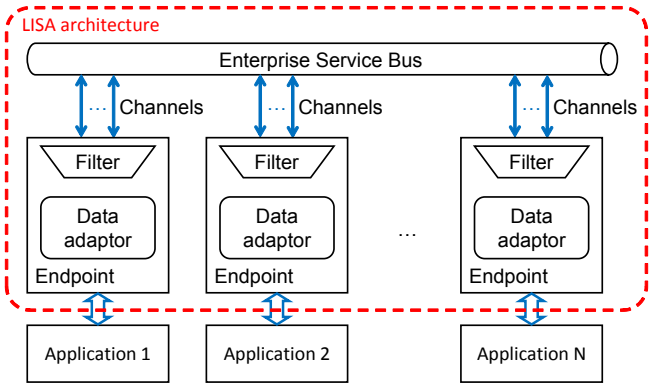

Fig. 3. Overview of the LISA communication architecture.

related to message sending and routing. The body is an ordered attribute–value map. Values are usually of primitive data types, but can also be lists or maps. Hence, LISA messages support hierarchical structures. The only mandatory attributes are in the body, namely event *id* and timestamp, *t*. Otherwise, there are no constraints.

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, which could be considered a drawback. However, it means that it is easier to change or extend events, which indeed makes LISA flexible.

3.4 Communication Endpoints

Many devices have limited capabilities and knowledge and they communicate with different device specific protocols and interfaces, for example, OPC or RS-232. To replace all production equipment with new devices which all support the same specific protocol and interface is unfeasible. Instead, the diversity of devices has been embraced. In LISA, devices are integrated with communication endpoints.

A communication endpoint is an adapter between a device and the ESB. Device event data are converted to the LISA message format and are published on ESB channels. Similarly, the communication endpoint filters events and converts and communicates event data to the device. If a device is modified or replaced, only the corresponding communication endpoint needs to be updated.

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.

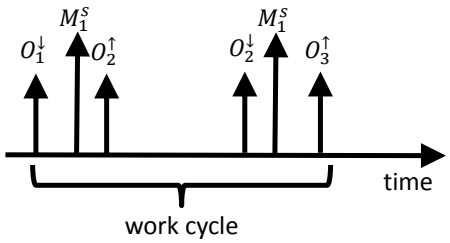

Fig. 4. Events fired during one work cycle. Note that the first and second M_1^s are different events with the same name. They have different *id*, *t*, and data.

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 (2014a)). 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 identi-

fier *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 algorithms 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).
- $productMessage = ProductFold(\{e \in s | e \leftharpoonup \{productID: p_i\}\})$. 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 also 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 the current layout, making the system layout rigid.

3.7 Standards

Standards offer harmonized terminology with the objective to improve communication. Hence, different terms for the same thing or the same term for different things are avoided. Standards also offer concepts to facilitate design and operation of industrial manufacturing systems. The following standards have been useful when designing LISA:

- ISA95/IEC 62264, Enterprise-Control System Integration (ISA (2009)) (IEC (2003)). This standard focuses on defining the domain of Manufacturing Operation Management (MOM) and its interface to business and logistics systems, also known as ERP.
- ISO 22400, KPIs for manufacturing operations management (ISO (2014a)) (ISO (2014b)). This standard defines common KPIs for MOM. In particular, the domains of manufacturing, product quality assurance testing, materials handling, inventory, and maintenance are considered.
- ISO 20140, Evaluating energy efficiency and other factors of manufacturing systems that influence the environment (ISO (2013)). This standard establishes a method for evaluating environmental influence of a manufacturing system, for example, energy consumption, waste, and release.

4. KPI CALCULATION AND VISUALIZATION

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.

This final lead time is calculated in the *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:

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

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. 6.

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 visualized continuously for the LISA demonstrator, see Fig. 5 and Fig. 6. □

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	
...	...	...	

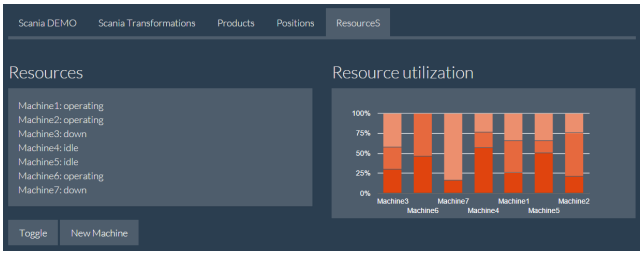

Fig. 5. Online KPIs for machine availability.

5. CONTROL

Events originating from different levels can be used to perform high-level coordination and control. The production could, for example, be initiated by a business level order event and carried out through interaction between production machines, coordination software, and field devices.

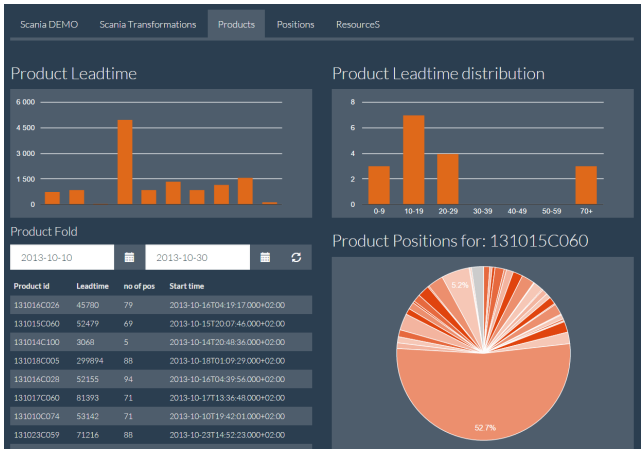

Fig. 6. Online KPIs for product lead time (top) and time spent at each position for a single product (bottom).

A machine-centered plant view focuses on the flow through each machine while an order-centered view focuses on flow of the product being produced for each order. A machine-centered plant view makes it easier to get the control right while an order-centered view is more relevant on the higher levels as produced products are parts of orders which should be possible to trace and visualize. Depending on which you choose you get a good overview of either the machines or the orders. Even so, with the right events both views can be constructed.

5.1 Grafchart

Grafchart is a graphical programming language which extends Sequential Function Charts (SFC), the IEC 61131-3 (IEC (2013)) PLC standard languages for sequential applications. SFC is supported by most industrial automation systems and is widely used and accepted for 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. Associated with the steps are actions which specify what to do. Associated with each transition is a condition for when the application state may change, for example, when an event with specific content is received.

JGrafchart is a freely available development environment for Grafchart which has previously been used for service orchestration using web service technology (DPWS) (Theorin et al. (2013)) and OPC Unified Architecture (Theorin et al. (2014)). Unlike these technologies, event-driven control does not provide any built-in error handling to detect, for example, invalid requests. To know if a request was successful, an acknowledgment event is required.

LISA is event-driven while JGrafchart applications are executed periodically. If events are allowed to arrive at any rate to JGrafchart, pulse events might be invisible to the JGrafchart application. To avoid this kind of issues, the communication endpoint throttles the message delivery rate to JGrafchart according to the execution rate.

With a machine-centered view, each machine selects which product to process next and gets information about how to process the product. With an order-centered view, each

product selects where it should be processed and tells the machine how to process the product.

Example continued Machine-centered and order-centered control for the workstations could look like in Fig. 7 and Fig. 8 respectively. Here, reusable procedures are particularly useful for workstation and order control. □

5.2 Demonstrator

Control with LISA has been evaluated on a system consisting of a real PLC connected to a physical system, a CNC machine, and an order system, each connected through a communication endpoint. The CNC machine is connected via MTConnect, the PLC system is connected via OPC, JGrafchart is connected via SocketIO, and the order system is a mockup. In the OPC endpoint, all writable variables generate an event when they change, which ensures acknowledgments for write requests. As there is no central coordinator, this is classified as a service choreography.

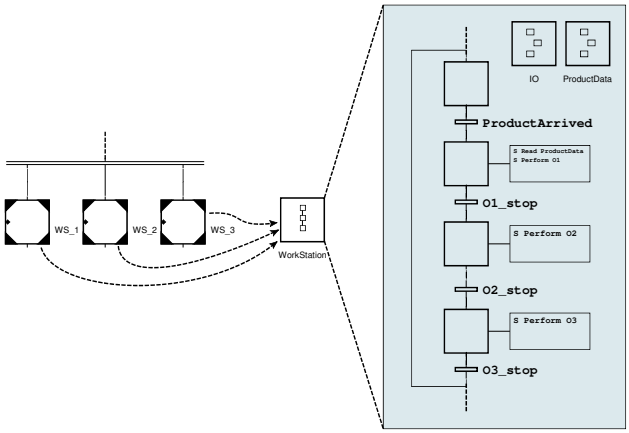

Fig. 7. Machine-centered control. The WorkStation implementation is reusable, parameterized by its I/O and product processing information.

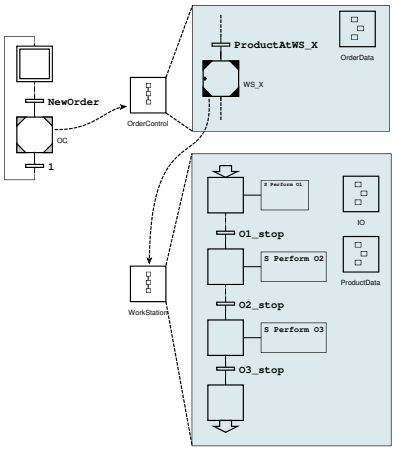

Fig. 8. Order-centered control. The implementation is reusable, parameterized by the order data.

For the demonstrator, an order-centered view was selected. 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 and JGrafchart begins to send requests to the PLC which handles the real-time control. When the production is completed, an event with the production log is sent. The resulting demonstrator control program in JGrafchart is similar to Fig. 8.

6. CONCLUSION

LISA has been developed with the objective to be industrially applicable. It is to a large extent based on international standards and established off-the-shelf solutions, for example, ActiveMQ. It 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. One core aim of LISA is that it should be usable for any device and application. To confirm interoperability, various industrial devices and software have been used in the demonstrators. Involvement of several industrial partners 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 low-level applications and aggregation of higher level information, such as KPIs.

An automotive company that has been involved in this research is currently installing LISA. It will be used in their new body-in-white plant. Previously, at the company, 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.

LISA has also been evaluated on historical data from another automotive manufacturer. The data did not conform to the LISA message structure, but due to the flexible nature of LISA, events could be identified and generated.

For the demonstrator, integration with LISA was straightforward. The advantages of the extreme loose coupling of EDA were also experienced. In particular, applications could be developed and tested in isolation as the other applications were easily replaced by mockups, which produce events without their respective physical device.

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

Blanc, P., Demongodin, I., and Castagna, P. (2008). A holonic approach for manufacturing execution system

design: An industrial application. *Engineering Applications of Artificial Intelligence*, 21(3), 315–330.

Boyd, A., Noller, D., Peters, P., Salkeld, D., Thomasma, T., Gifford, C., Pike, S., and Smith, A. (2008). Soa in manufacturing guidebook. *MESA International, IBM Corporation and Capgemini Co-Branded White Paper*.

Cassandras, C.G. and Lafortune, S. (2008). *Introduction to Discrete Event Systems, second edition*. Springer.

Cheng, F.T., Shen, E., Deng, J.Y., and Nguyen, K. (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, T., Mancina, A., Anastasi, G.F., Lipari, G., Mangeruca, L., Checcozzo, R., and Rusinà, F. (2009). A real-time service-oriented architecture for industrial automation. *Industrial Informatics, IEEE Transactions on*, 5(3), 267–277. doi:10.1109/TII.2009.2027013.

Cugola, G. and Margara, A. (2012). Processing flows of information: From data stream to complex event processing. *ACM Comput. Surv.*, 44(3), 15:1–15:62.

Dai, Q., Zhong, R., Huang, G.Q., Qu, T., Zhang, T., and Luo, T.Y. (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., Christensen, J., Vyatkin, V., and Dubinin, V. (2014). Function block implementation of service oriented architecture: Case study. In *Industrial Informatics (INDIN), 2014 12th IEEE International Conference on*, 112–117.

He, W. and Xu, L.D. (2014). Integration of distributed enterprise applications: A survey. *Industrial Informatics, IEEE Transactions on*, 10(1), 35–42.

Hill and Smith (2009). Performers at ISA Expo 2009, Enterprise integration track.

Hohpe, G. and Woolf, B. (2003). *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley, Englewood Cliffs, NJ.

IBM (2007). Transform to an on demand business thru IBM business integration solution. IBM Business Continuity Solutions Seminar.

IEC (2003). IEC 62264: Enterprise-control system integration, Part 1: Models and terminology. IEC Standard.

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

ISA (2009). ISA95: Enterprise-control system integration, Part 1: Models and terminology. ISA Standard.

ISO (2013). ISO 20140: Evaluating energy efficiency and other factors of manufacturing systems that influence the environment - Part 1: Overview and general principles. ISO Standard.

ISO (2014a). ISO 22400: Key performance indicators (kpis) for manufacturing operations management - Part 1: Overview, concepts and terminology. ISO Standard.

ISO (2014b). ISO 22400: Key performance indicators (kpis) for manufacturing operations management - Part 2: Definitions and descriptions. ISO Standard.

Kuhn, R. and Allen, J. (2014). *Reactive Design Patterns*. Manning Publications, MEAP 2 edition.

Leitao, P., Marik, V., and Vrba, P. (2013). Past, present, and future of industrial agent applications. *Industrial Informatics, IEEE Transactions on*, 9(4), 2360–2372. doi:10.1109/TII.2012.2222034.

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

Michelson, B.M. (2006). Event-driven architecture overview. *Patricia Seybold Group*, 2.

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

Perez, J., Jimenez, J., Rabanal, A., Astarloa, A., and Lazaro, J. (2014). FTL-CFree: A fuzzy real-time language for runtime verification. *Industrial Informatics, IEEE Transactions on*.

Ribeiro, L., Barata, J., and Mendes, P. (2008). MAS and SOA: complementary automation paradigms. In *Innovation in manufacturing networks*, 259–268. Springer.

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

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

Theorin, A., Hagsund, J., and Johnsson, C. (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.

Theorin, A., Ollinger, L., and Johnsson, C. (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*, volume 472 of *Studies in Computational Intelligence*, 213–228. Springer Berlin Heidelberg.

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

Welke, R., Hirschheim, R., and Schwarz, A. (2011). Service-oriented architecture maturity. *Computer*, 44.