TAMPERE UNIVERSITY OF TECHNOLOGY

JORGE A. GARCIA IZAGUIRRE MONTEMAYOR

A COMPLEX EVENT PROCESSING SYSTEM FOR MONITORING OF MANUFACTURING SYSTEMS

Master of Science Thesis

Examiner: Professor José L. M. Lastra

Examiner and topic approved in the Automation, Mechanical and Materials Engineering Faculty Council meeting on 7 Mar 2012

Abstract

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Information Technology
GARCIA IZAGUIRRE MONTEMAYOR, JORGE A.: A Complex Event Processing system for monitoring of manufacturing systems
Master of Science Thesis, 84 pages, 12 Appendix pages
February 2012
Major: Factory Automation
Examiner: Prof. José Luis Martínez Lastra
Keywords: complex event processing, event-driven architecture, service oriented architecture, production engineering, web services, factory automation, OPC-UA, DPWS

Future manufacturing systems will require to process large amounts of complex data due to a rising demand on visibility and vertical integration of factory floor devices with higher level systems. Systems contained in higher layers of the business model are rapidly moving towards a Service Oriented Architecture, inducing a tendency to push Web Technologies down to the factory floor level. Evidence of this trend is the addition of Web Services at the device level with Device Profile for Web Services and the transition of OPC based on COM/DCOM communication to OPC-UA based on Web Services. DPWS and OPC-UA are becoming nowadays the preferred options to provide on a device level, service-oriented solutions capable to extend with an Event Driven Architecture into manufacturing systems. This thesis provides an implementation of a factory shop floor monitor based on Complex Event Processing for event-driven manufacturing processes. Factory shop monitors are particularly used to inform the workshop personnel via alarms, notifications and, visual aids about the performance and status of a manufacturing process. This work abstracts the informative value of the event-cloud surrounding the factory shop floor by processing its content against rules and formulas to convert it to valuable pieces of information that can be exposed to business monitors and dashboards. As a result, a system with a generic framework for integrating heterogeneous sources was reached, transforming simple data into alarms and complex events containing a specific context within the manufacturing process.

Preface

In this space I would like to thank all of the people that have supported me during the development of this work.

First, I would like to thank my wife Eija who has always been at my side supporting me regardless of the time this thesis has consumed from my personal life. Always pushing me forwards and making me improve constantly as a person and as a husband.

I would like to express my gratitude to Prof. Jose L. Martinez Lastra for the opportunity to develop this work under his research group. Also thank the European research project PLANTCockpit who has founded this work.

Thanks to Andrei for guiding me throughout the development of this thesis work. Also for all the discussions related to this work and the projects.

I would like to thank Johannes and Axel for helping me debugging devices and giving programming tips as well for the long discussions we had regarding this topic. Also to the rest of my co-workers that have supported me by standing my constant questionings.

To Hanna, Taina and Sonja for helping me with all of the bureaucratic issues as well for trip planning and advices. Also, thanks to Matti who has been providing material and support un the conveyor line installation.

También quisiera agradecer a mis familiares y amigos que me han apoyado siempre y aun estando tan lejos de casa. A mi padre y a mi madre que me han dado la oportunidad de tener una educación superior la cual me ha abierto puertas para salir adelante. A mi hermana y a mis tíos por los momentos y enseñanzas que hemos compartido y que seguiremos compartiendo.

Jorge Andres Garcia Izaguirre Montemayor (B.Sc.)
Tampere, Jan 2012

Table of Contents

List of Figures

List of Tables

Acronyms

AI	Artificial Intelligence
BAM	Business Activity Monitor
BI	Business Intelligence
BMM	Business Motivation Model
BPM	Business Process Management
CAMX	Computer Aided Manufacturing using XML
CEP	Complex Event Processing
COTS	Commercial-Off-The-Shelf
COM	Component Object Model
DCOM	Distributed Component Object Model
DFS	Depth-First Search
DPWS	Device Profile for Web Services
EC	Electronic Commerce
EDA	Event-Driven Architecture
ED-SOA	Event Driven- Service Oriented Architecture
EIB	Enterprise Integration Backbone
EPA	Event Processing Agent
EPL	Event Processing Language
ERP	Enterprise Resource Planning
ES	Expert Systems
ESB	Enterprise Service Bus
ESP	Event Stream Processing
EXI	Efficient XML Interchange
FA	Factory Automation
IT	Information Technology
JDL	Joint Directors of Laboratories of the US Dep. of Def.
KB	Knowledge base
KPI	Key Performance Indicator
MA’s	Mobile Agents
MES	Manufacturing Execution System
MRL	Model Representation Language
OPC	OLE for Process Control

OPC-UA	OPC-Unified Architecture
PLC	Programmable Logic Controllers
RFID	Radio Frequency Identification
SCADA	Supervisory Control and Data Acquisition
SEP	Simple Event Processing
SN	Sensor Networks
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SUO	System Under Observation
UDP	User Datagram Protocol
UI	User Interface
WGLA	Weighted Granular Level of Appropriateness
WS	Web Service
WSA	Web Service Architecture
WSD	Web Service Description
WSDL	Web Service Description Language
WSN	Wireless Sensor Networks
W3C	World Wide Web Consortium
XML	eXtensible Markup Language

1. Introduction

This chapter provides a solid background on the thesis work domain preparing the reader to understand the problematic with a clear definition of the problem and the justification of work. The work is described by setting objectives followed by a methodology which intends to achieve such objectives defining assumptions and limitations.

1.1 Background

Since the industrial revolution, manufacturing industries has taken an important role in economy with a significant effect on the environment and society. Nowadays, only in the EU, there are around 430 thousand manufacturing enterprises providing 28 million people with jobs and generating about 20% of the EU output [Jovane et al. 09]. These enterprises are the motor of one of the most important industries in Europe, however, they require of constant tuning in order to maintain themselves against the constant increase of demand that is being subjected by the market.

In the last decades, there has been an important progress in Information Technology (IT) which has directly modified the internal functionality of manufacturing companies. The introduction of IT in combination with classic manufacturing systems has evolved into smart automated systems capable to react automatically to certain production specific situations to increase productivity. Such systems are the result of an attempt to counteract the demands and requirements of a constantly evolving market. Due to this, many companies have invested greatly in automation technologies to provide more intelligence into their systems while at the same time generating the need for monitoring their automated processes. The introduction of Supervisory Control and Data Acquisition (SCADA) provided visibility to such processes by generating human-machine interfaces, giving a graphical interpretation of the process status. SCADA systems entirely rely on data acquisition from field devices to do these representations. Throughout the time, many industrial communication protocols have been developed due to the lack of standardization in the field. Many vendors have developed their own communication protocols that do not allow interoperability with others, a huge problem at the time of trying to interoperate with devices and applications from different vendors started. Higher integration costs and programming experts were needed in order to integrate to different systems. The need for interoperability made Ole for Process Control (OPC) to appear as a specification to standardize secure communication between applications and field devices [Hannelius et al. 08]. Costs for integration time were reduced considerably, leading vendors to adopt this specification.

Recently, web-based technologies have become widely used and reliable, drawing attention within the factory automation domain. In particular, Web Services

(WS), which are the preferred technologies to implement a Service-Oriented Architecture (SOA) according to Jammes & Smit [2005]. WSs provide flexibility and interoperability of distributed systems regardless of the vendor architecture, suiting perfectly to the tendency of having decentralized intelligent components implemented throughout different layers of the business model.

Nowadays, many Business Intelligence (BI) and Manufacturing Execution Systems (MES) solutions are available, and many of them rely on SOA paradigm. Because of this, there has been an inclination for vertical integration of SOA. This caused a transition of SOA into the lower levels of the automation layer. Evidence of this trend is the development of Device Profile for Web Services (DPWS) and Ole for Process Control- Unified Architecture (OPC-UA). Such protocols propose the addition of WS down into the device level as a result of the increasing capabilities of Programmable Logic Controllers (PLCs). WSs include event mechanisms to devices which can act as an interface for interoperability inside an enterprise, carrying valuable information about a process. Nonetheless, new challenges come across with these implementations due to an increasing amount of event data being generated which is surrounding the business IT systems creating a so called event cloud. Events contain information which sometimes is not relevant on its own and logging its data makes analysis more complicated. In the IT domain, events have been widely studied. Complex Event Processing (CEP) has been introduced, tested and adopted in monitoring tasks with the aim of avoiding the inaccurate task of analyzing substantial event logs manually, demonstrating to be a stable set of tools capable to filter, map and generate higher level events with more representative information of what is happening on a system [Luckham & Frasca 98].

Overall, SOA solves the interoperability and decentralization problems among applications by loosely-coupling components, while Event-Driven Architectures (EDAs) promises to extend SOA and complement the integration among other SOA based systems by coupling components with events [Van Hoof 07]. Extensive research is currently studying the possibility of a cross-layered enterprise visibility solution. Moving towards EDA has started a new need to implement complex event processing methods as automatic data aggregators for cross-layer data interpretations of many heterogeneous event sources in factory automation.

1.2 Problem Definition

Current market demands are pushing current automation systems to assume a position where visibility of every component of a company is needed, from process control up to business management. Product customization, maintenance, quality control and, business monitoring are some examples of requirements that pushes towards a holistic manufacturing monitoring which can provide a proper visibility [Hardy 08]. Due to this rising demand on visibility and vertical integration of factory

floor devices with higher level systems, future manufacturing systems will require to process large amounts of heterogeneous data describing different states and situations on different levels of a business.

The bond between higher level systems and lower level systems is made currently by system experts. Meetings are held in order to take further actions across the layers of automation defined in ANSI/ISA-95 [Isa 95]. This non-automatic decision making and cross-layered monitoring of the current status of a business takes loads of valuable time. Information across layers is critical for manufacturing and it must be available at any time in any layer of the business model.

The need to integrate and correlate information automatically across layers of the business model is arising. Transition from local monitors to a holistic monitoring solution is required, increasing the reaction speed a company to act accordingly to situations that can be triggered by different factors within a company.

1.2.1 Justification for the work

Integration of the different levels of automation for increasing visibility is a current topic of research. Factory floor systems are as well as Manufacturing Execution Systems (MES) and Enterprise Resource Planning (ERP) moving towards a Service Oriented Architecture. These systems together can construct what it is now the modern business architecture. This transition to SOA allows the integration of the relatively new device level SOA with the other higher level systems, this to improve a holistic visibility of an enterprise.

SOA systems can be extended and interoperate with EDA [Van Hoof 07]. The inclusion of EDA in the automation model can provide a framework for horizontal and vertical integration of SOA based systems by including an event processing manager to handle the “cloud of events” which surrounds a business. Although event processing technologies has been already studied as solution of cross-layer visibility and control for Event Driven Manufacturing (EDM) [Walzer et al. 08]; interoperability among different factory-wide integration specifications is pending. This leaves a gap in the study of horizontal SOA integration on the device level. Furthermore, a more flexible implementation of SOA on factory floor can be achieved by adding up the benefits of different specifications [Colombo et al. 10]. Due to this, the inclusion of heterogeneous data aggregation and event processing is required in order to achieve a more flexible and complete integration of SOA based factory floor systems with the rest of the components distributed along the companies.

1.2.2 Problem statement

As previously mentioned, there is a need to enhance the enterprise visibility by aggregating data from heterogeneous sources within the factory floor level. A direct

connection of factory floor systems into higher level systems may lead to a state of "IT blindness", due to the quantity of data generated at this level [Luckham 02] [Luckham 04]. This instead of helping will hinder the global visibility which is targeted. In order to provide better visibility to higher levels it is necessary to process hundreds of events incoming from the factory floor before making them available to other systems. Implementation of EDA and Event processing techniques can be introduced to overcome this situation, but at the same time it prompts the following questions which this thesis tries to solve:

- *How to simplify integration of different heterogeneous information sources into a single event-processing system?*
- *How to automate event aggregation to provide new higher level event generation?*
- *How to leverage the factory-shop floor information?*
- *What components could create a framework that can allow event management?*

1.3 Work description

1.3.1 Objectives

1. Design and implementation of a mechanism for unification of event streams of heterogeneous systems into a common processing engine.

2. Implementation of an event processing engine with automatic aggregation capabilities.

3. Design and implementation of a framework for automatic web services / OPC-UA device integration with a complex event processor for improved visibility of the factory floor process.

4. To define the principles for addition of new information sources on the factory floor

5. The event processing engine should be capable of storing event information.

1.3.2 Methodology

Study and implementation of web based factory floor information systems

A detailed study on integration protocols for factory floor information systems is done to understand similarities, benefits and drawbacks of each approach. In addition, a detailed analysis is made on current solutions available for event processing implementations on these to communication specifications. Finally, a set-up of factory floor information systems in a distributed line is performed to establish the test bed for this thesis.

Selection of an event processing platform

An extense research of available event processing platforms is done in order to compare their capabilities and select a platform that suits the integration approach. During this methodology step it has to be considered that some tools require licensing and not all the features might be available. Open source solutions may be suitable as well considering that performance is not on the scope of this thesis. Moreover, programming languages and input data formats also have to be well thought-out for fast prototyping for factory floor information system integration.

Design and implementation of the event manager

Based on the information gathered by the previous steps, the selection of technologies, tools and platforms is done in order to develop a processing engine capable of automatic event aggregation and processing. Afterwards, components of the proposed framework are implemented on a discrete manufacturing line to test its capabilities.

Definition of requirements for factory floor system integration to the event management platform

Considering the framework design, generic requirements are defined and mapped for the specification of a methodology to implement heterogeneous information of factory floor data into higher levels.

Empirical study

The empirical study was performed over a light assembly line. Such line consists of lifters, workstations, conveyors and cross-conveyors that are controlled with multiple devices that communicate with heterogeneous technologies. The devices provide subscriptions to notifications related to the process status. Notifications generated were submitted to a sink during process orchestration, pushed through complex event statements in a processing engine which filtered and aggregated such notifications for event patterning and relevant information extraction.

Tests were performed in such system to detect complex situations extracted from atomic notifications as well as to prove the automatic registration of events from both communication technologies during the engine configuration process.

1.3.3 Assumptions and limitations

The current study applies for manufacturing systems composed of modular segments controlled with DPWS and OPC-UA communication technologies. The development scope does not go further than the automatic data aggregation for monitoring purposes using complex event processing. Event Processing Agents (EPA) were studied but not implemented during this development.

Assumption 1: Modular components in the manufacturing line must be able to provide subscription to notifications.

Assumption 2: The manufacturing system is composed of two or more heterogeneous sources of notifications.

Assumption 3: Notifications received contain more than a single value related to the process.

Assumption 4: Orchestration of the process runs independently from the monitoring tool, no feedback is required to keep the process running.

1.4 Thesis Outline

This thesis work is structured as follows. Chapter 2 presents a literature review containing concepts and technologies relevant for this work. Chapter 3 presents a methodology approach for the development of this work. Chapter 4 describes the technical implementation as well as the use case chosen for testing purposes. Chapter 5 presents the results obtained for the tests performed. To finalize, Chapter 6 presents conclusions, future work and final thoughts.

2. Literature review

This chapter introduces technologies and tools related to this thesis work. Technologies within the scope of this study are described and explained with industrial examples made by the research community. Furthermore, the tools in the scope of this study consist of Event Processing technologies, its implementations and concepts will be covered in this chapter as well.

2.1 Monitoring of distributed Manufacturing Systems

The alignment of process performance and equipments state with business objectives has always been of top priority within the manufactory industry. Keeping track on process variables and performance is critical to analyze and predict the possible effects and deviations of these goals. Monitors are the main bridges between process and humans. Because of this, monitors are considered to be a critical part of any business process. Currently monitors have evolved in business applications as performance dashboards as explained by [Eckerson 11]. Such dashboards allow business people to monitor processes, analyze cause of problems and manage resources to improve decision making. But until now such applications have a rough connection with the layers of automation, limiting the visibility and crippling the reactivity of an enterprise. As mentioned by [Panetto & Molina 08, Karnouskos et al. 09], proprietary solutions that currently exist to achieve enterprise integration are the main cause of this problem. Thus, a more heterogeneous enterprise integration and interoperability in manufacturing systems could be the solution to this problem. Such assertion has been the starting point and trend for future research focusing in aggregation and unification the information across the factory without compromising reliability and performance of the system.

Early studies by Weaver [2001] show that Electronic commerce techniques have been previously used to solve the monitoring issues of Factory automation. Requirements such as universal data access, ubiquitous programming, data security and, user authentication can be solved using solutions borrowed from internet-based Electronic Commerce (EC) such as HTTP, IP, HTML and XML. In Weaver's work, a web server was fed with factory information which later was later accessible by a web browser using java-applets showing a tendency of the time to move toward web environments for factory monitoring. This tendency has opened several research branches such as the analysis of current network and systems monitoring methods for implementation in Factory Automation and manufacturing domains [Park 10, [Balasubramanian et al. 09].

The main contribution of this thesis work surrounds on a heterogeneous approach for monitoring of distributed automated manufacturing systems. Based on the main

trends previously explained, a review of current monitoring techniques is essential in order to scope, choose and implement the best technique/mode/architecture available for the realization of this work. However, a detailed classification is out of the scope of this work; however this will provide helpful input to the methodology for selection of the fittest approach.

Since the nature of the manufacturing monitoring systems is application dependant, it is complicated to develop a taxonomic classification of these systems. Due to this, a short methodology as an attempt for analysis and classification of current monitoring works was applied. This methodology consisted in the research from several sources of information using several keywords related to factory monitoring to compile the work related to this area. Subsequently, the research results are filtered to the level of relevance in the field to finally highlight commonalities among the approaches. However, it must be considered as a coarse classification due to the extensive nature of this topic. It is only valid under the scope of this work which objective is the identification of available monitors for distributed systems.

2.1.1 Classification of monitors

Several works and publications have put in evidence that monitoring modes, techniques and, architectures tend to differ depending on the process or equipment demands, communication constraints and distribution of control as seen in different works [Goodloe & Pike 10, Leitao 09, Liotta 02, Bernhard 02, Han 03, Kusunoki et al. 98]. Due to this reason it is proposed to divide monitors in three identified classifications that contribute for later implementation decisions.

2.1.1.1 Monitoring mode

Classifying monitors by mode can be one of the proposed categories previously mentioned. Figure 1 depicts the hierarchy of monitoring types described by [Goodloe & Pike 10]. In this classification, the types of monitors are separated by the method of obtaining and handling data. Offline monitoring manipulates information once it has been collected from the process, making the required calculations while being disconnected from the process. On the other hand, online monitors focus on runtime and even real-time information for acquisition, processing and display of data.

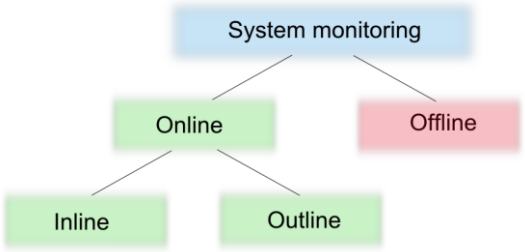


Figure 1: Monitoring modes as explained in [Goodloe & Pike 10]

Online monitoring can be divided in two sub classes as well. In one hand, Inline monitor proposes the addition of monitoring code within the execution code. In Havelund & Roşu [2004] work, they have proposed the use of inline monitoring for testing the finite execution trace of events generated by executing programs to detect errors. Using PathExplorer (PaX) as the monitoring environment, it allowed adding extra code in the algorithm execution program that allowed the identification of errors. According to the authors inline monitoring in this application has higher precision than offline monitoring due to the fact that one can know where the event comes from in the execution program.

Alternatively, an outline monitor executes another external process for monitoring. Pellizzoni et al. [2008] gives a simple example of the use of online outline monitoring for Commercial-Off-The-Shelf (COTS) components. Runtime verification of the COTS peripherals can be achieved by the use of external hardware that is able to predict and detect disturbances in the transmission bus. In this case the hardware referred as monitor module; act as a non-intrusive external monitoring process.

According to [Trinitis et al. 00 and DAWAC 05], online tools provide more benefits than offline monitoring. One of the benefits is that online monitoring is running in parallel while executing a process, hence it is possible to adjust and guide the trajectory of the processes during process execution. However this later is more expensive due to the needs of exclusive hardware and support for manipulation of the target systems making it a heavy system which lacks of portability. On the other hand offline diagnostics can provide more accurate results due to the time independence it has with the System under Observation (SUO) [Grubic et al. 08].

Substantial research has been done due to the benefits of online monitoring [Barringer et al. 04, Bodden 05, Fei et al. 06, Barbon et al. 06]. In particular, Liotta [2002] on his work tries to define the real-benefit of using Mobile Agents (MAs) for monitoring of networks. His work defines an algorithm for network adaptability of agents following the premise that a distributed monitor can become fault-tolerant by dividing their task into several monitors, diminishing the probability to enter into a faulted state. Many other publications have later publish on this subject recommending the distribution of control and monitoring tasks as surveyed and analysed by [Leitao 09]. However the level of required adaptability is dependent on the dynamic behavior of the monitored system. This leads to the fact that not always the most complex solution is the fittest in every case.

A collection of seen advantages and disadvantages of different monitoring modes are shown in Table 1.

Table 1: Contrast of monitoring modes [Adapted from DAWAC 05]

Monitoring mode	Advantages	Disadvantages	Applicability
Offline	• More precise and complex algorithms can be applied	• Results are useful only for time independent applications • Reaction is only possible with very slow processes	• When no on-line monitor is available for certain parameter • The required frequency of analysis would induce more time for on-line monitor • The economical situation is not favourable to the on-line monitor and if the required frequency does not require a frequent value • Reliability, sensitivity or adequacy of the online monitor is not as good as the laboratory method.
Online / Inline	• Data can be processed during runtime • Simpler localization of flaws	• Monitoring code is embedded with process code affecting performance	• When high frequency of data generated allows early detection of anomalies • When risk of product contamination and human errors is reduced • When response needs to be fast • When some parameters cannot be measured by offline monitoring (grab sampling)
Online/ Outline	• Data can be processed during runtime • Better performance due to distribution of monitoring tasks • Robust data analysis without influencing the process execution • Better fault-tolerance	• More expensive implementations • Difficult to debug	

2.1.1.2 Monitoring architectures

Monitor architecture seem to depend on process requirements, reason why thousands of different architecture proposal exists. Even though several authors have tried to generalize the monitoring architecture of networks [Tang et al. 07 and Zhaohua et al. 07], different monitor architectures keep emerging to attack different problems. Some of them, heading towards a simpler implementation for less critical applications while other trying to bring performance and high fault-tolerance to applications such as hard real-time systems. The process in the end is the one that has to guide to a selection of a fitting architecture. Goodloe & Pike [2010] in his work describe three monitoring architectures to follow in future implementations, which based on his work; they cover most of the distributed monitoring approaches currently available. The categories proposed are consistent with other works found within the research community as seen in [Hongliang et al. 09, monALISA 08]. For the authors three base architectures are: Bus-monitor architecture, Single Process-Monitor architecture and distributed Process-monitor architecture which will be explained using industrial state-of-the-art examples.

Bus-monitor

This architecture is the simplest of the implementations to be described. It consists in a silent monitor connected as part of the system reading messages through a bus. These monitoring architectures are commonly used to find faults in bus protocol messages or as well monitor performance of systems as in [Hongliang et al. 09] where monitoring systems are being interfaced with a CAN bus interface allowing monitoring and control of the SUO. However this architecture may interfere with the control messages of the system. One of the major drawbacks being also the low fault-tolerance, Nonetheless it requires the least hardware implementation making it in the least expensive approach to implement.

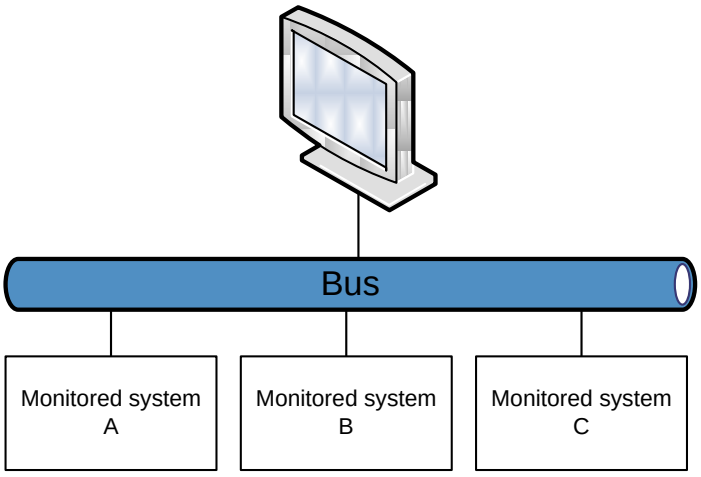


Figure 2: Generic bus monitoring architecture [Goodloe & Pike 10]

Single Process-Monitor

As a resolution for the problems seen in the bus-monitor architecture, the single process monitor intends to solve the messaging interference between data messages and monitoring messages that may be caused by the violation of timeliness guarantees. The main difference comes in the addition of a dedicated monitoring bus. As a result, instead of having a single process sending data to a single monitor, multiple processes send monitoring information via a monitoring bus ensuring functionality of the monitored system.

Implementations of this architecture propose to use different networks such as wireless sensor networks (WSN) for process monitoring. For example, [Ciancetta et al. 10] in his work shows a plug-n-play solution based on web services that consists on a dedicated network for monitoring while the system under observation contains its own control bus for process execution.

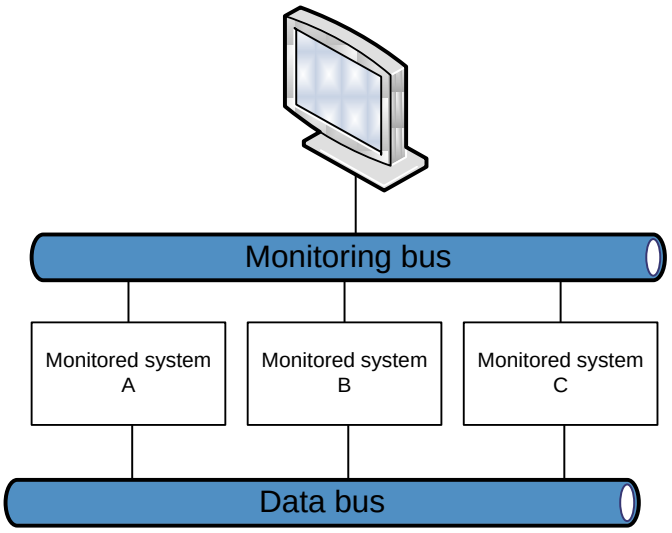


Figure 3: Generic single process-monitor architecture [Goodloe & Pike 10]

Distributed Process-Monitor

This architecture proposes to distribute monitoring tasks among “guardians” that monitor different components of a process. These can communicate with each other allowing increasing fault-tolerance to a next level where the system under observation cannot interfere under any circumstances. Compared to the single monitor approach, reliability is potentially increased by the premise that the probability to fail of several monitors is less than one single instance monitoring the whole system.

Implementations of this architecture are inclined to be of great scale and with variable number of composite monitored systems. monALISA [2004] is a good example of the latest massive deployment of agents globally. Based on JINI and Web Services technologies this architecture is able to provide complete monitoring,

control and global optimization services for complex systems. The high reach of scalability is determined by the capability of this system with a multi-threaded engine to host loosely coupled self-describing dynamic services.

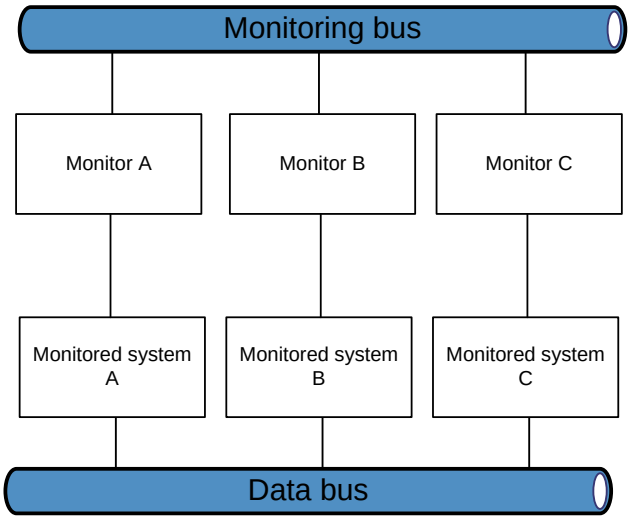


Figure 4: Generic distributed process monitors [Goodloe & Pike 10]

2.1.1.3 Monitoring techniques

According to Liotta [2002], management and monitoring of future networks is being affected by factors such as scalability, topology dynamics and diversity of complex services in heterogeneous networks. Conventional approaches for network monitoring using management protocols and distributed object technologies cannot satisfy requirements for future networked systems. [Philippe et al. 00] provides an extensive review of management technologies where four main techniques can be identified as shown in Table 2.

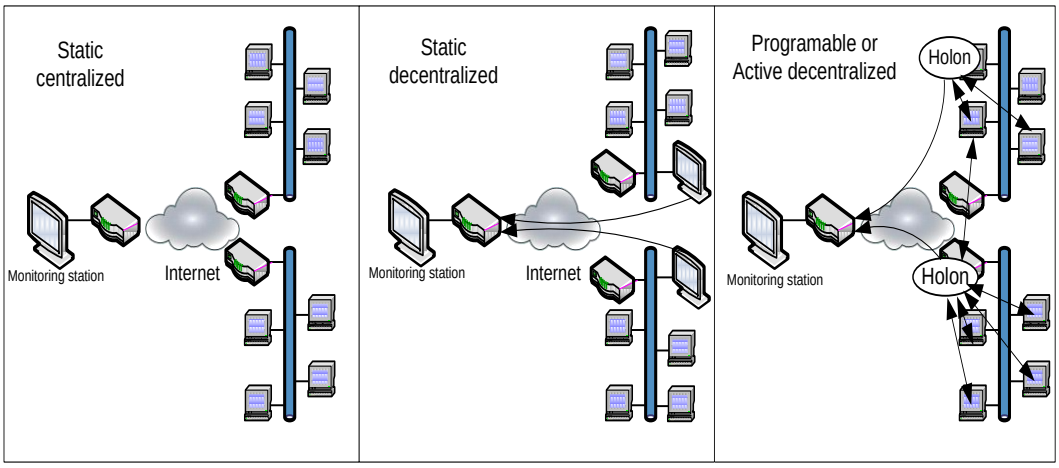


Figure 5: Monitoring techniques [Adapted from Liotta 2002]

Table 2: Monitoring techniques [Adapted from Philippe et al. 00 and Liotta 02]

Monitoring technique	Description & drawbacks
Static centralized monitoring	One main monitoring station, every SUO is communicated directly. Used for small-scale networks using for example Simple Network management Protocol (SNMP). **Drawbacks:** • Limited responsiveness and accuracy. • Lack of scalability • Concentration of management intelligence (single point of failure) • Bottleneck • When using polling approach, it limits the tracking of problems in a timely manner and overflowing networks even when no change has happened. (SNMP)
Static decentralized monitoring	Consists of hierarchical management architecture where a main monitor is communicating with distributed area monitors. CORBA and JAVA-RMI are examples for implementing this technique. **Drawbacks:** • Monitoring functionality is restrained to simple and rudimentary operations. • Low adaptability to network changes • Marginally better scalability • Limited level of decentralization
Programmable decentralized Monitoring	This technique proposes the use of mobile code in network management. Within the code new management functions can be dynamically introduced in the nodes as needed. The main advantages are the decentralization of tasks and re-configurability of nodes. **Drawbacks:** • Relatively static mechanism (Deploys management logic at start-up) • Deployment logic made centralized
Active distributed monitoring	A system that self reconfigures based on the monitoring system changes. The exploitation of distributed area monitors autonomy to optimize the monitoring tasks while decreasing network traffic and increasing responsiveness and robustness. **Drawbacks:** • This technique does not provide any improvement if the monitored system is not of dynamic nature

From the techniques presented it can be noticeable that they intend to mitigate different problems. Even though, the active decentralized technique seems to have the least drawbacks of all, there has to be a system evaluation beforehand. The selection of a correct technique is directly related to the dynamic and granularity level of the system to monitor [Agarwala & Schwan 06, Liotta 2002]. Khoshkbarforoushha el al. [2010] proposes a semantic model for weighting the appropriateness of the granularity level (WGLA) this setting a basis for quantitative granularity appropriateness analysis. Based on a quantitative analysis one can determine the optimal granularity of services from which would serve as a guide for the monitoring technique selection.

2.1.2 Business Activity Monitors

Business intelligence (BI) is referred to the procedure of any software or IT solution that can transform important information out of data. This solutions show how a business is doing by analyzing historical data, identifying patterns and understanding trends. Current BI products mostly rely in historical data for data mining. Due to this, real-time decision support is not possible with these solutions. Increasing demands and ever changing fast-paced business environments has incorporated the requirement for fast reactivity on business. To overcome with this need, BI has been extended with Business Activity Monitors (BAM). This technologies aim on real-time event-driven business analytics. Getting information from transactional data sources such as Web Services, BAM correlates heterogeneous events that can lead to real-time KPI definitions as well as trend and pattern identification for real-time business reactivity. BAM solutions consist of dashboards that can display business information. One of the core technologies that BAM integrates is CEP. Rules correlate, aggregate and analyze data into information real-time and directs it to BAM dashboards.

According to Bayer [2009], CEP platforms commonly provide connectivity to custom applications through adapters. So before choosing a CEP engine it is necessary to evaluate a CEP solution in the context of the integration architecture guidelines compatible to the IT systems. For an event processing architecture it is better to consider first a mixture of SOA and EDA because they make business events available to BPM, BAM, and CEP tools in a standardized way. A possible combination of EDA and SOA would enable a layer of high-value services that could have a visual impact in the business.

As seen from Figure 6, a typical CEP architecture has business events entering the CEP engine in form of data streams incoming from an EDA foundation. The integration adapters capture events starting the sequence of activities the CEP architecture performs. Events are transported to the CEP engine through messaging or web services. Rules and patterns create aggregated or complex event that could be sent to dashboards, BPM, BAM or a custom application.

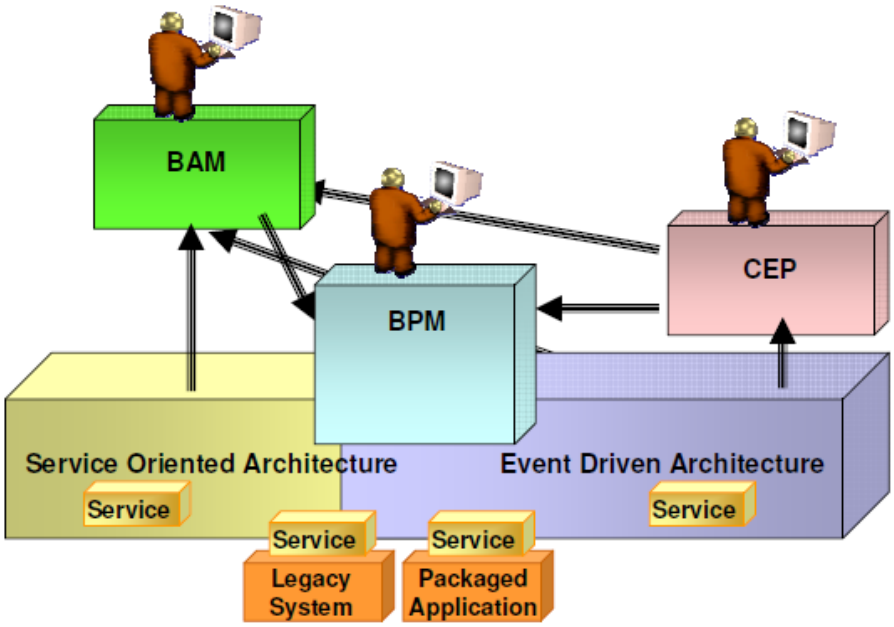


Figure 6: Event Processing Architecture [Bayer 09]

Due to the relatively new concept of service orientation in shop-floor devices, few works on BAM integration with shop-floor have been introduced. This leaves a research gap which this work intends to fill.

2.2 Towards distributed system integration

System distribution has been taking place during the past few decades. Its benefits combined with the continuous development of embedded systems, have stirred its applicability in system architectures. The main advantages of system distribution do not only involve performance and speed improvement; its mayor contribution comes from the reliability and scalability that a system can reach while hiding its underlying intricacies and heterogeneous nature.

By definition, a distributed system in software engineering is:

"A collection of independent computers that appear to its users as a single coherent system"

Tanenbaum & van steen [2006]

The definition may lead to the assumption that distributed systems are a networked bundle of computers. However, two categories may be divided from this concept: Tightly couple components and loosely-coupled components. Tightly-coupled components are commonly related to the management of homogeneous multiprocessor computers. Usually they maintain a global view with computer resources. On the other hand, Loosely-couple components consist of a bundle of

heterogeneous multicomputer systems allowing local services to be available to remote clients.

Distributed systems solve several of the recurring problems caused by centralized systems while at the same time adding different challenges. Single points of failure in the system are resolved, also performance, scalability and, reliability is improved. Nevertheless, management of the resources increases in complexity due to the networked nature of the distributed system. Information on distributed systems may be endangered more than centralized systems. In a nutshell, the main challenges to deploy a successful distributed system can be summarized in four main aspects that need to be ensured: secure communication, fault-tolerance, replication, coordination and management of systems.

Subsequent from distributed computing era, several design principles and paradigms have been developed in the last decade. The concept of distributed services along networked loosely-couple components started to stand out with the beginning of service oriented architectures, and currently continues to be adopted and extended with other methodologies and concepts such as Event Driven Architectures and Event-Driven Service Oriented Architectures.

Nowadays, service orientation and cloud computing towards a fully integrated enterprise is an ongoing topic of research. Transformation from the classical production-oriented manufacturing towards a service oriented manufacturing is taking place. The key of this transition comes from the development of manufacturing clouds based on the capabilities that SOA provides to the whole enterprise [Li et al. 2010]. The following subsections will cover in detail the architecture paradigms mentioned here and its current state on manufacturing systems.

2.2.1 Service-oriented Architecture

2.2.1.1 Overview

Starting as a concept brought by the Information and Communication technologies sector, SOA opened new perspective for a high-level service-based communication infrastructure. According to the OASIS [2006] reference model for service oriented architectures the definition of SOA is:

"A paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. It provides a uniform means to offer, discover, interact with and use capabilities to produce desired effects consistent with measurable preconditions and expectations."

SOA paradigm core driver is services by which capabilities and needs are brought together from service provider entities to service consumers. Table 3 lists the set of characteristics that SOA introduces that comprise its design paradigms.

Table 3: SOA characteristics [adapted from Valipour et al. 09]

Characteristics	Description
Discoverable and Dynamically Bound	Ability of a costumer to discover a service during runtime based on certain criteria.
Self-Contained and Modular	Cohesive interfaces allowing a service to be modular in certain context The modules should be decomposed and composed into other modules.
Interoperability	Ability to communicate different platforms and languages with each other.
Loose Coupling	Having few well know dependencies among modules
Location Transparency	Ability to move a service from location to location without the consumer's knowledge.
Composability	Ability to structure modules to assemble applications, federations or service orchestrations.
Self-Recovery	Ability of a system to recover from errors without human intervention during execution.

Concisely, SOA intends to reduce the integration complexity of systems on a heterogeneous environment allowing them to be modular, reusable and flexible. The basic model of a SOA paradigm consists on a consumer and a provider. First, the consumer has to know the location and description of the produces. In order to do that, the provider exposes its service description on a directory allowing customers to obtain it. The interaction between them consists primarily of a request-response message pattern. This is closely related to a client-server paradigm where the producer acts passively and reacts on a consumer's request. Figure 7 summarizes the basic characteristics of a SOA solution.

Figure 7: Basic characteristics of SOA solution [Marechaux 06]

For instance, WS technology provides a standardized vehicle that complies with the core characteristics of SOA. Service loose coupling, reusability and autonomy is achievable by the ever-present platform-independent XML standardized messaging.

Abstraction and a standardized service contracts are contained in a Web Service Description (WSD). Service discoverability is given by the WS-discovery specification. Composability can be achieved by using WS-Metadata, WS-Federation and WS-Policy specifications. To be precise, SOA is not bound to a special technology; several technologies are available as explained by Zeeb et al. 2009]. Nevertheless, Web Services is the preferred technology to implement service-oriented architectures due to the presented set of compliant specifications that allow implementing the majority of the characteristics of SOA. Insight on WS technology will be presented in the following subsections.

2.2.1.2 Service Oriented Manufacture state-of-the-art

Service Infrastructure for Real-time Embedded Networked Applications

The European project SIRENA (Service Infrastructure for Real-time Embedded Networked Applications) is one of the pioneering projects to consider the Service Oriented paradigm in a manufacturing domain. Its main goal consisted in the definition of a framework to seamlessly connect heterogeneous devices and services hosted in such devices. A technology analysis taking place during the project and compiled in Table 3, highlighted DPWS as the promising technology for implementing Service Oriented architecture on a device level.

Table 4: Contrast of SOA compliable technologies [Bohn et al. 06]

Criteria	OSGi	HAVi	JINI	UPnP	WS	DPWS
Plug and Play	-	x	x	x	-	X
Device support	X	x	x	x	-	X
Programming Lang independent	-	x	-	x	x	X
Network media independent	-	-	x	X	x	x
Large scalability	x		x	-	x	x
Security	X	X	x	-	x	x
High market acceptance	X	-	x	X	x	x

Jammes & Smit [2005] reviewed the opportunities and challenges to solve in a Service Orientation. As a two edged sword, making a device visible and reusable to all layers of automation comes with major challenges. One of the major challenges comes in the management domain, where devices shall be managed by a higher-level system to facilitate configuration, monitoring, fault-diagnosis and maintenance. To demonstrate the SO paradigm in industrial devices, in [Lastra & Delamer 06, Delamer & Lastra 07-2, Jammes et al. 06] they present an industrial applicability of DPWS-enabled devices for the exposure of service. For instance, a dose maker implementation demonstrated the composability feature of the services allowing

them to be encapsulated in high-level services. Figure 8 depicts the logical representation of the services provided by different devices. An orchestration component acting acted as a controller, allowed granular services to perform the specific task as shown in [Delamer & Lastra 06].

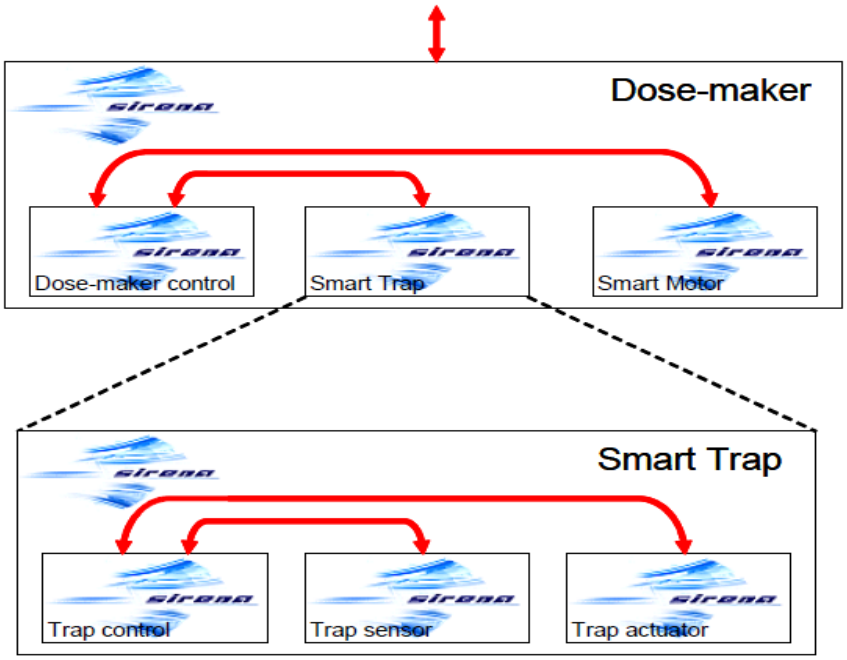


Figure 8: Dose-maker logical implementation [Jammes et al. 06]

Service-Oriented Cross-layer infRAstructure for Distributed smart Embedded devices

As a successive research project, SOCRADES was created based on the concepts and results SIRENA. The aim of the project consisted on the creation of an infrastructure for manufacturing where loosely-coupled smart embedded devices to communicate seamlessly with business level service based components. The closure of the gap between shop-floor and top floor systems conveyed the project towards considerable physical results with the implementation and orchestration of WS-enabled controllers in an industrial environment. De Souza et al. [2008] presented a reference implementation where two devices were able to connect to ERP systems using WS as driving force. The use DPWS in a manufacturing domain allowed seamless vertical integration with business levels. Cannata et al. [2008] presented the impacts of SOA adoption in manufacturing systems. Among them Business Activity Monitoring would be possible due to the effective seamless integration. This integration would allow calculating real-time Key Performance indicators whilst having real-time reaction based on the increased granularity that an enterprise system can consist of.

Cachapa et al. [2010] presents a monitoring methodology in a SOA-based industrial environment. The approach combines two types of monitoring indexes,

Feature-based and model-based. The first one consisting in the wrap-up of sensor data into XML to expose it as web service while the later one intends to compose the services into higher-level services detailing certain model. For his methodology, the identification of services for a monitoring application is crucial and it can be done by a two phase approach combining a top-bottom and bottom-top approaches to define the required services in a monitoring operation.

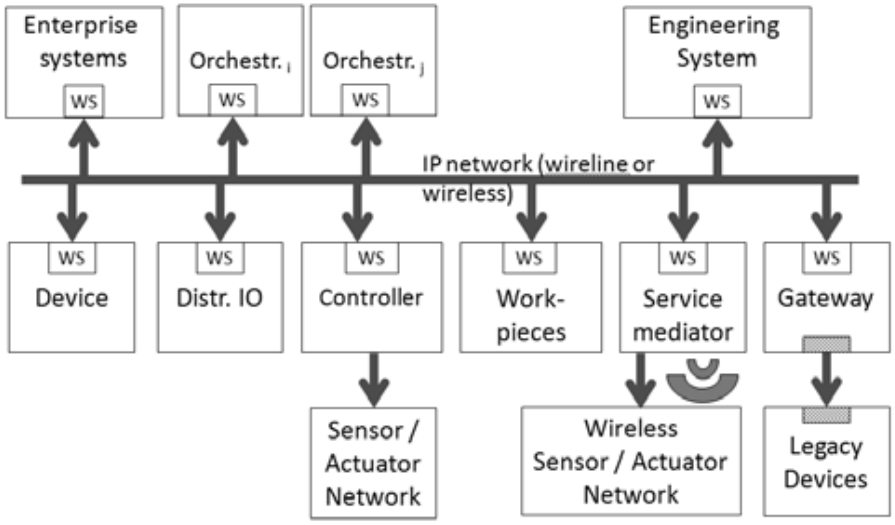


Figure 9: SOCRADES general Architecture [Cannata et al. 08]

Factory-wide Predictive Maintenance in Heterogeneous Environments

Factory-wide data integration can lead to the calculation of situations that cannot be detected by a standalone system. Jakob et al. [10] propose a framework for predictive maintenance that claims to automate the prediction process for fine granular data incoming from various machines. Their approach consists on a Service Oriented generic prediction process that can retrieve data and information from machines and modeling tools that exists already. They use a prediction control process that acts as an orchestrator to execute every step in the prediction process and later map the information in an ERP. Figure 10 shows the stair case architecture of the system as coined by the authors.

Active monitoring is not visible in this approach based on the passive SOA nature of the system. The system has to be invoked and no active health alarming could be generated. On the other hand, an EDA predictive maintenance tool would be able to do.

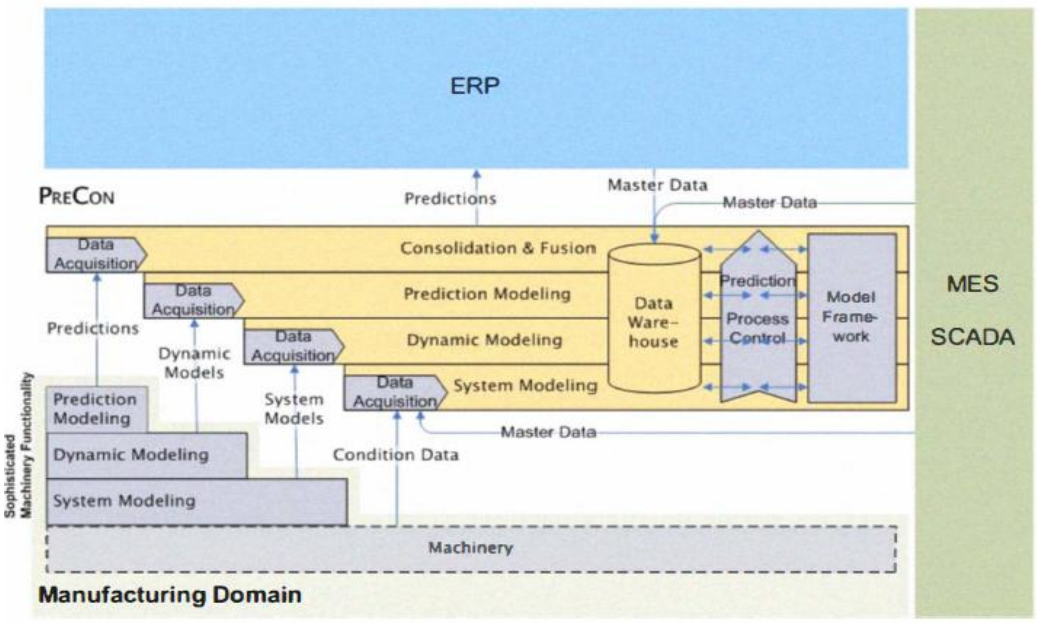


Figure 10: Factory wide predictive maintenance architecture [Krauze et al 10]

2.2.2 Event-driven Architecture

2.2.2.1 Overview

Event-driven Architecture (EDA) is an architectural paradigm that uses events as main execution drivers. This paradigm follows the publish/subscribe model providing asynchronous messaging among components. Differently from SOA, the event-driven paradigm allows components to be extremely loosely-coupled. This concept comes primarily by the fact that publishers of events are not aware of the existence of subscribers and that they only share the semantics of the message [Van hoof 07]. The core implementation components consist of: Enterprise Integration Backbone (EIB), event management tools, event processing engine, event processing rules, service invocation, event transport, event specification and event data. The basic characteristics of EDA can be summarized in Figure 11.

In a simplistic point of view, EDA system interaction consists of: Several consumers that subscribe to a messaging backbone, later publisher posts a message. The messaging backbone (broker) will route the messages subscribers that are interested in that particular message.

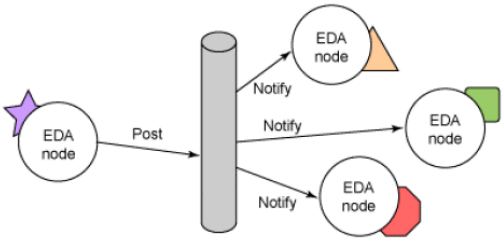


Figure 11: Basic characteristics of EDA [Marechaux 06]

2.2.2.2 EDA in manufacturing

Event-Driven Manufacturing: Unified Management of Primitive and Complex Events for Manufacturing Monitoring and Control

According to Waltzer et al [2010] State-of-the-art event management systems only cover selected levels in the automation pyramid. The architecture shown in Figure 1, proposes an approach for event management that allows high-level systems and low level systems to communicate via events. The solution shows a reference framework for event management that can be separated in 5 main segments: Data acquisition, data processing, data persistence, data exposure and, configuration. The Data acquisition adapters and data exposure adapters are commonly used supply the system with incoming data allowing any external system to be interconnected to the manager. This is a centric approach based on a broker as a backbone system for messaging. A CEP for control and aggregation is proposed as part of the manager closely coupled to a publish/subscribe system to receive events and route its results to the consumer. The configuration of CEP is done by a configuration environment component that defines input and output message relationships among components.

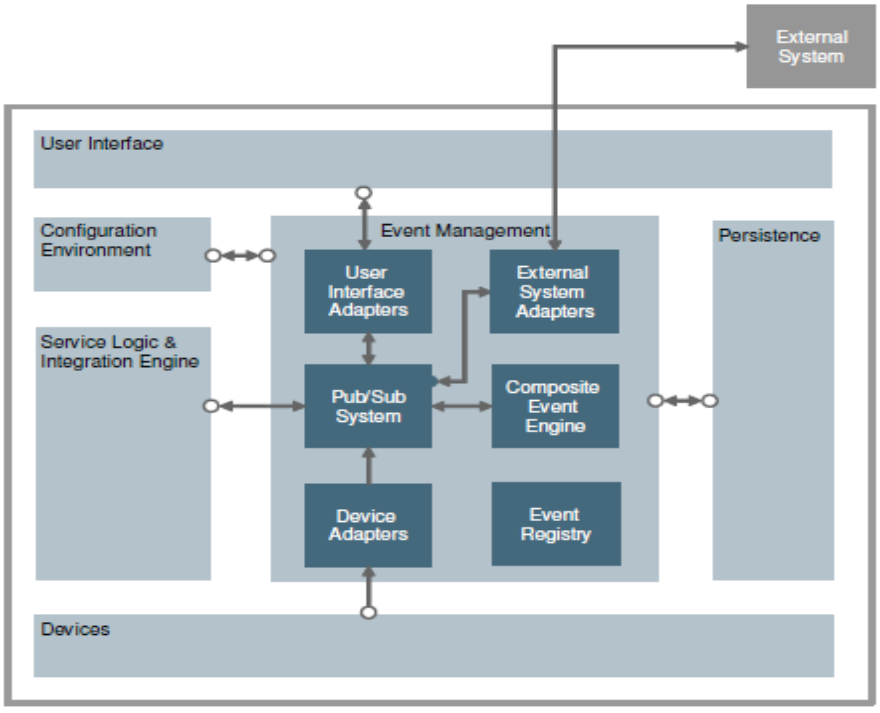


Figure 12: Unified Management Architecture

Combining Service-Oriented Architecture and Event-Driven Architecture using an Enterprise Service Bus

An Enterprise Service Bus (ESB) is an architecture model that allows the interaction of heterogeneous systems acting as a SOA enabler. ESB is currently the preferred model to consider due to the advantages of combining benefits of both

EDA and SOA. It exposes transport, event and, mediation services that facilitate integration using a strict asynchronous communication.

An analysis presented by Marechaux [2006] describes the benefits of implementing an ESB instead of a standalone EDA or SOA. The first benefit presented is the standardized connectivity, which is achieved based on messaging backbones that allow heterogeneous systems to connect. This also provides pervasive integration that bridges systems, following the concepts of ubiquitous computing. Also this architecture model allows reliable integration whilst increasing robustness. Transport services guarantee reliability and transactional integrity which is crucial in a demanding environment. In [Delamer & Lastra 07-1] they propose a optimization for QoS-aware event driven middleware in electronics production. While in [Ming et al. 09], they study the benefits and drawbacks of current SO paradigms systems, proposing an Event-driven SOA (ED-SOA) as an improved solution for plant visibility. According to this author, WS-based SOA systems are not optimal for shop-floor systems considering the inefficiency of request-response mechanism on natural event producing systems such as factory floor equipment. And improved event-triggered service paradigm would be more efficient whilst allowing event processing systems to leverage this complex transactional information into a visible format for decision making.

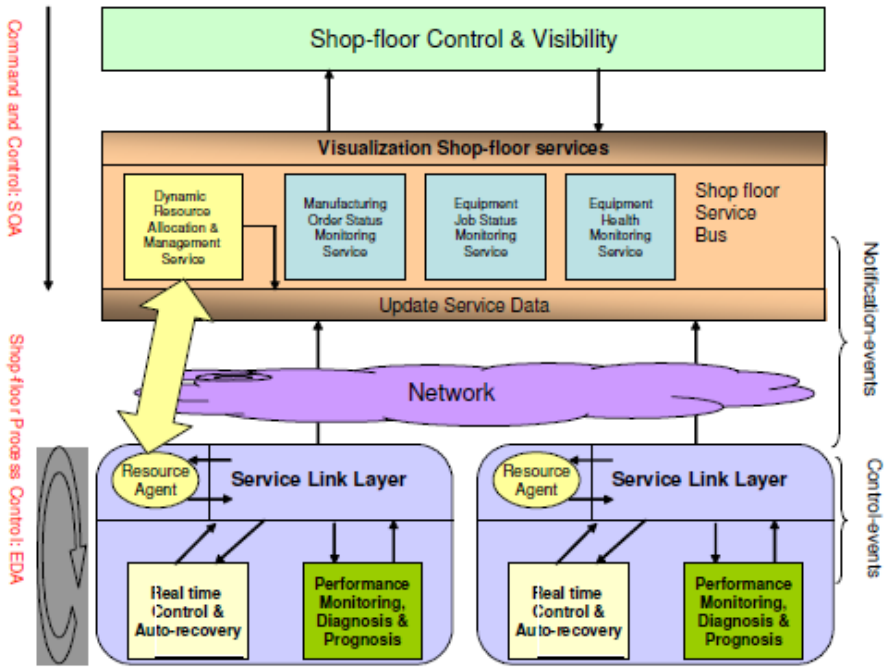


Figure 13: ED-SOA implementation in Factory-shop floor [Ming et al. 09]

2.2.3 Web Services architecture

A Web Service according to the W3C [2004] is a software system designed to support interoperable machine-to-machine interaction over a network. It possesses an interface described in a machine-processable format. In its abstract form it is a

resource characterized by the set of functionality that it provides. Considering an agent as a computational resource, an agent hosts a service in a piece of software of hardware that sends and receives messages. The agent may change but the abstract set of functionalities of the service is maintained.

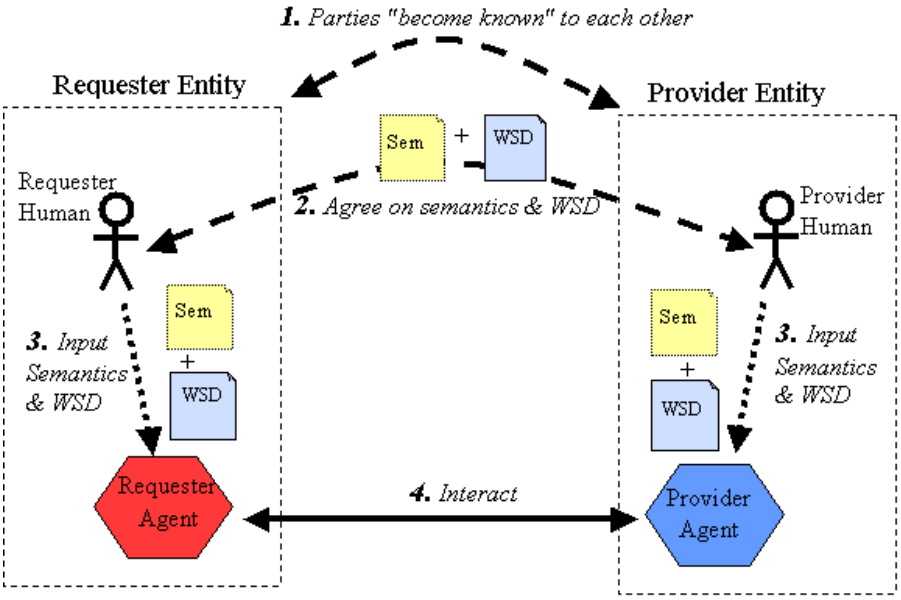


Figure 14: Web services general process [W3C 04]

The scenario presented in Figure 14 is divided in 4 steps which defines the general process for system interaction using Web Services. Firstly, the requester and provider entities discover each other. Afterwards, transaction of contract and semantics is agreed. Subsequently, both parties input the semantics and Web Service description to finally interact with each other. These interactions between entities may be manual or automated.

Web Services Architecture relies on several technologies and specifications such as: Extensible markup language (XML), Simple Object Access Protocol (SOAP) and, Web Service Description Language (WSDL). Also there are several specifications that govern specific process of the web service process interaction such as: WS-transfer, WS-Eventing, WS-Discovery, WS-Metadata, WS-Addressing, and WS-Notification among others.

These technologies will be explained in Section 2.3.1 due to the fact that DPWS is an actual implementation of Web Services Architecture for embedded devices.

2.3 Web-based communication protocols in manufacturing

Considering the results obtained from the previous section it is noticeable the strong trend towards SOA and EDA. The main focus of this section is on the description of the standards available that enable vertical enterprise integration in a manufacturing environment as well as the state of the art implementations by the research community.

2.3.1 Device Profile for Web Services

2.3.1.1 Overview

The device profile for web services is an OASIS standardized device-level protocol that succeeds Universal Plug n' Play (UPnP) for communication of networked embedded devices [Jammes & Smit 2005]. It is built on top of a set of web technologies that together comply in Service Oriented paradigms. This protocol relies in IP, TCP, UDP, HTTP, SOAP and XML making it suitable for cross-platform integration.

DPWS exposes devices as services that can be discovered, invoked or notified by any other networked devices that support Web Services. For instance, a DPWS-enabled device can host services that can be dynamically discovered. At the same time it provides metadata exchange service to load the exposed services into clients. Other feature of DPWS is the compliance with the WS-eventing specification for eventing mechanism; this allows other components to communicate asynchronously via semantically rich events.

2.3.1.2 Web service specifications

SOAP

This protocol is a recommendation from the W3C that once stood for "Simple Object Access Protocol". It encodes data for structured information exchange. The core aspects of SOAP are neutrality and independence. This specification can be send over any transport protocol and it does not depend on any programming model. SOAP structure is XML-based and it consists of an envelope element containing a header element in that holds application-specific information and a body element which conveys the message contents. The platform-agnosticism characteristic of SOAP makes it preferable as transport protocol for Web Services.

Web Service description Language [W3C 01]

This W3C specification is used for describing network services as a set of endpoints operating on messages. It is extensible to allow description of endpoints and their messages regardless of what message formats or network protocols are used to communicate. The WSDL documents are structured in 6 main sections called definitions. *Type definition* describes data types used in the messages. *Message* defines input and output message parameters. These are bound to operations input

message and output message. *Port type* defines the Web Service operations and binds the messages to the operations. The *Binding* specifies interface and defines the transport with the binding style. *Operations* define the actions and the message encoding. *Services* define system functions that are exposed to web-applications. Overall specification is used to define the contractual mechanism in DPWS to expose the services and their endpoint to other clients.

Web Service Eventing

WS-Eventing specification defines the message structure and composition for subscription messages as well as the notification structure.

Eventing in DPWS allows purely asynchronous messaging. Operations with *one-way* message patterns can be subscribed by invoking a subscription request. Subscription request are submitted by the client containing the required subscription setup properties such as duration and event sink endpoints directions.

The process consists in three basic steps. First, a client with knowledge of a service requests a subscription. The hosted service processes the requests and registers the client as consumer. The hosted device responds to a client and finally submits notifications to the registered consumers. The subscription mechanism of DPWS is depicted in Figure 15.

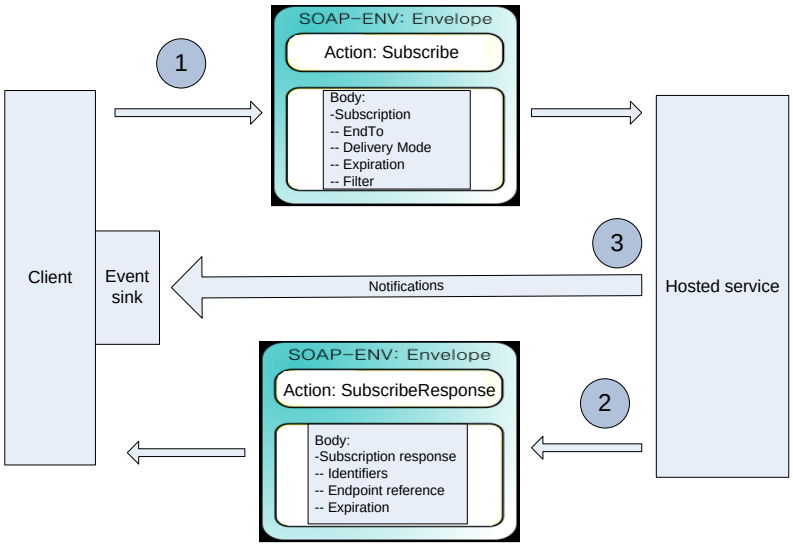


Figure 15: DPWS subscription mechanism

Web Service Discovery

Web service discovery specifies the procedure for message exchanges and conceptual content that a consumer and a service provider must use for detection of each other existence. The WSA does not define a specific technology for discovery. DPWS on the other hand, specifies de use of SOAP-over-UDP to implement the discovery services. In a sample scenario for discovery, a consumer would multicast a

probe message to a broker. The broker would route the message to hosted services. Hosted services would then notify a probe match message to the consumer. Additional information can be found in [W3C 01].

2.3.1.3 Industrial experience state-of-the-art

Device Profile for Web Services for Service Oriented Manufacturing

Machine health monitoring is one example of the important measurements that manufacturers are interested for equipment maintenance. Wireless Sensor Networks (WSN) is a common solution that provides Condition-based Maintenance (CBM) to machinery. According to Sleman & Moeller [2008], the interoperability and dynamic discovery of DPWS can interface 6LoWPAN-based WSN with other IP-networked systems. In their methodology, DPWS can acts like a gateway allowing IP-based clients to subscribe to sensor data. This work is in concordance with solutions described by [Jammes et al. 07] where he points out the cost-effectiveness of a "wrap-and-reuse" basis for industrial solutions rather than a "rip-and-replace". As a gateway, DPWS can interact with any industrial field bus and expose the functionality of its components as web services. This proposes a solution as similar the one that OPC-UA and its address space provide. Nonetheless Jammes et al. [2007] also mentions that the gateway solutions are ad-hoc solutions that work today based on the current equipment restrictions. For that reason, future implementations should follow the SOA paradigm of having "smart devices" up to every corner of the industry. In [Moritz et al. 2009], DPWS was directly integrated to WSN. Results reflected the need of enhancements to the protocol in order to fit in the constrained resource sensor nodes. A set of considerations and profiling is needed to achieve a successful implementation of DPWS in a sensor node. In [Delamer & Lastra 06],

Enhancements for DPWS

Reliability and real-time requirements that commonly are found in the factory shops and manufacturers are the common aspects that DPWS cannot guarantee. The non-deterministic nature of the standard IP-networks is the main cause of this issue. Adding to the issue, the SOAP messages tend to be large and bulky slowing down the message transmission rate.

Because of the previously stated issues, DPWS has been over-criticized regarding the encodings and overheads it relies on. Analysis by [Moritz et al. 10 and Moritz et al. 10-2] has shown that DPWS as it is would not fulfill the requirements to fit in a constrained-resource device environment. Results of his work showed Efficient XML Interchange (EXI) and Fast Infoset (FI) as the most prominent compressions for SOAP messages that should be considered. In [Jammes 11], the author proposes an example implementation of EXI and DPWS, highlighting that including a binary-XML encoding for the DPWS stack could improve considerably

the real-time performance of SOA at the device level. While in [Collado et al. 08] they present an approach for XML data processing without the need of binary encodings for real-time systems requirements.

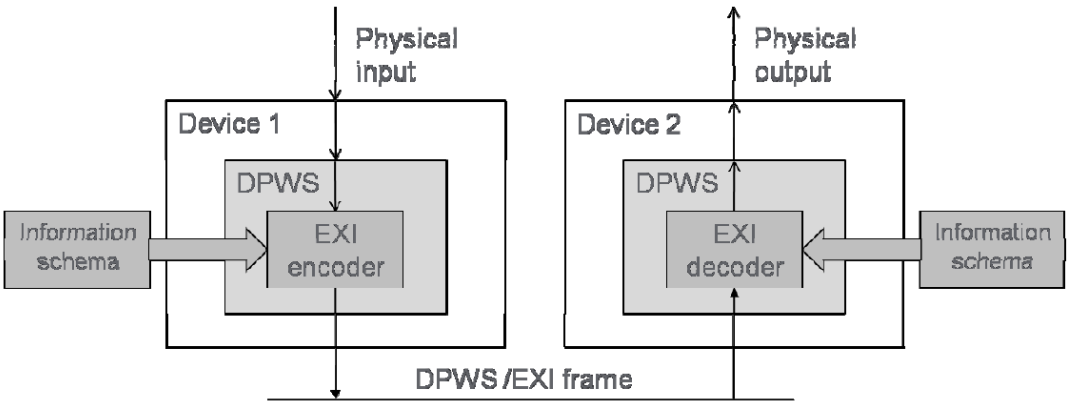


Figure 16: DPWS/ EXI example

2.3.2 Ole for Process Control –Unified Architecture

2.3.2.1 Overview

OPC-UA is the next generation implamentation of the former OPC architecture with the difference of applying cross-platform web services instead of the Microsoft dependant COM/DCOM communication model. As its name Unified Architecture implies, it unified several OPC data models such as Alarm and Events, Data Access and Historical Data Access and other OPC specifications, as a set of services to increase its domain within manufacturing production and business applications. OPC-UA's popularity in the industry is constantly growing due to the premise that it allows from top to bottom level integration whilst providing a backward compatibility with previous OPC solutions that are widely deployed around the world.

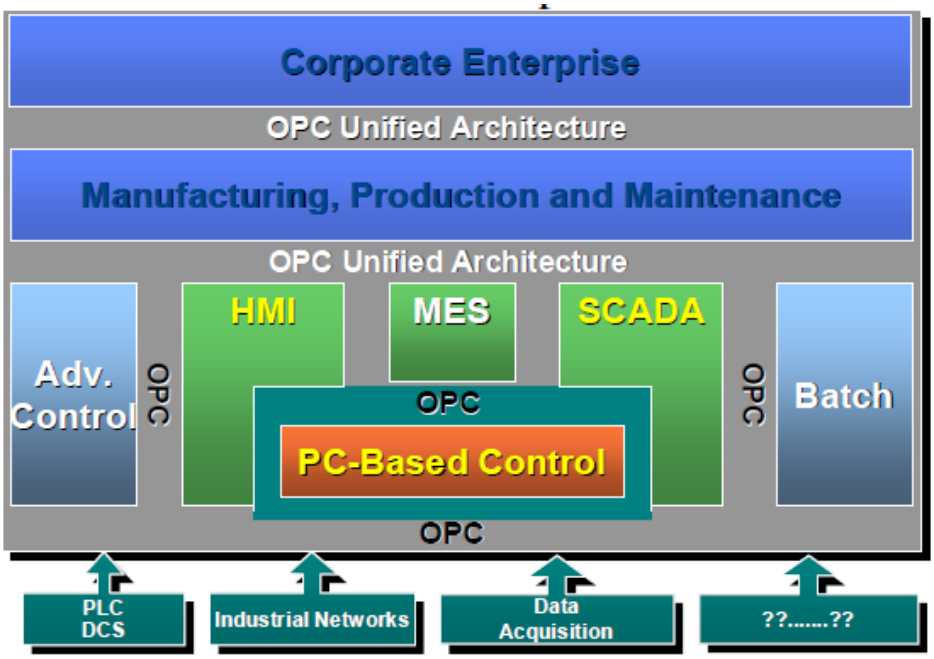


Figure 17: OPC-UA interaction in the automation levels [Burke 06]

OPC-UA consists of 13 specifications [OPC-UA 1, OPC-UA 2, OPC-UA 3, OPC-UA 4, OPC-UA 5, OPC-UA 6, OPC-UA 7, OPC-UA 8, OPC-UA 9, OPC-UA 10 and OPC-UA 11] that define a standardized set of services and policies that assemble this relatively new interoperability protocol. The specifications can be divided in three main sections: the core, the access type and the utility type. The core consists of the first 7 specifications that defines concepts, services, security model, information model, address space, profiles and, service mappings. The access type specification defines Data access, alarm and conditions, programs and historical access. The last two specifications parts of the utility type have not been released yet. The next subsection will deal with the overview of core specifications that are of interest in this work.

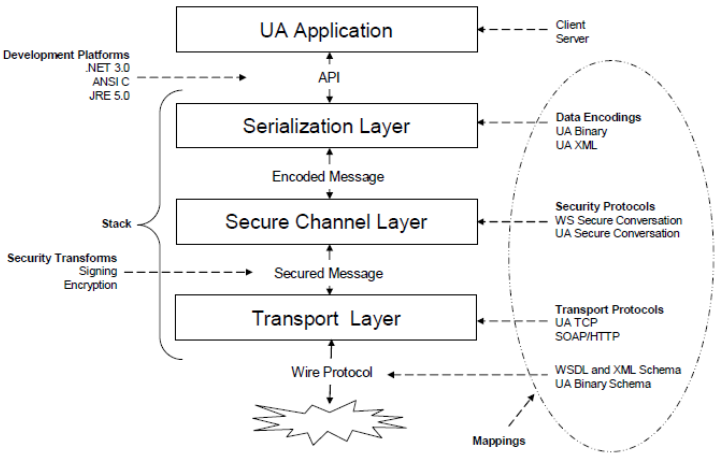


Figure 18: OPC-UA stack overview [OPC-UA 6]

The majority OPC-specifications have in general a technology free definition. However OPC-UA relies in several core technologies as depicted in Figure 18. UA binary and UA XML are the two encoding supported. Related to the encodings, the security, transport and contract protocols are also related to the encodings. It is noticeable that OPC-UA provides two technology options for implementation.

2.3.2.2 OPC-UA specifications

OPC-UA Address Space [OPC-UA 3]

The OPC-UA address space provides a standard structure for servers to expose data to clients. Figure 19 depicts the OPC-UA object model representation. Objects are used to structure data closely following object oriented design concepts. These can contain variables, methods and, other objects. Generally, objects are not used for containing process data; instead, variables are in charge of this. There are two kinds of variables, properties variables and data variables. The first one can be seen as the metadata (configuration parameters) while the other one represents the data of an object.

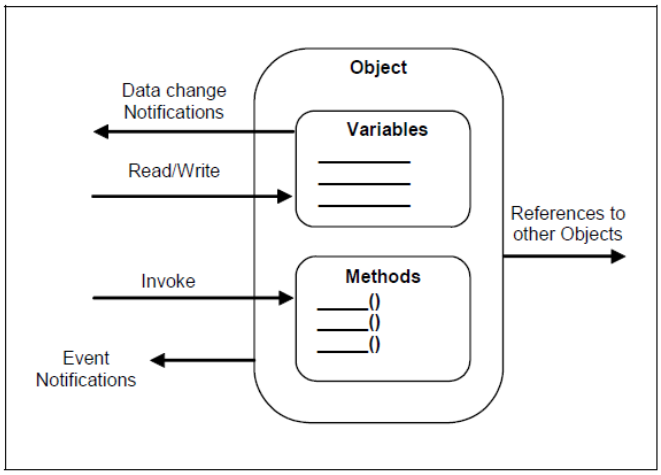


Figure 19: OPC-UA object model [OPC-UA 3]

The objects and components are represented in the address space as nodes and they are described by attributes and linked by references. The attributes may contain Id's, names or descriptions while the references could define a source node, a reference type or a target node.

Just as Object-oriented programming, the address space specification provides the TypeDefinitionNodes that defines instances of Objects and Variables. These can be seen as the schema or the original class for instantiating Variables and Objects. The overall functionality of the address space is provide a view of the nodes available in the server allowing the client to navigate its node elements using services defined in [OPC-UA 4].

OPC-UA Services [OPC-UA 4]

This specification defines the standard service sets that OPC-UA provides to clients for communication with the servers. Table 3 describes the core functionality of these sets.

Table 5: OPC-UA service sets [compiled from OPC-UA 4]

Service set	Description
Discovery service set	Allow a client to discover the endpoints implemented by a server and to read the security configuration for each of those endpoints
Secure channel service set	Allow a client to establish a communication channel to ensure the confidentiality and integrity of messages exchanged with the server
Session service set	Allow the client to authenticate the User it is acting on behalf of and to manage Sessions
Node management service set	Allow the client to add, modify and delete nodes in the address space
View service set	Allow clients to browse through the address space or subsets of the address space called views
Query service set	Allows clients to get a subset of data from the address space or the view
Attribute service set	Allow clients to read and write attributes of nodes, including their historical values. Also it allows clients to read and write the values of variables
Method service set	Allow clients to call methods. They may be called with method-specific input parameters and may return method-specific output parameters
MonitoredItem service set	Allow clients to create, modify, and delete MonitoredItems used to monitor attributes for value changes and objects for events
Subscription service set	Allow clients to create, modify and delete subscriptions. subscriptions send notifications generated by MonitoredItems to the client

From all the service sets, the relevant service sets that are of relevance to this work are the MonitoredItem and subscription service sets. These provide OPC-UA with eventing mechanism for monitoring of OPC-UA objects, variables or attributes. Subscriptions may contain a set of MonitoredItems defined by the client. A subscription will allow the client to monitor an item and receive notifications. Notifications can be subscribed for data value change, Events via EventNotifier

definition or for aggregated values calculated from a time interval [OPC-UA 4]. Monitored items contain settings like mode, sampling interval and notification queue size which sets the relationship and restrictions between client and the server for notification delivery. The procedure for notification delivery in OPC-UA is composed of: Publishing, Subscribing, and queuing.

The description of the notification process is as follows:

1) Client adds as MonitoredItem of certain node in the address space to a subscription
2) Publish requests will be sent during an open session on a UA server. These requests will not be bounded to a certain subscription.
3) Requests will be queued in the OPC-UA server session.
4) Later on publish responses will be sent back to the client.

The responses can include notification messages, a sequence number or a "heartbeat" message to keep the connection alive. Notification messages may include the properties of the message such as timestamps, values and name of the MonitoredItems. Figure 20 depicts the notification model of OPC-UA. This model has been criticized by some authors, claiming that this model is more synchronous than asynchronous as seen in [Colombo 10].

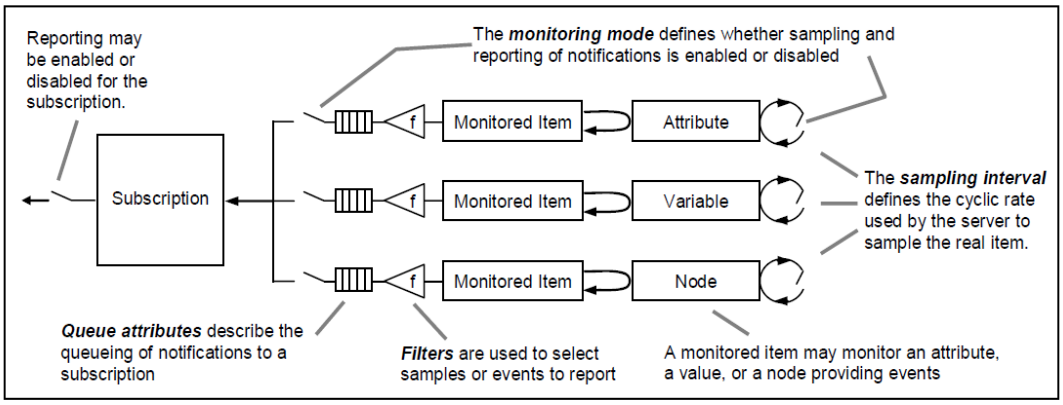


Figure 20: MonitoredItem Model [OPC-UA 4]

OPC-UA Information Model [OPC-UA 5]

The information model is one of the core specifications of OPC-UA. This specification is an extensible mechanism to model the server's information to expose the items, properties, metadata and diagnosis information as a hierarchical model in the address space. In other words, the information model defines the address space of an empty OPC-UA server. The basic information models can be extended allowing standards to be built on top; this is one of the core features of the specification [Virta et al. 2010].

The information in OPC-UA defines a standards information model for general use. This information can be retrieved by a client in order to navigate through its address space. The nodes consist of standardized: references types, dataTypes, object, objectTypes, variable and, variableTypes.

2.3.2.3 Industrial experience state-of-the-art

Combining CAEX with OPC-UA for production monitoring and control system support

As part of the transition of companies towards SOA, Schleipen [2008] demonstrates a framework using OPC-UA address space and Information modelling to include plant information using CAEX-based descriptions. Notifications arriving to the server triggered the addition of new nodes. After the nodes were generated, clients are notified of current changes in the address space for registration. OPC-UA allowed synchronizing the production processes by notifying the clients about new relevant information available. Hence OPC-UA can be used for support for monitoring and control processes.

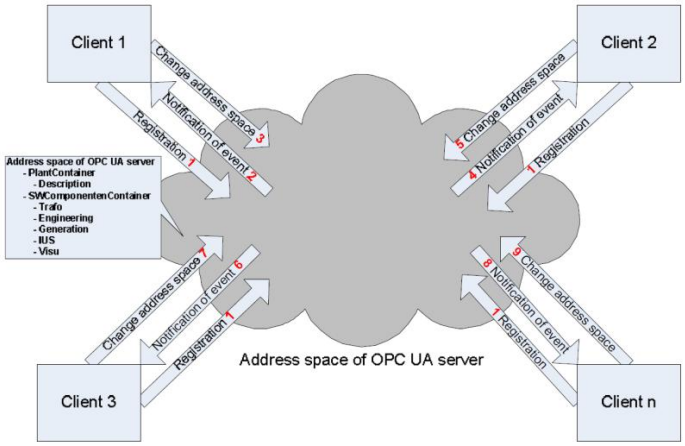


Figure 21: Interaction of clients with the address space [Schleipen 08]

Monitoring and control framework using OPC-UA

Van Tan et al. [2009] propose a framework for building monitoring and control in factory automation. He defines a set of components that would allow bridging existing software systems and OPC servers enabled devices. Using OPC-UA as communication technology, he emphasizes that this technology is a good choice for development of web-enabled industrial automation and manufacturing software systems. Figure 22 shows the proposed architecture. The overall implementation consists in an OPC-UA server acting as a gateway for integrating devices them into a web-based environment. In this case client monitors would require of an OPC-UA client to obtain information integrated by the server.

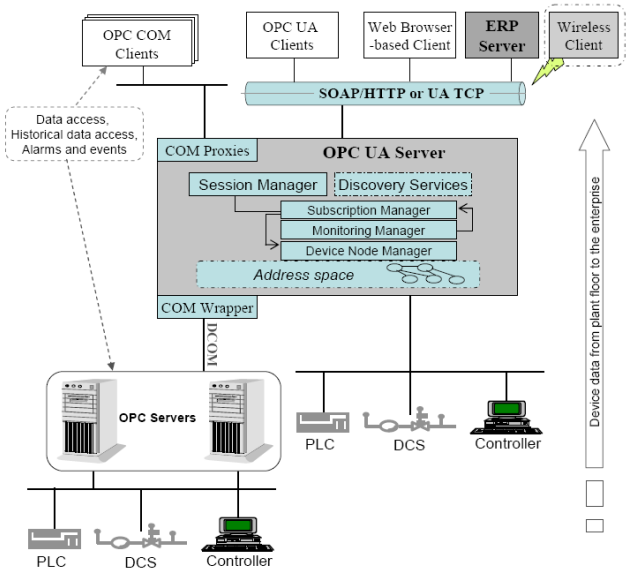


Figure 22: Architecture for monitoring and controlling of field devices [Van Tan et al. 09]

OPC-UA integration with ISA-88/95 for batch process management

Similar implementation by Virta et al. [2010] proposes integration for batch process management using OPC-UA and ISA-88/95. However he emphasizes that the information exchanged using the OPC UA can be configured to follow standard information models. However, due to the binary nature of OPC-UA, he proposes the use of a UA2XML adapter in order to connect to the business process engine which commonly uses XML messaging as shown in Figure 23.

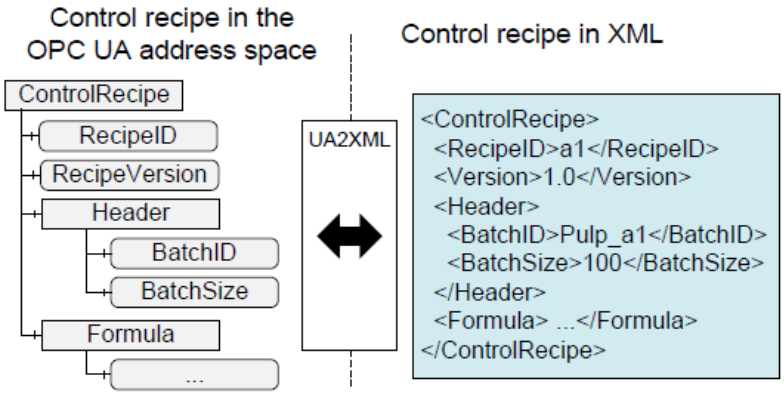


Figure 23: UA2XML conversion [Virta et al. 2010]

In other similar approach by Seilonen et al. [2011] propose the integration of an Enterprise Asset Management for condition-based monitoring. He also made use of the UA2XML adapter for the integration of the OPC-UA with the EAM system WS's. The results demonstrated that OPC-UA can be combined with existing service-oriented frameworks such as Windows Communication Foundation (WCF) in CBM applications through an adapter-based design.

2.3.3 Assessment on OPC-UA and DPWS

From all the technologies that implement SOA, Only two cope with the requirements and demands that an industrial environment requires for SOA implementations. As seen from table Table 4, technologies such as UPnP, JINi and WS are technologies that can apply SOA. However, they have been studied and discarded as technologies that could bring SOA to device levels.

On the other hand, DPWS and OPC-UA are suited best to lower down SOA into the device level. Even though, they have similar end-goals, they are yet different approaches for interoperability that would make these technologies difficult to compare. Despite the fact that both technologies claim to use Web Services, the differences must be addressed. OPC-UA intends to provide interoperability via gateways and address spaces. For instance, building an address space for certain protocol and navigate the values on the address space using already defined services does not allow the process management to create modular services for business processes. The services are not as descriptive as DPWS allows. For example the service set of navigating node would not be of any worth for a Business Process Management (BPM) application. Customized services on DPWS can have self descriptive services that could be joined together into a service composition for business process. As seen in [Virta et al. 10] adapters for OPC-UA are needed for connectivity with typical XML based MES systems. In summary, both technologies reach the interoperability end-goal; they both have drawbacks and advantages. Nonetheless, as highlighted by Colombo et al. [2010], interoperability among these two specifications can bring a better SOA implementation. Thus neither of these technologies can be discarded.

Table 6: Contrast between OPC-UA and DPWS characteristics

Criteria	OPC-UA	DPWS
Implementation level	Device level / although profiling is possible to achieve sensor level	Sensor level / using enhancements and profiles
Plug-n-play solution	Yes	Yes
Dynamic discovery	Nor available still	Available
Services definition	Standardized	Customizable
Business process management compatibility	Currently based on adapters [Virta et al. 2010]	BPEL compatible
Transport	UA XML (not available yet) /	XML/SOAP / XML-binary

technologies	UA binary	under research
Eventing mechanism	Publish / Acknowledge	Publish/subscribe
Information modelling	Very extensible	Limited
Mappings	WSDL / XML-Schema UA-schema	WSDL
Integration approach	Via gateways and clients	DPWS clients, plain web service calls
Security	Provides authentication/sessions and encryption	Can be implemented using WS-Security

2.4 Rule engines

Rule engines are part of Expert Systems (ES) which are within the Artificial Intelligence (AI) area, a computational system that intends to emulate the decision-making ability of an expert human being. Reasoning and evaluation of the knowledge within this system, can lead to conclusions that may predict desirable or undesirable situations inside a system. ES may be based upon rules or knowledge base (KB) as key part of the inference capabilities of the engine.

Three types of inference methods can be commonly found among the research community. Substantial work has been done towards making hybrid implementations of these methods and noticeable enhancements can be provided by integrating different inference approaches as described in implementations such as [Xiuqin et. al 09, Weidong & Warren 96]. Nonetheless hybrid approaches exist; inference methods should be distinguished and categorized as shown in Figure *24*, due to the different nature among the engines.

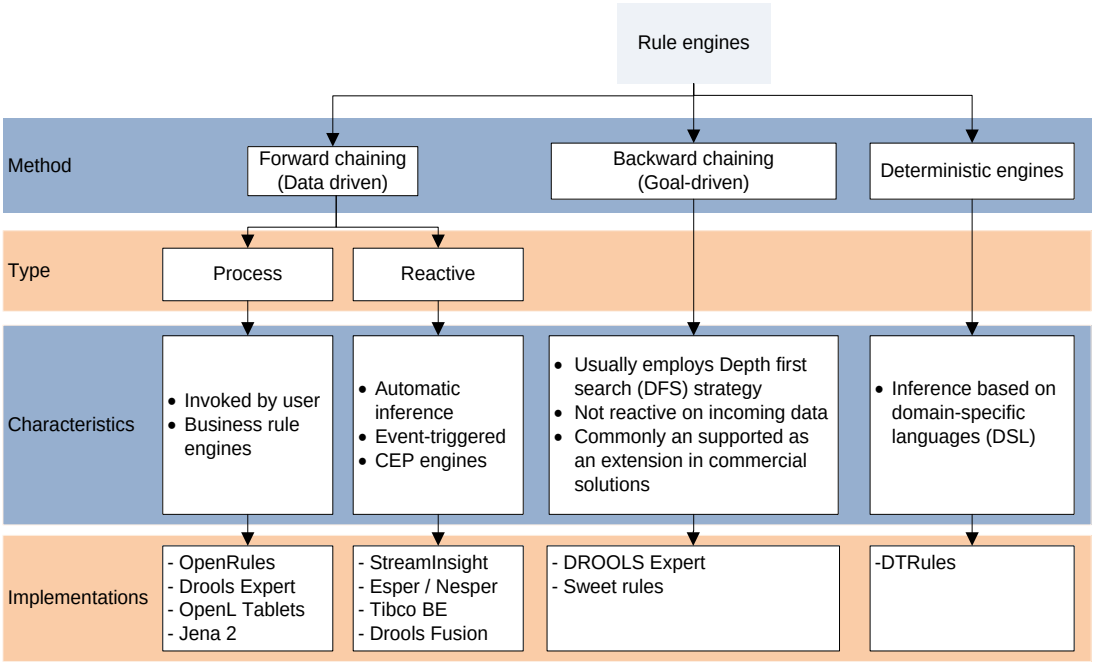


Figure 24: Categorization of rule engines

Deterministic inference engines

These are most commonly relying in Domain Specific Languages for inference using a custom algorithm. [Carvalho & Simoes 11] for example describes the use of a deterministic engine based in OML (Open Modeling Language) for the manipulation of ontologies while [Qichao et. al] introduces an improved metadata model inference engine using Model Representation Language (MRL) as part of an extension for MARS system that relies in a metadata configurable modeling tool which cannot only infer single-tier domains but also multi-tier ones. Usually these type of inference engines are easier to implement and maintain than the ones based

on forward or backward chaining, however the due to its constrained implementation domain, its popularity is not as prominent as the other approaches.

Backward chaining inference engines

Also known as goal-driven inference engines, these reasoners iterates within its own KB based on the data that is available. Differently from forward chaining, the reasoned is triggered by a goal; it uses the available data once is triggered to search for this end-goal. The most common algorithm for this implementations is the Depth-first Search (DFS) which transverses a hierarchical graph of nodes in a uniform manner [Cormen et. al 01]. Figure 7 depicts the control flow where can be seen the goal as the trigger of the whole inference procedure.

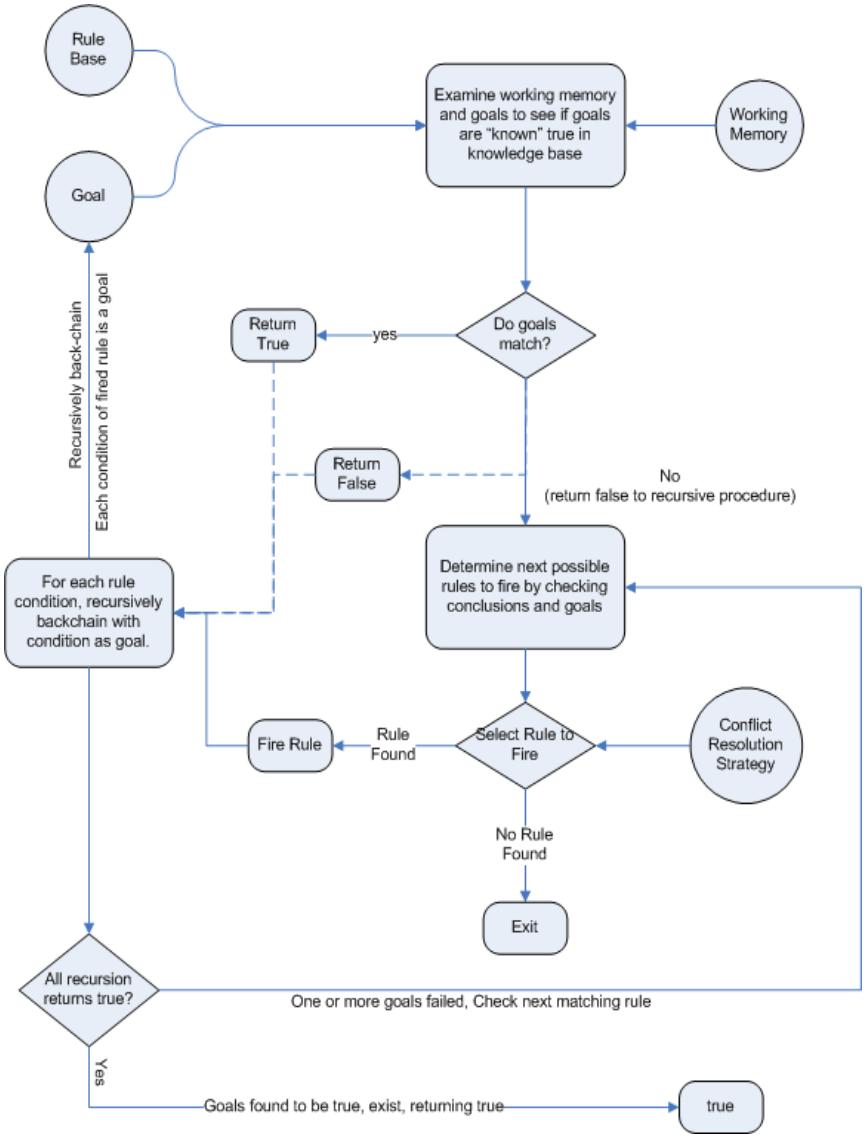


Figure 25: Backward chaining control flow [JBossCom 11]

Forward chaining inference engines

In a certain extent, this is considered to be the opposite method of the backward chaining. This method is also known as data-driven inference engines, meaning that based on an event/data arrival; the system reacts by checking along its rule base to fire outputs. Figure 26 depicts its control flow which, differently from backward chaining, it exits the execution when no rule can be fired.

This method has two different approaches:

Process inference engines: A Non-automatic inference engine invoked by the user. Commonly known as business rule engines, they look for the current data inputs and compares for rule triggering.

Reactive engines: Commonly known as Complex Event Processors. This is an event-triggered engine that provides automatic (reactive) inference.

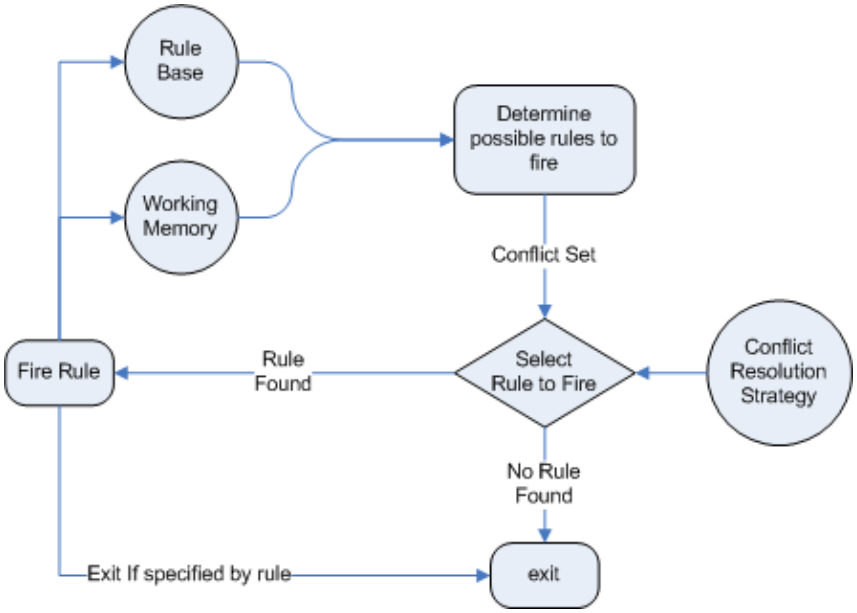


Figure 26: Forward chaining control flow [JBossCom 11]

Reasoners in general react to either data and/or goals; however reactivity of the reasoner is crucial for the implementation in an EDA environment. Manufacturing systems are mostly modeled and characterized as discrete event systems. Their behaviors are driven only by instances of different types of events [Cassandras & Lafortune 99]. Due to this a system that requires to be invoked in order to obtain a value or certain output cannot be efficiently used in EDA's. Table 7 defines the compliance of an inference method by its reactivity level.

Table 7: Contrast of inference engines

Inference method	EDA compliance	Reactivity	Algorithms
Forward chaining/ Process inference	No / Invoking is required (Client-server paradigm)	Stateless	• RETE • Linear • Treat • LEAPS
Forward chaining/ Reactive engines	Yes / Event triggered	Stateful	• RETE • Linear • Treat • LEAPS
Backward chaining	No / Invoking is required (Client-server paradigm)	Stateless	• DFS • RETE
Deterministic	Yes / Publish – subscribe paradigm supported	Stateless / Stateful	• Custom

2.4.1 Complex Event Processing

2.4.1.1 Overview

Complex Event Processing (CEP) is a technology which defines a set of tools and techniques for analyzing and processing the complex series of related events that drive modern distributed information systems [Luckham 02]. This technology proposes a reactive alternative to process information in an ES to quickly identify and solve problems. At the same time, it effectively utilizes events for enhanced operation, performance, and security. CEP is applied to a broad spectrum of information system challenges, including business process automation, schedule and control processes, network monitoring, performance prediction, active monitoring and, intrusion detection.

CEP solutions are strongly linked to fast moving data streams and event clouds. They leverage the information to achieve operational insight in the areas of Business Intelligence (BI), security, monitoring of Systems and, networks. These processors handle events in real-time, seeking out the patterns and relationships within the data that have a meaning to the organization. They can identify important complex events, event patterns and situations that notify new opportunities, critical threats, changing conditions, or other material factors that will impact the organization. CEP solutions can also offer organizations increased capacity for competitive action and improve their level of security.

According to the book of David Luckham [Luckham 02], “The power of events”, the CEP processes are classified in two main categories:

- Processes that react to events by executing activities: Related to what it can be called business logic of the CEP engines. The collection of static queries running in the core of the application receive the events transforming the data based on rules specified by an Event Processing Language (EPL).
- Connectors that transport events between activity processes: These types of processes are related to the system integration perspective. Input and output adapters are needed to convey the abstracted event to a system of interest.

The process architecture of a CEP defines the components that together configure the two previous types of processes described previously. The architecture is composed of four main components:

- A diagram that shows the processes in the system and their connections
- The flow of events along connectors
- Behavior specification that consists of rules which specify the process behavior to events.
- Design constraint that specifies the limitations on process behaviors.

Complex events are generated by abstracting events called low-level events, which are the events that do not have any level of abstraction, also known as “atomic events”. The level of abstraction of a complex event depends on the correlation between the low-level events and the iteration to higher layers as studied by [Jobst & Preissler 06]. Figure 27 shows the relationship of low-level events with complex events. All the connections shown between low-level events represent the correlation rules specified in the EPL of the CEP.

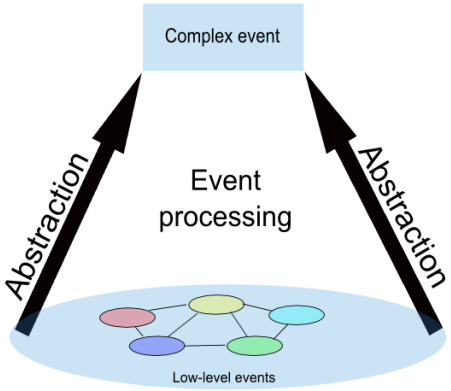


Figure 27: Event abstraction [Adapted from Luckham 02]

This technology relies on different techniques, such as, event abstraction, pattern detection, event hierarchies, event relationships and event-driven processes. From these techniques, causal, temporal and spatial properties of the event cloud can be analyzed. As exemplified by Figure 28, simple events with different characteristics

can be abstracted into a high-level complex event using pattern detection and event relationships.

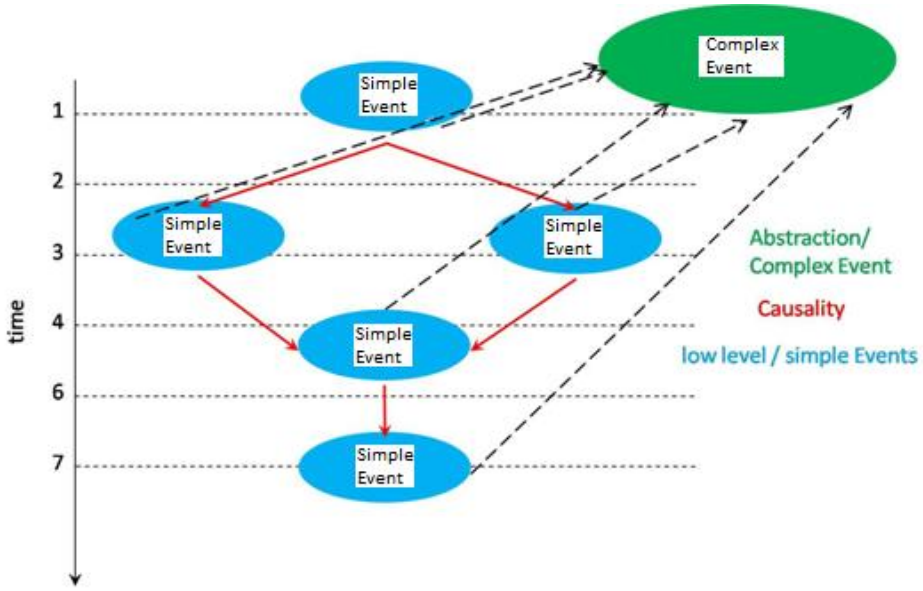


Figure 28: Event pattern matching [Adapted from Luckham 05]

2.4.1.2 Classification

Due the origin of CEP from the Aerospace and Defense industry, Eftimov [2006] highlights the use of JDL Data fusion functional model to define the levels of CEP may be appropriate. The JDL model specifies 4 levels depending on the increasing complexity of data processing and inference of high-level information. Level 1 taking care of event refinement and adaptation. Level 2 joins spatial and temporal relationships between groups of events to infer abstract patterns. Level 3 performs the impact assessment based on predictive analysis of the results from level 2. Level 4 performs ongoing monitoring for process refinement that could be translated into a feedback loop used in automatic control.

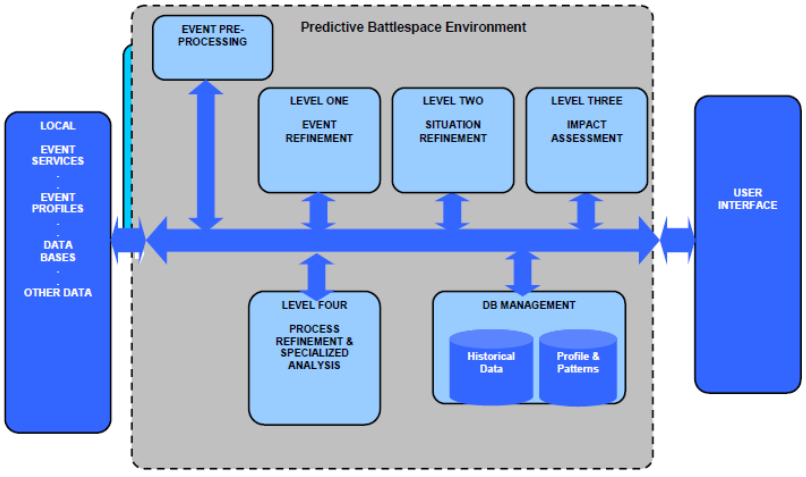


Figure 29: JDL Data Fusion Model [Tibco 07]

Currently commercial products do not provide more functionality than the first two levels of JDL model. Substantial research is being done using Bayesian inference engines to achieve more predictability that could leverage CEP functionality to higher levels [Tibco 07].

Table 8: Available CEP solutions (See Appendix A)

Vendor	Inference type	JDL model level	Licencing	Support	Development
Progress Apama	Rule based	1,2	Commercial	Good	Studio
StreamBase	Rule based	1,2	Commercial	Good	Studio
Coral 8	Rule based	1,2	Commercial	Good	Java API
Esper and Nesper	Rule based	1,2	Open Source	Supported	API C#/Java
Inference Machines	Bayesian Inference	1,2,3	Commercial	Supported	Not found
TIBCO BusinessEvents	Rule based	1,2	Commercial	Good	Studio
MS Streaminsight	Rule based	1,2	Commercial	Good	API C#
Rapide	Rule based	1,2	Open Source	No support	API Java
RuleCore	Rule based	1	Free	No support	API Java

2.4.1.3 CEP architecture

The architecture of complex event processing engines do not differ one to another. The same five components can be found by any open or commercial tool available in the market. Based on a list of the shown in Table 8 it is possible to identify these five components:

Input adapters: Provide connectivity to external sources and streams of data routing it towards the processing engine.

Processing engine: Holding static queries which are used for rule matching.

Database connectivity: Provides pull mechanism for passive databases or data warehouses.

Configuration front-end: Configures the processing engine with queries, registries patterns and actions using an application development tool most of the times provided by the vendor.

Output adapters: Route information to external systems

The architecture resembles a bottleneck funnel, where the events from several input adapters are guided towards an engine and then redistributed to interested applications. Even though the word bottleneck is a curse word in production engineering, CEP takes this concept to other level by deliberately funneling events to generate compressed chunks of more relevant information without hindering the monitoring process performance by excelling processing efficiency and high throughput. Figure 30 depicts an example of a commercial CEP architecture.

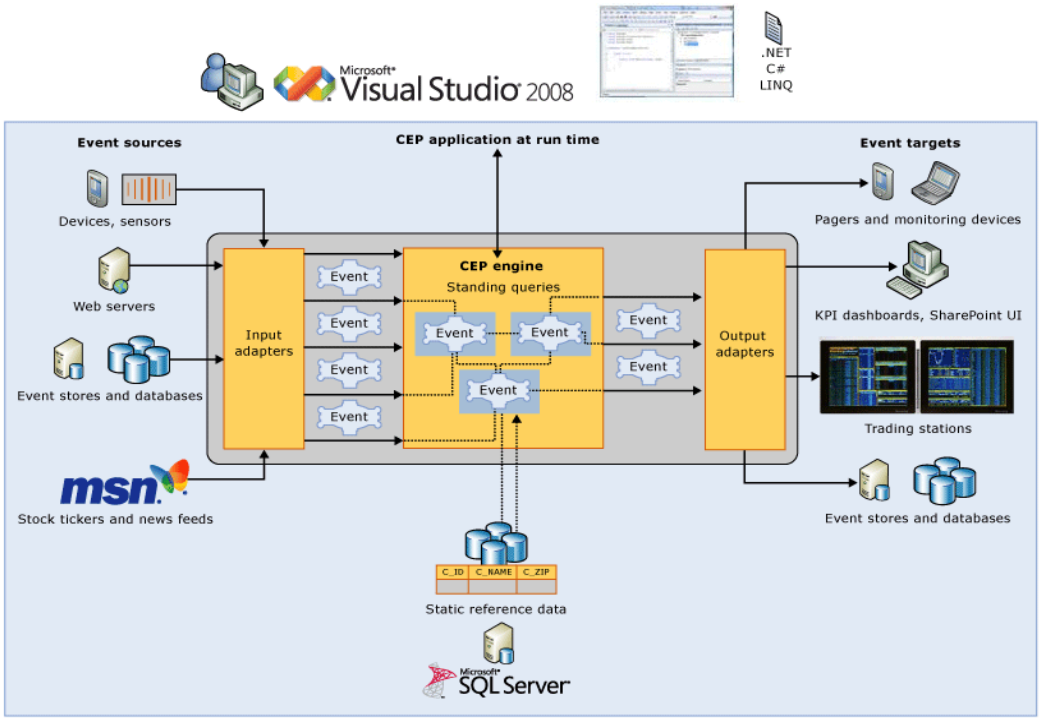


Figure 30: Microsoft Stream Insight CEP architecture [Microsoft 11]

2.4.1.4 Event processing languages

Event correlation using CEP is tightly linked to event clouds and streams. In order to aggregate this flow of events, static queries have to be defined. These definitions are based through event processing languages that describe the rules and data relevant to the processor. CEP solutions have non-standardized event processing languages; due to this three different languages will be discussed in this analysis. Generally, these languages represent base for other languages, meaning that any other existing languages may be based in these or on the same concepts.

RAPIDE-EPL

The RAPIDE [Luckham 95] event pattern language or RAPIDE-EPL is a declarative computer language for writing patterns of events. The patterns can specify sets of events together with their parameters, timestamps, and causal dependencies, and which events are causally independent of each other.

RAPIDE-EPL consists of mathematical expressions that describe patterns. It does not include any algorithmic programming features like assignment or conditional branches. It is as simple a language as can meet the basic requirements for CEP. It has the ability to declare event types, and then match on them extracting information from the event object. It contains mathematical semantics, temporal operators and strong typing to avoid common errors in writing patterns.

```
EVERY StockTickEvent(symbol = "IBM", price > 80) WHERE timer: within(60 seconds)    .... (1)
A -> (B or C)                                                                       .... (2)
```

Program 1: EPL examples

Program 1 shows EPL queries that define the continuous query with a temporal statement. The first statement will notice about every stock event that has as parameters IBM and a prices larger than 80 in a time window of 60 seconds. The second example exemplifies a causal pattern statement where even A needs to happen first and then event B or C in order to set true the statement and send a CEP event. Combination of causal, spatial and temporal statements is possible and it could be as complex as needed to correlate events.

SASE+

Starting as a proposal from [Gyllstrom et al. 2008], SASE+ is a pioneering language created in the University of Massachusetts. It has a high-level structure similar to SQL for ease of use. The design of the language, however, is centered on temporal event patterns that have not been sufficiently addressed in relational data processing. The language features: event sequencing, negation, Kleene closure, parameterized predicates and sliding window. SASE+ has one of the simplest language structures. Table 9 explains the meaning of each statement, the structure of the query look like follows:

Table 9: Description of SASE+ statements

Statement	Description
[FROM < input stream >]	Provides the name of an input stream.
PATTERN < pattern structure >	Specifies the structure of a pattern to be matched against the input stream.
[WHERE < pattern matching condition >]	If present, imposes value-based constraints on the events addressed by the pattern.
[WITHIN < sliding window >]	Further specifies a sliding window over the entire pattern.
[HAVING] < pattern filtering condition >	Further filters each pattern match by applying predicates on the constituent events.
[RETURN < output specification >]	Transforms each pattern match into a result event for output

CAYUGA EVENT LANGUAGE (CEL)

Based on the documentation made by [Brenna et al. 07], CAYUGA is an expressive Complex Event Processing (CEP) system developed at the Cornell Database Group. Cayuga uses non-finite automatons with buffers to match event patterns to queries. The language is derived from "event algebra" and it is similar to SQL and SASE.

Table 10: CAYUGA query structure

Statement	Description
SELECT <attributes>	Provides the name of an input stream.
FROM <algebra expression>	Specifies the calculations to be done against the input stream.
PUBLISH <output stream>	Names the output stream (for layered processing or subscription).

2.4.1.5 CEP in manufacturing state-of-the-art

Complex processors have been contemplated in manufacturing systems for several years. Luckham [1998] in his early work presents a case study of a silicon chip fabrication line connected to a TIBCO Rendezvous Information bus. In his work, he demonstrates that causal relationships of events between different levels of abstraction are possible by using a complex event processor. He specifies a method for abstraction hierarchies to define views in different levels of a distributed event-based system. This methodology utilizes event pattern mappings which led to the low level causes of errors in such systems. Subsequent studies on the field, introduced the concept of Event Processing Agents (EPA) for data mining in automation systems across the network [Perrochon et al. 1999]. Distribution of the processing resources and parallelism using EPA's allows a flexible and dynamic run-time configuration of the processing framework as well as performance for high-throughput event processors.

Several reports sharing the experiences of CEP in industrial environments are available, highlighting benefits and opportunities of CEP [Magid et al. 10, Vidackovic et al. 10, Kellner & Fiege 09]. Summarizing approaches and methodologies, the most obvious advantages encountered are the abstraction of information allowing users to define rules dynamically instead of the IT experts. Extends the expressiveness of the system by detecting complex patterns and also makes it flexible by externalizing rules, avoiding hard-coded rules. One of the most important advantages included, is the relationship of data freshness and data value where CEP stands out from passive systems due to its immediate response (Figure 31).

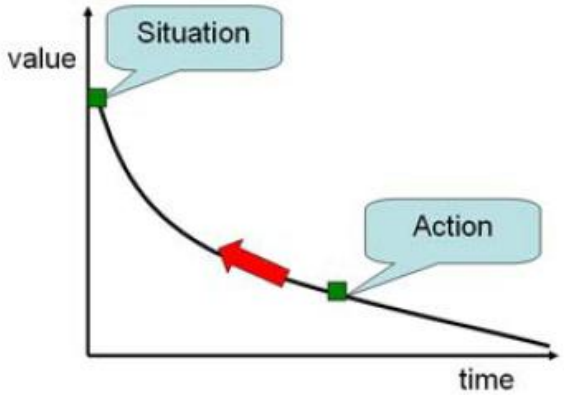


Figure 31: Data freshness to business value [Vidackovic et al. 10]

The contrast between CEP and BAM is the level where the data is reflected in a monitor. On one hand, BAM focus displaying the current state of a business, while on the other hand CEP focus in the state of events over time. This abstraction level difference makes the definition of KPI's to be declared from two different viewpoints: Goal-Rule or Rule-Goal. Kellner & Fiege [2009] presents a methodology based on the Zackman framework [Zackman 87] for building a monitoring model from top-down perspective. The methodology proposed starts from a Business Motivation Model (BMM) towards the specification of rules. Doing so, the development process would be guided into situations with more impact and higher return for CEP.

Plenty of work has been done within the manufactory domain in the area of CEP and RFID, but few in ubiquitous manufacturing. Rosales et al. [2010] presents architecture for RFID and Sensor Networks (SN) for process integration and management. The approach mapped the CEP engine acting as a service connected to a broker inside an integration bus. His implementation considers the benefits of CEP and an event middleware for factory-wide connectivity breaking up the layers between automation levels and enabling events cross unaware of its source location.

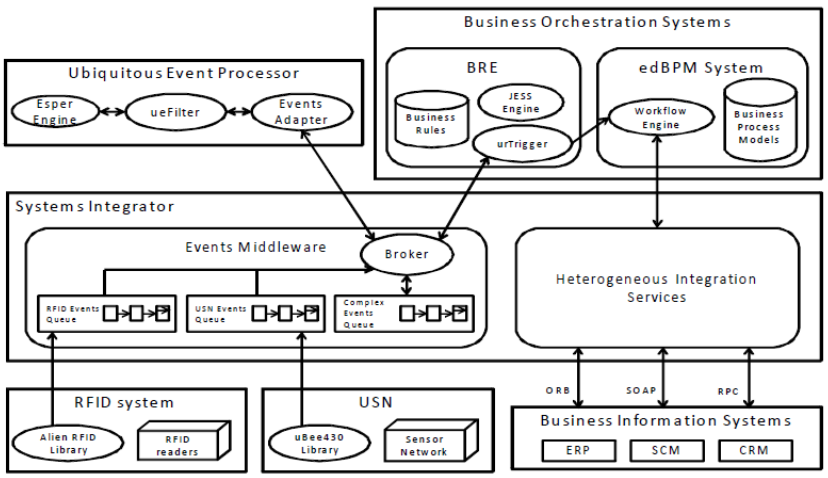


Figure 32: edUFlow system architecture [Rosales et al. 10]

2.4.2 Other event processing technologies

2.4.2.1 Event stream processing

This technology rises from the need to do real-time data analysis when in the early 90's no database was capable of such characteristic [Luckham 06]. The main difference with CEP and ESP comes in the conceptual capabilities of each technology. The cloud corresponds to CEP, meaning that events do not need to be in order to be processed, while streams to ESP may need to come in an organized manner in order to detect trends and patterns on streams. ESP is in general a subset of a CEP. In more detail, Event stream processing is focused more on high-speed querying of data in streams of events by applying mathematical algorithms to the event data. Some applications were directed to stock-market feeds in financial systems and algorithmic trading. On the other hand, CEP is focused more on extracting information from clouds of events created in enterprise IT and business systems. CEP includes event data analysis, but places emphasis on patterns of events, and abstracting and simplifying information in the patterns. The goal is mainly to support as much as possible the area of enterprise management decision making. The first commercial applications with CEP were in the Business Activity Monitoring, for example monitoring conformance to service level agreements.

2.4.2.2 Simple event processing

Simple event processing (SEP) consists in the analysis of atomic events for further actions. It does not consider the abstraction or patterning of events but mostly the action that an event may cause as reaction. Example of this technology is the generation of an event based on the temperature of a sensor. The change in temperature may trigger an event that while processed can be turned into an alarm.

2.4.3 Summary of rule engines

Semantic reasoners also known as rule engines come in different types depending on their inference method. Three basic types of inference methods exist. The most relevant to manufacturing of production processes leans towards the reactive forward-chaining inference engines. The concept "Data freshness" shows the important of a system to react early and on time for business processes to be efficient. Non-reactive (goal-driven) inference methods are prone to process information out of the data freshness zone due to its passive nature. Moreover, reactive processors can be strategically modeled in a business-oriented (goal-oriented) fashion as seen in [Kellner & Fiege 09]. Currently big IT players are betting on CEP as technology for Business rule implementation which makes this technology a target for implementations on future enterprise applications. Other technologies such as ESP and SEP are basic subsets of CEP and due to this those technologies alone cannot offer the full potential of a reactive rule engine as CEP. This leaves CEP as the technology to follow in the implementation of this work.

3. Methodology approach

The selection and design processes of the manufacturing monitor will be elaborated within this section. Architecture paradigms, technologies will be selected based on the outcome of the technology review presented in the previous section.

3.1 Technology mapping and tool selection

As seen from the technology review, many techniques for monitoring have been implemented and analyzed through the research community. Figure 33 shows a summary of the techniques and modes identified in the technology review. Finding the right technique was done based on criteria extracted from the review and by directly applying it to the system under observation. Based on the outcome of the review presented in chapter 2, the criteria detected for the selection of the right approach is the following:

- Performance: Will the monitor operations hinder the SUO performance? Do the operations need to be performed externally from the controlled SUO?
- Data freshness: Do the SUO need the monitoring results on runtime?
- Fault-tolerance: How critical is the SUO dependency on the monitor?
- State-of-the-art: What are the tendencies for future monitors?
- Scalability: Ease to integrate to other systems
- Dynamism of the system: Will the system requires re-configurability? Does the system change its physical configuration?

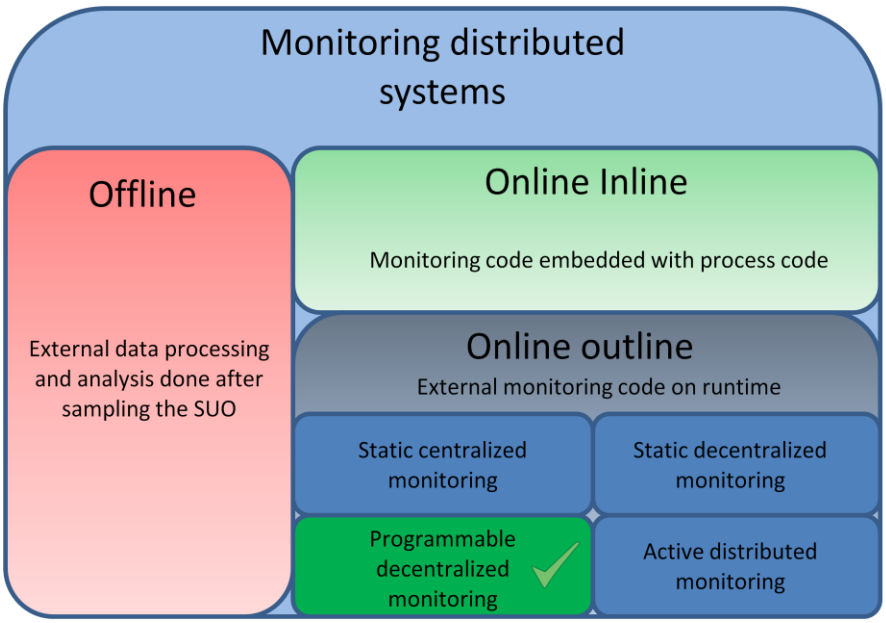


Figure 33: Monitor techniques and modes map

A set of assumptions were made in order to select a monitoring method. Information output incoming from the monitor should be as fresh as possible. Failure of assembly lines or bottlenecks can be costly and fast reaction toward failures is required immediately. For this reason, offline method is not suitable. Adding monitoring code and data aggregation on controllers it's a possibility, however resource constrained devices might not possess the required processing power and the system visibility to abstract information into a higher-level event. For this reason online inline monitoring would not satisfy completely the goals of the monitor implementation, still online inline could provide pre-processing to reduce data transfer and increase abstraction of messages injected to the monitors processing unit. Moreover, the monitor must be robust enough to actively report alarms and events. At the same time it should be flexible enough to adapt other elements to its framework and follow the current trends for fault detection and data processing. Therefore, static centralized and decentralized techniques do not entirely comply with those requirements; this reduces the options to only the programmable and active monitoring techniques. Finally, considering the static nature and the granularity level of the data in the system, it can be concluded that Programmable Decentralized Monitoring is the technique that best matches the criteria to the SUO proposed for this work (see section 4.2.1). Active distributed monitoring would be an overstated implementation and its main feature of re-configurability would not be practical in this case.

Using a decentralized monitoring technique leads us into the analysis of technologies and paradigms of distributed systems. As the technique states, mobile code should be able to re-configure distributed monitors on setup from a monitor manager. In order to propose the environment for communication, standardized communication technologies in the manufacturing domain were analyzed concluding in an interoperability implementation of OPC-UA and DPWS. Both technologies are middleware for system integration across several layers of automation. Furthermore, their specifications implement eventing to provide an asynchronous paradigm that leverages SOA towards SOA 2.0. The inclusion of publish/subscribe paradigm benefits the monitor performance by acting on system happenings instead than on user demand for information (client-server behavior).

Monitoring implementations involve data processing and mining form large data warehouses in order to aggregate and detect patterns that trigger actions or alarms. Data freshness is a concept that introduces the relevance of information of a monitor against time. By actively processing information arriving to the monitor, relevance of the information generated can be kept high and system reactivity improves. Current approaches to process and analyze data consist in the use of rule engines. As result from the state-of-the-art, it can be concluded that the only inference method that comply with the data freshness concept is the reactive forward-chaining inference engines. Other inference methods consists on goal driven and user invoke

methods that are prone to unnecessary data processing. Besides, this other methods propose a polling mechanism for data processing which can cause delays and additional processing.

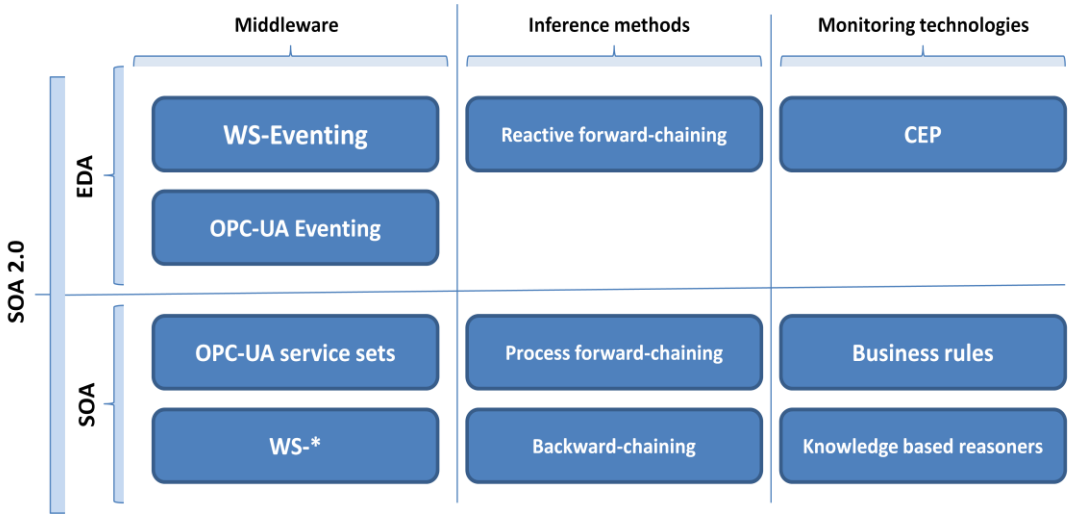


Figure 34: state-of-the-art Technology map

Figure 34 shows the summary of the technology review presented. It maps the technologies to their respective inference method and architectural paradigm. Based on the previous assumptions, it can be concluded that CEP tackles the most of the requirements that are commonly found in monitors for manufacturing systems such as responsiveness, heterogeneous scalability, fault-tolerance, data freshness and, performance. Given to the vertical integration provided by the middleware technologies presented, it can be possible to extend the visibility of Business Activity monitors which one of the driving technologies consist of CEP.

As seen from Table 8 several platforms for CEP are available and most of the commercial solutions offer similar benefits and characteristics, however, source code is not available in commercial solutions as well as expensive licensing is required. Open source solutions such as Esper provide similar characteristics whilst making available full documentation and support. In addition, Esper also provides a .NET implementation called NEsper giving this framework advantage from other commercial solutions such as MS Stream Insight which has only .NET implementation. Moreover, NEsper offer to the data processing community the capability to embed CEP engines into applications. Due to the aforementioned reasons, the tool selected for implementation is NEsper CEP. NEsper is targeted for EDA architectures which allow triggering custom made actions following an Event Condition Action (ECA) paradigm.

OPC-UA bases its architecture on gateway servers and clients that can navigate the servers address space via web methods. The platform independency advantage of OPC-UA allows any third party client implementations to be compatible with any server. However, due to the membership issues with the OPC foundation, OPC-UA

.NET v1.01 was the only option available for embedding a client into a processing engine. This pushed the whole implementation towards a .NET environment. Connectivity between legacy PLC systems and OPC-UA should be done by OPC-UA adapters contained within an OPC-Server. For our experimental implementation, Ignition OPC gateway provides the functionality needed for the test bed connectivity. It packs web based OPC-UA functionality with sample modules that allows connectivity to Modbus, DeviceNet and other field buses. Furthermore, it uses binary-TCP encoding that gives high performance for data transmission.

The tool and stack selection for this work is summarized on Table 11. The .NET framework provides a DPWS stack. However, it turned out that the stack is being contained on the MS .Net Micro Framework which made it incompatible with the other stacks available for OPC-UA and NEsper since they need .NET framework 3.0. For that reason an in-house DPWS stack was developed in order to subscribe support WS-Eventing in the implementation. This would avoid having an ad-hoc solution for overall integration.

Table 11: Selected tools for development

Name	Functionality
OPC-UA .NET v1.01	Connecting to OPC-UA servers for event and notification subscriptions
NEsper CEP/ESP	Data aggregation and event processing platform.
DPWS .NET (In house)	WS-Eventing compliable stack able to discover and subscribe for relevant events.
Ignition OPC-UA server	Act as gateway for legacy PLC systems. Map devices on a navigable address space
Windows SQL server 2005/2008	Event logging for historical analysis

3.2 CEP Monitor functional architecture

The architecture design intends to take advantage from the benefits of CEP. The overall design consists of a funnel like structure where the events are filtered to deploy actions and messages that extract information out of data. Following an Object Oriented approach, the functionality of the monitor such as configuration, external connectivity and processing was located within function blocks. The blocks are clustered together with internal interfaces. Each component provides interfaces for integration among other function blocks in the architecture. This approach allows distributability of the system by defining interfaces between blocks.

The monitor architecture structure was designed based upon a combined analysis of technologies, tools and techniques identified on the technology selection and the requirements of the implementation use case. Each tool provides specific connectors that allow the technologies to match into a coherent architecture. Founded on the native composition of the CEP engines, the central component in this architecture is the event processor. Considering the basic components of the CEP architecture, the CEP should contain input adapters in order to route information towards the processing engine.

Available ESB solutions provide functionality such as translation, routing, and integration of heterogeneous systems. NEsper allows integration to enterprise service buses via JMS and JMX. However, DPWS and OPC-UA already provide a standard communication middleware making the use of the service bus redundant in a factory-shop floor environment. Nonetheless, an input event manager component is needed in order to manage subscriptions, event translation and, event registration to the CEP engine. To ensure runtime flexibility for rule definition, the event manager and processing engine requires of a configuration component to set up configuration of the rules, event registration, event transformation, output configuration and routing.

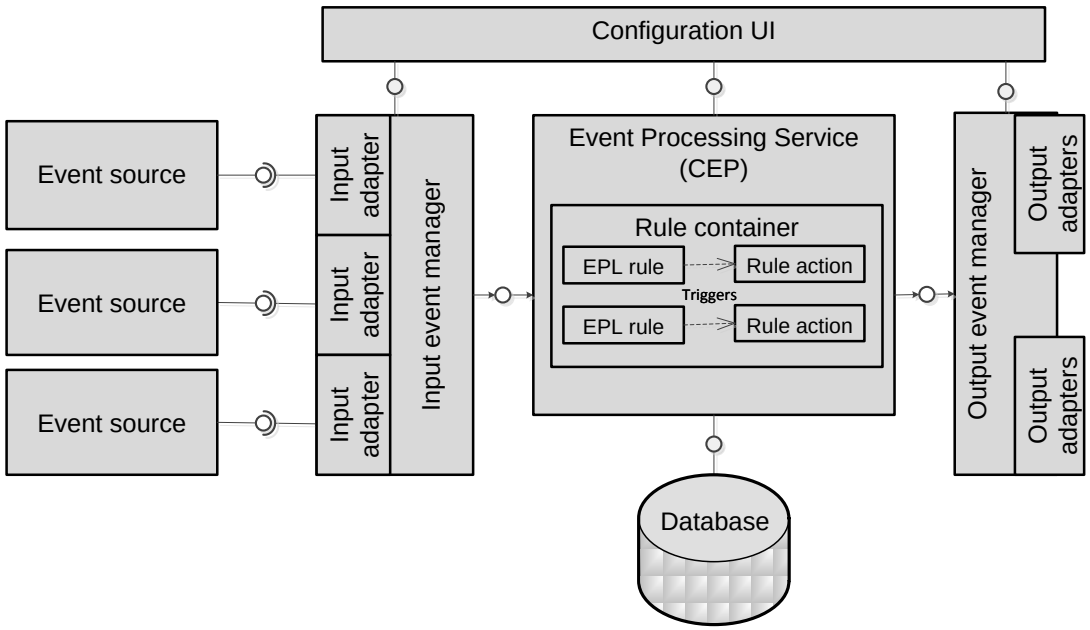


Figure 35: CEP Monitor functional architecture

Closely following the CEP architecture, four main function blocks were defined in order to provide accessibility to heterogeneous events. From Figure 35, starting from left to right, the first module identified is the input event manager. Its functionality consists in the interaction with the CEP to transform events into an understandable format for the processing engine. This module configuration has to navigate and register the events available from data sources. Event registration is crucial for the CEP in order to identify the type of event is handling. Events with same body name in the message would be treated the same even though the event

source is different. Hence, this module gives a unique name to events for preventing "event confusion". The next component is the event processing service. This component receives events from the input manager and matches them into rules. Once a rule is matched, a rule action is triggered. The rule action may calculate, transform, route or generate complex events for output adapters. For historical event analysis and event logging, the processor provides database connectivity. This also allows transforming the passive nature of the databases into active event based behavior. The output adapter manager takes events and data generated by the event processor and routes it into a specific output adapter. The output adapters can be proprietary connections, interoperable Web Service or even an OPC-UA server. Finally a UI configuration module is required in order to govern the properties of each other module allowing flexible configuration of rules and actions. Table 12 describes in detail each of the components found in the functional architecture.

Table 12: Functional description of the architecture

Component	Functionality description
Event source	• Provide an eventing mechanism for data exchange
Input event manager	• Subscribe for events on behalf of the input adapter • Interface with the Configuration UI • Manage input adapter configuration
Input adapter	• Receive information from event sources • Transform event into internal data format • Route event into the CEP engine
Rule container	• Create a pool of rules for multiple rule matching
EPL rule	• Define the business logic of the system via rules • Pattern detection and data aggregation
Rule action	• Define extra business logic for post-processing
Configuration UI	• Provide UI for event subscription • Provide UI for EPL rule definition • Provide UI for rule action scripting • Provide UI for CEP initialization and configuration
Processing engine	• Host the rule container for rule management • Receive events and match it against rules
Output adapter manager	• Receive complex events from the rule actions • Manage output adapters for external connectivity
Output adapter	• Interface with external systems

4. Implementation

This chapter describes in detail the technical architectural and experimental implementations of this work. Each of the components designed for the implementation of the CEP-based monitor.

4.1 Monitor implementation

This section describes the tool implementation characteristics and functionality as well as the relation between components and its technical structure.

4.1.1 Event manager implementation

As previously mentioned, an ESB's could be used to route and transform events. Nevertheless, OPC-UA and DPWS already gives a middleware for system integration in a factory floor domain. For that reason, an event manager was implemented giving the functionality of a micro ESB to homogenize events from OPC-UA and DPWS.

NEsper CEP allows processing of POJO, MAP and XML input events. With that in mind, SOAP messages incoming from DPWS-enabled devices are fully compatible and they can be processed directly without any event transformation. In a similar way OPC-UA XML encoding also is compatible with the processor. However, two main issues are involved in this approach. One is that OPC-UA binary would not be supported since the response is not natively in XML. Thus, wrapping this information in a semantic language is required. And the most significant issue is the "event confusion". The CEP engine gets confused by having XML events without a unique main XML element. Considering SOAP to have an envelope as main element, the CEP engine would not be able to recognize events from different sources. For that reason, the event manager should wrap the events into a uniquely named XML element in order to avoid this situation. Each input adapter wraps the events on arrival. The following subsections will describe in detail the technical architecture of each adapter.

NEsper requires event schemas for event registration. The GUI developed for the event manager, allows the monitor administrator to subscribe to events from the input adapters and automatically register the events. The input manager generates unique event names by combining endpoint information with the event name. For instance, considering the case that a DPWS-device contains the address 192.168.2.100 and that it exposes an event called *ItemTransferOut*. The name of the event would get internally transformed into *ItemTransferOut2100* before registration and submission to the CEP engine, this considering that every endpoint does not contain an event of a similar name. This practice solves the issue of "event confusion" and allows the registration of multiple events.

Figure 36 shows a detailed concept map diagram of the event manager functionality. The event manager provides on set up the registration information to the CEP engine whilst routing the wrapped events on runtime from the event sink to the CEP engine. The generation of CEP events follows a similar transformation than the schemas. Using the metadata of the incoming notification is possible to build the corresponding event that would match the schema's main element.

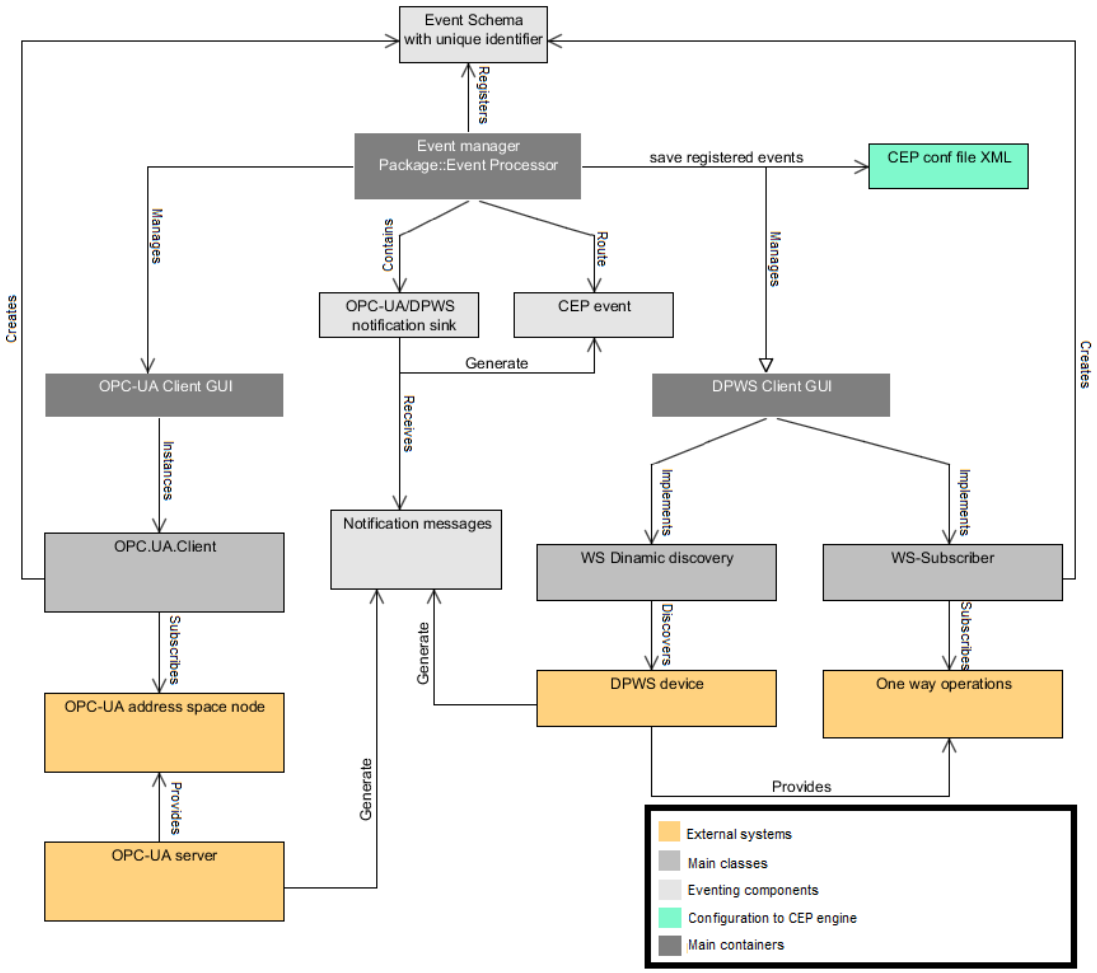


Figure 36: Event manager concept map

4.1.1.1 OPC-UA input adapter

OPC-UA manages subscriptions via standardized service sets. Therefore, an OPC-Client capable of navigating thought the address space was implemented for accessing to the desired nodes that contains the data of interest. OPC-UA V1.01 .NET stack serializes the notifications and makes them available with the *notification* and *monitoredItem* objects in the *Quickstart* client implementation. The event can be converted into a CEP event type by wrapping the message contained in these objects with XML. Figure 37 shows an example of the attributed contained in an address space node of OPC-UA. The wrapping of the events should be done with the least information as possible taking in consideration that not all the information of the notification is useful for the monitoring application.

In the device level, the OPC-UA notifications are triggered on value change. This translates in notifications containing a publish timestamp, a value, and several other attributes containing session information and subscription. The solution of this work implements a static wrapping; the OPC-UA input adapter only wraps the timestamp, ID and value in order to mimic the elements contained in the *ItemTransferOut* event defined in CAMX specifications. A more dynamic and automatic event wrapping would be needed in case of selecting different nodes on the address space with different attributes. This could be possible by reading the information model of the OPC-Server; however it is out of the scope of this work.

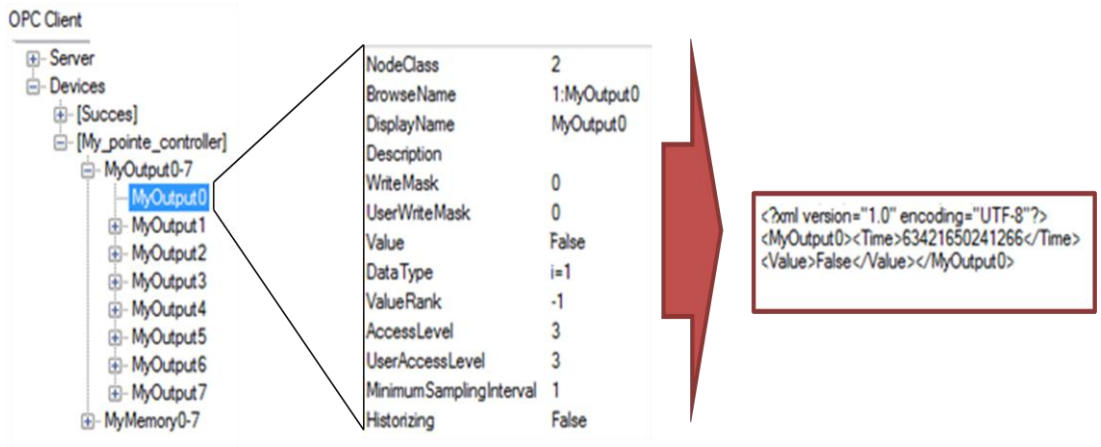


Figure 37: OPC-UA to XML wrapping

4.1.1.2 DPWS input adapter

DPWS enables devices to expose operations that can be subscribed to. The DPWS input adapter possess a custom Configuration UI that allows WS-Dynamic Discovery, WS-MetadataExchange and WS-addressing, The UI allows to display the operations described in the WSDL files and subscribe to the desired events. On event subscription, the event manager is notified to register the newly subscribed event. On notification arrival, the input adapter transforms the event from SOAP to plain XML to forward it to the CEP engine.

Differently from OPC-UA, DPWS notifications come in SOAP format. However, as mentioned previously, this format brings problems with the CEP engine. To overcome this, the message is parsed using XPATH. After that, the elements are wrapped with a unique identifier based on the main element of the event and the event metadata such as the event endpoint source. Figure 38 depicts the transformation made in the DPWS input adapter once the message arrives to the event sink.

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-
envelope"> <soap:Header>
</soap:Header>
<soap:Body>
<ItemTransferOut
Time="2000-02-02T10:47:12.00-05:00"
itemInstanceId="001"
laneId="1" />
</soap:Body>
</soap:Envelope>
```

```
<ItemTransferOut265>
   <Time>2000-02-02T10:47:12.00-05:00</Time>
   <itemInstanceId>001</itemInstanceId>
   <laneId>1</laneId>
</ItemTransferOut265>
```

Figure 38: SOAP message and internal CEP Event

4.1.1.3 Processing engine implementation

The NEsper CEP engine is the foundation of the event-based manufacturing system monitor. This specific CEP provides routing, processing and post-processing capabilities for incoming data based on rules and rule actions. These rules can be defined via UI that sets up all the components needed for initializing the processing unit. However, before start-up, the CEP engine goes through a process of configuration. The configuration consists on several steps that are mostly made through the configuration UI: EPL statement definition, validation of EPL statements with the event registry, definition of the actions triggered by EPL statements and finally initialization the engine. Explained as a sequence, first the user defines EPL statements and actions in the configuration UI, after that, the CEP engine then loads the registered events from the configuration file generated by the event manager. Afterwards, the EPL statements are validated with the schemas to detect elements that may not be included in the event. Initialization of the CEP is concluded if the validation returns success. The CEP engine cannot initialize if any rule fails to validate. This ensures that the rules are matching the expected events.

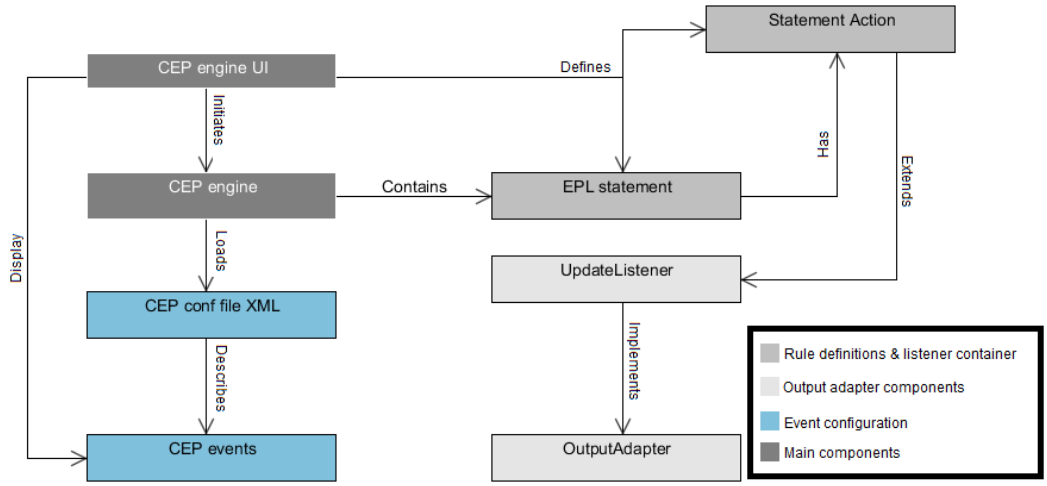


Figure 39: Concept map for CEP configuration and rule composition

The statement actions extend the *UpdateListener* class of NEsper API which updates every time the related rule is matched. The action definitions are in charge of

routing and post-processing as well as implementing output adapters. Results of the EPL statements are placed in EventBeans that can be accessed via getter methods in the *UpdateListeners* Figure 39 shows a concept map for configuration and rule definition.

The processing engine also provides connectivity to databases. This is done via EPL statements. However, database connection and drivers should be handled during the configuration phase. EPL just applies queries to databases. For more information of EPL statements see Appendix B.

4.1.2 Output adapter implementation

There are three methods how the WS Output adapter may operate. The first method consists in describing all of complex events in a WSDL. This requires of dynamically building this contract on CEP configuration. The second requires configuring a client that invokes a service in the consumer. The third method requires of a generic set of operations described in a WSDL that provides to the customer information on CEP available complex events. Differently from the first one, this requires of consumer invocation. This last approach defies the concept of EDA.

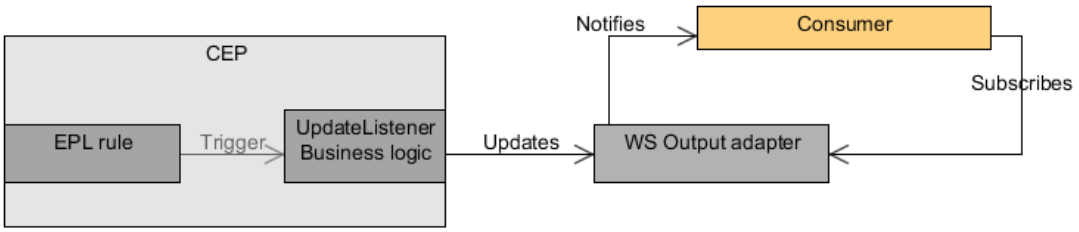


Figure 40: Output adapter description

The output adapters are directly associated to the actions registered in an EPL statement. Once a rule is matched, the action associated with the rule will generate a complex event. The complex event has to be available for other systems to consume. For this reason and considering the interoperability that Web Service technologies provide, A WS output adapter allows consumers to subscribe to the complex events generated. The implementation made use of .NET WCF *WebMethods* for generating an endpoint connection. Complex events generated within the *UpdateListener* inside the action were exposed as one-way operations.

4.1.3 Configuration model description

The configuration is divided in two main sections: source configuration and CEP configuration. During source configuration, the UI triggers the input adapters for discovery and connection to data sources. It also displays the available messages in order to select and subscribe. The DPWS input adapter is configured to load the WSDL file, parse the service description and notify to the DPWS input adapter UI about operations available for each source. The OPC-UA UI displays the address space of the server. It allows subscription to nodes using the *monitoredItem* and *Subscription* service sets. The CEP configuration consists in the setup of statements, actions and registration of the actions to the specific EPL statement. The actionListener is configured using C# plain classes. An on-the-fly compiler was developed using the *Reflection* libraries of .NET in order to use C# as scripting code. This provided the flexibility to load actions in runtime without compiling each time an action is changed. A more detailed configuration process is depicted in Figure 41.

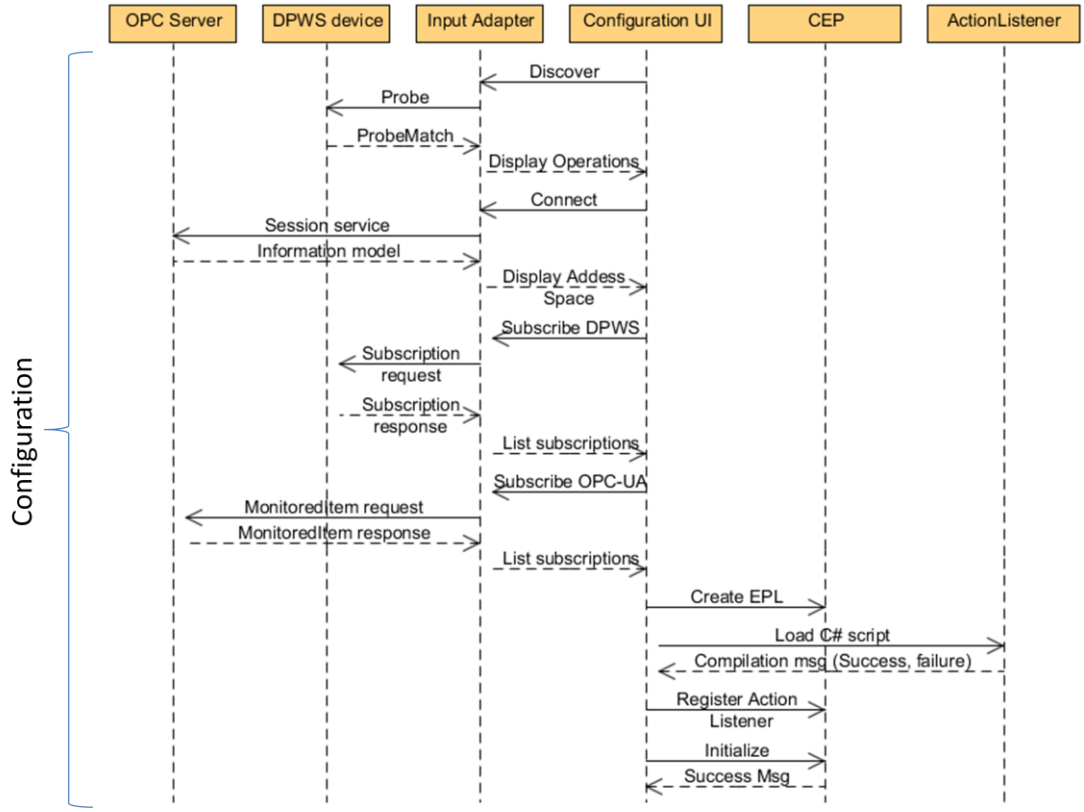


Figure 41: Deployment and configuration model

4.1.4 Runtime technical description

Runtime functionality becomes simple once the system is configured. Having extense configuration instead of complex runtime functionality is a tradeoff that benefits the process monitoring and increases its reliability in the end.

During runtime the monitor receives notification from heterogeneous event sources. The event-based monitor input adapters act as a router while transforming

notifications for the CEP. All notifications going through the CEP are match against rules. The business logic contained in the Action Listener is triggered when the rule's pattern is matched. Finally the listener directs the complex event into an output adapter for further distribution of complex events into visual components of a consumer. In cases when the complex events have to be fed back into the processor, there are two approaches. The first consist in an EPL statement containing the INSERT INTO clause that internally routes the event results into another EPL rule. The second approach is to feed back the event using the *SendEvent* operation of the CEP engine factory within the business logic of the ActionListener.

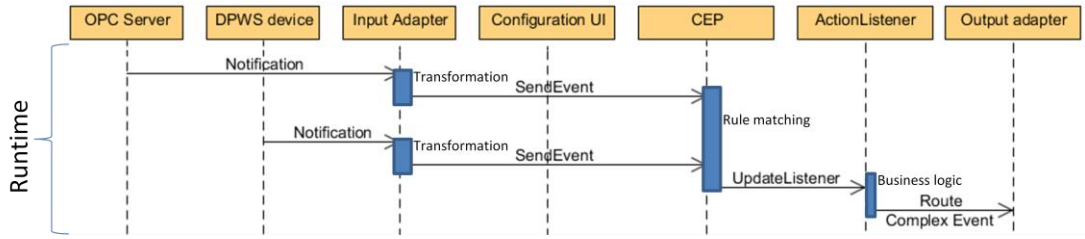


Figure 42: Platform functionality

4.2 Experimental implementation

The monitoring application was tested using a Flexlink Dynamic Assembly System (DAS). DAS products consist of modular factory solutions for assembly, inspection, testing and repair of products. The system main feature is the modularity of its standard components that can interconnect with each other for reconfiguration and scalability to meet production demands. Some of the modules available consist of robot cell, workstations, conveyors, buffers and lifters. Figure 43 shows some of the solutions provided by Flexlink. DAS 30 middle segment was the only module used during this experimental study.

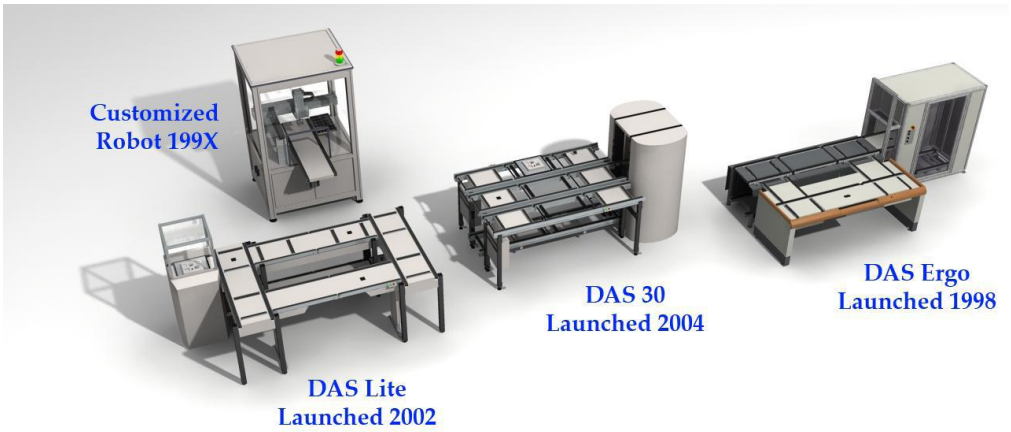


Figure 43: Flexlink products

4.2.1 Test bed

The module used for the test scenario consists of a DAS 30 middle segment. This particular module consists of two workstations, a main line, a middle lifter and a long conveyor used for coupling with other DAS products. Each workstation consists of three conveyors from which two are cross-conveyors. This later type of conveyors allows the pallets to turn 90 degrees for routing. The main line segment also consists of two cross-conveyors and a middle single direction conveyor. The middle lifter can transport the pallet to 4 different heights from which other modules can be interconnected. Each conveyor segment is being controlled and monitored by an INICO S1000 which is a DPWS enable device.

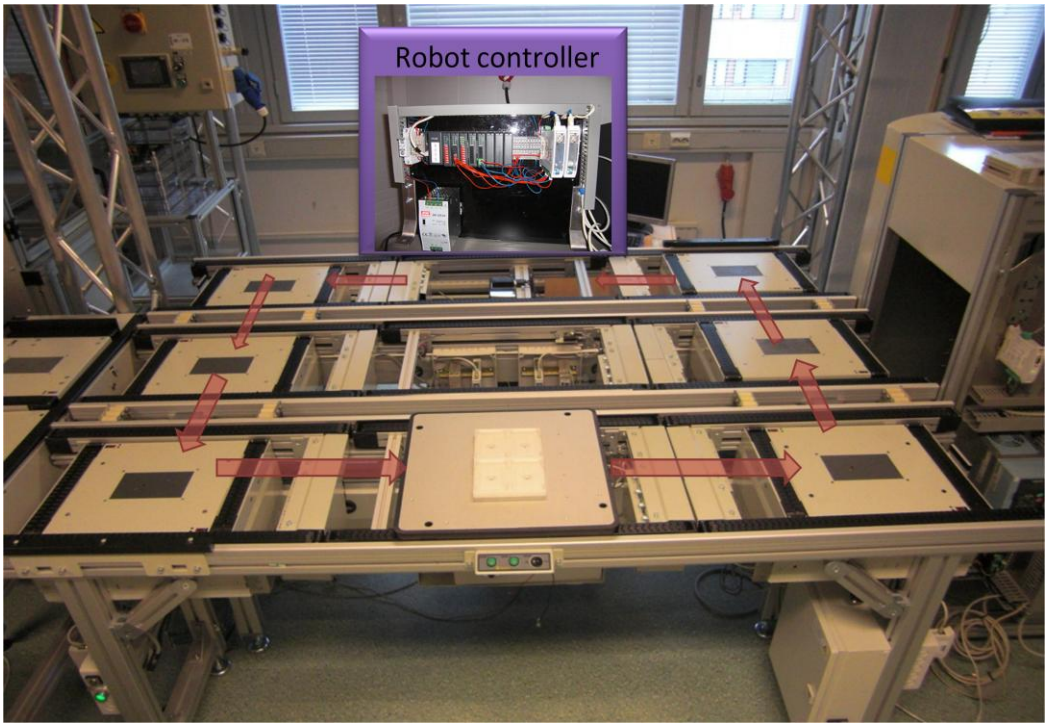


Figure 44: Test bed configuration

A robot is located at one of the workstations. The robot is controlled with a Nematron PCT-5800 controller. This particular kind of controller connects to other components via MODBUS-over-Ethernet. However, an Ignition OPC-UA server was used to access information instead of creating single input adapters for each communication protocol. OPC-UA server solutions already provide modules for legacy system connectivity, for that reason creating input adapters for the CEP-based monitor would be unnecessary.

Each conveyor segment generates transport events. The transport events consist of *ItemTransferOut* event as described by the CAMX (IPC-2541). Event messages contain information about the *dateTime*, *laneID* and *itemInstanceID*. However, since in this case this system will be monitoring the same product in the same line, these last two values will be fixed. Differently from DPWS, the OPC-UA server exposes

in the address space the variables and outputs of the PTC-5800. A specific output of the controller describes the process status of the robot. The notifications generated by the OPC-Server contain a timestamp and a value that describes process start and process end. Figure 45 describes the test bed scenario in detail.

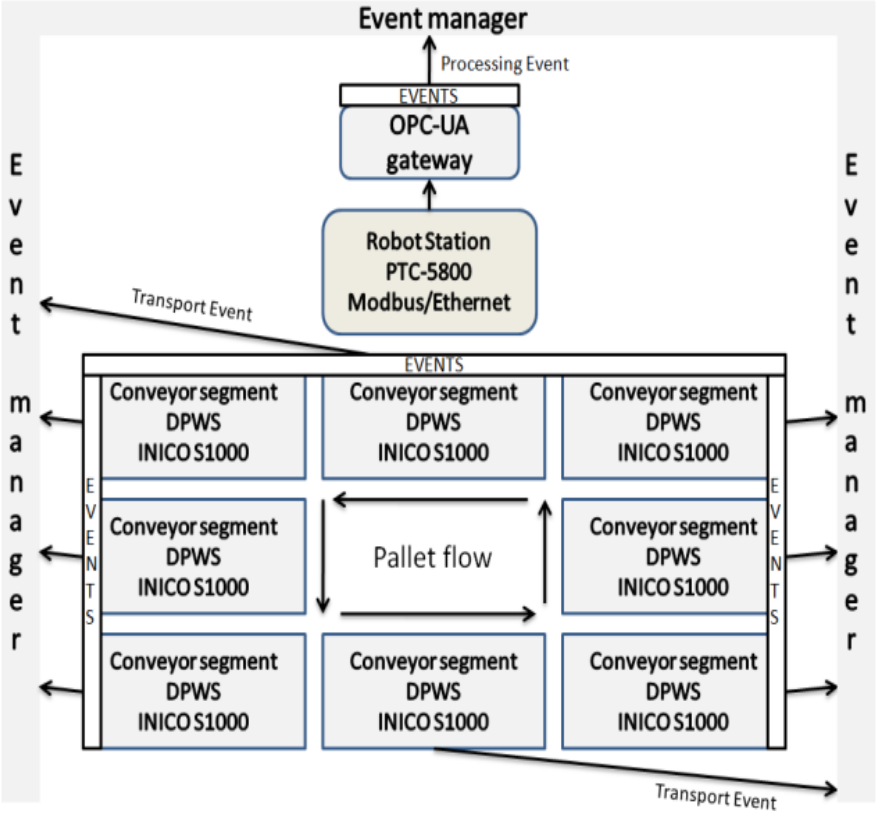


Figure 45: Test bed description [Garcia 11]

4.2.2 Use case definition

Conveyor system testing and system performance analysis requires of precise monitoring systems for an accurate detection of faults and productivity rates respectively. The use case for this study involves monitoring of a DAS 30 based manufacturing line for electronic assembly. During installation of DAS 30 systems there are commonly physical miss alignments between conveyor systems. This affects to efficiency of a pallet to transfer from one conveyor segment to another generating unwanted delays. Furthermore, electronic assembly operations are only a single task within the whole production chain, for that reason, productivity rates must be generated on a monitor for system performance notification to other operations of the production chain as well as for operator personnel.

Endurance tests are commonly performed to detect flaws in the system, as well to verify the consistency in the process performance and system installation success. In this experimental study the system under observation is processing constantly a product in a loop sequence as shown in Figure 45. The system is orchestrated using a Business Process Execution Language (BPEL) script. Events generated must be

received by the monitor and processed to notify performance indicators to the clients of the monitor.

The monitor implemented must analyze the pallet transfer timings to calculate the lap times based on single events generated by the production line. Moreover, the monitor has to notify whenever a pallet transfer takes more time than the expected. For this, complex events containing lap times should be generated as well as complex events with average lap times. Finally flaws must be detected and notified.

4.2.3 Tests performed

Four main tests were prepared to fulfill the requirements of the use case defined in this chapter and to evaluate the monitor capabilities for data aggregation and processing:

Monitor testing

This test validates the usability of the monitor for the use case. Subscription and connectivity is checked as well as the behavior of the monitor with sample events. An INICO S1000 was configured to submit notifications with different rates by changing the analog input of the device. Overall outcome from these analyses is the definition of the capabilities and pitfalls of the monitor.

Lap time calculation

This test proves the use of the monitor for detection of patterns and aggregation of data on a manufacturing environment. The generated events incoming from each conveyor segment are processed using EPL for generation of complex events containing the aggregated timestamp values of each *TransferOut* event in the system. The information contained in the complex event must inform the user about the processing lap times.

The test bed described previously defines a system which natively does not possess any message that provides the overall lap time of each production lap. For that reason, the monitor should aggregate the events that can be aggregated to obtain the data needed for this calculation. In this case the EPL shown in Program 2 was configured to detect the *lapTime* complex event.

The EPL selects all of the *TransferOut* events of the conveyors and robot station. On runtime the events are matched against a pattern that causally detects when a full circle is complete. The pattern consists of causal operators (->) that detects whether the sequence of events has happened. On detection, an action containing getter methods is triggered. The business logic of the action takes the times filtered from each event and adds it up. The outcome is later posted as a notification in the CEP configuration UI to show the calculation.

```
INSERT INTO
  lapTime
SELECT
  A.TimeStamp as T1,
  B.TimeStamp as T2,
  C.TimeStamp as T3,
  D.TimeStamp as T4,
  E.TimeStamp as TR,
  F.TimeStamp as T5,
  G.TimeStamp as T6,
  H.TimeStamp as T7,
  I.TimeStamp as T8
FROM PATTERN [every
 (A = TransferOut1Evt263 ->
  B = TransferOut1Evt247 ->
  C = TransferOut1Evt237 ->
  D = TransferOut1Evt244 ->
  E = RobotStationTransferOut ->
  F = TransferOut1Evt228 ->
  G = TransferOut1Evt222 ->
  H = TransferOut1Evt230 ->
  I = TransferOut1Evt240)]
WHERE E.Value = “True”
```

Program 2: EPL statement for lap detection

Average Lap time calculation

This test consists in the recursion of the *lapTime* complex event. The output generated has to aggregate values and notify subscribed systems about the average lap time of the last 5 laps. This test proves the recursive features of the monitor for higher level data abstraction.

Feeding back the events from one statement to another is done by adding the clause *INSERT INTO* on the EPL statements. When doing this, a second rule can be defined for evaluating the average time every five instances of the complex event generated in the previous test. Recursiveness of the CEP engine allows several rules to be fed back internally into other rules without having to code input and output adapters. In this case, *lapTime* complex event becomes a *lapTimeAvg* complex event. Program 3 shows the EPL configured to detect five instances of the *lapTime* complex event.

```
SELECT *,
  Count (*) as myCount
FROM
  lapTime.win:length_batch(5)
HAVING
  Count (*) = 5
```

Program 3: EPL statement for detection of five *lapTime* complex events

The first clause defines a counter as well as selects all the information inserted by the *lapTime* event. The second clause creates a length batch that allows only 5 events to be cached by the processor. This is important in order to clean the *UpdateListener* for new calculations. Finally the *HAVING* clause triggers the action by expecting the counter to be equal to five.

Flaw detection

This test proves the capabilities of the monitor to use its post-processing capabilities to notify about flaws in the system. The circulating pallet generates *TransferOut* events on each conveyor segment. The calculation of the difference of the *TransferOut* timestamp of segment *A* against the *TransferOut* timestamp of conveyor *B* can result in detection of conveyor misalignments that prevent the pallet to flow smoothly. This is done after the lap complex event is generated, getting the filtered results of the pattern matching and calculating time differences between each conveyor segment interaction.

For this test, the script shown in Program 4 was registered to the EPL rule defined by Program 2. This algorithm is executed every time the rule is matched. This script calculates the transitions of each conveyor interaction. A threshold was defined to detect if any transition takes more than 8 seconds to transfer. If the transition is higher than this threshold a notification will be published.

```
// Get all times from the EPL rule results
Time[1] = (Double)eventBean.Get(“T1”);
Time[2] = (Double)eventBean.Get(“T2”);
Time[3] = (Double)eventBean.Get(“T3”);
Time[4] = (Double)eventBean.Get(“T4”);
Time[5] = (Double)eventBean.Get(“TR”);
Time[6] = (Double)eventBean.Get(“T5”);
Time[7] = (Double)eventBean.Get(“T6”);
Time[8] = (Double)eventBean.Get(“T7”);
Time[9] = (Double)eventBean.Get(“T8”);

Double threshold = 8;
Double previousValue = 0;
Double transition = 0;
foreach(double T in Time)
{
transition = T – previousValue;
If (transition > threshold)
{
   deployAlarm(“conveyorFlaw”);
}
   previousValue = T;
}
```

Program 4: *UpdateListener* script for detection of transition flaws

5. Results

This chapter presents the experimental results from the tests specified during chapter 4 as well as a discussion of the conceptual results reached within this study.

5.1 Experimental results

5.1.1 Overall monitor functionality and limitation test

During this test, the system was set up for receiving a constant amount of events from external sources at different rates. Performance, configuration and overall limitations of the system were analyzed in this section.

The CEP-based monitor provides several UI that allows configuration of the several function blocks. Subscription to events can be successfully done using the Event Manager UI which allows navigating and subscribing to heterogeneous event (see appendix C for more information on monitor initialization via UI). Receiving events from different sources shows that a transformation layer is needed after event subscription in order to process them a homogeneous manner. However, this transformation requires processing time and restricts the engine's high throughput. Subscriptions to OPC-UA and DPWS are covered with rich clients in the UI; nonetheless, the monitor relies on the SUO's event description. The monitor can only see what the SUO informs. If a SUO does not provide rich variety of events, the monitor is incapable of executing calculations and inferences. On the other hand, the monitor can navigate on the SUO's events and form rules that can be derived into KPI calculation or flaw detection. Also if the SUO does not have a certain event, the monitor can generate this notification by aggregating other associated events.

All components were deployed over the same CPU hindering the processing power. However, the main focus of this work consists of aggregation of heterogeneous information and performance is more suitable for future work rather than results for this implementation. Orchestration engine as well as the monitor were deployed on a single i7 2.53 Mhz CPU with 4 GB of RAM. Due to that only 30 evt/sec were able to be processed with this implementation. In any case, the expected quantity of external events for this test consists of 0.5 evt/sec. Due to that, the system was reliable to apply further tests to solve the use case problem. If needed, future distribution of the monitor into a dedicated server may improve the performance of the overall processing power.

Having WS-output events requires building one-way operations for each rule defined in the event processor. This implementation only allows up to two rules that give two complex events. The operations are fixed, however it suffices the proof-of-concept for the use case defined.

5.1.2 Lap time test results

The EPL statement defined in section 4.2.3 detects every lap sequence and triggers successfully the configured action. The processing engine calculates and notifies the aggregated data into a visual dashboard. Results show that the monitor can aggregate and abstract events in a non-intrusive manner. Aggregated data from incoming events is shown in Figure 46. The 6th lap time shows a neglectable increment in the average time probably due to the non-deterministic behavior of the network.

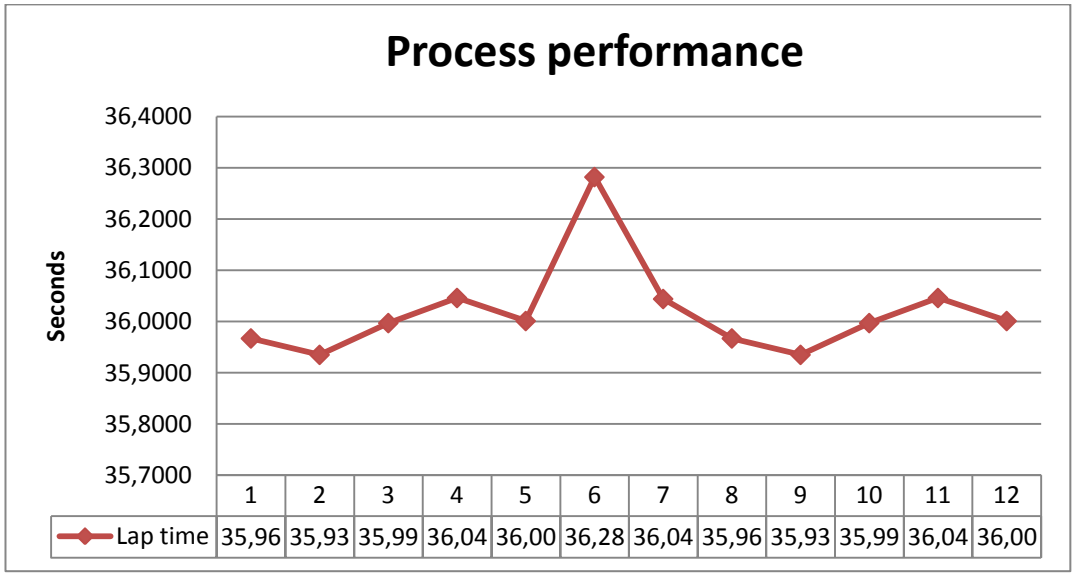


Figure 46: Data aggregation for lap time calculation

Results show an average value of 36.1 seconds per lap. The monitor is able of measuring performance of the line. However, there has to be a transition analysis to understand the behaviors of each pallet transaction and validate if the lap time results are correct.

5.1.3 Average lap test results

This test can be considered as a follow up of the previous test. The action registered to the EPL defined in Program 3 is triggered every five times the *lapTime* rule is matched. Aggregated values are successfully displayed on a dashboard showing average times. Figure 47 shows the dashboard displaying a sample aggregated value for the average lap times. As seen from Figure 46, from lap 1 to lap 5 the values are around 35.98 seconds giving a coherent value with the previous test results.

Recursiveness of the system allows having rules only for detection while other rules can be use for action. In this way, less action listeners will be required. This feature of Nesper is located in the ESP part of the API. The event generated by *lapTime* is inserted into the stream generated in this case *lapTime* stream which any EPL statement can query.

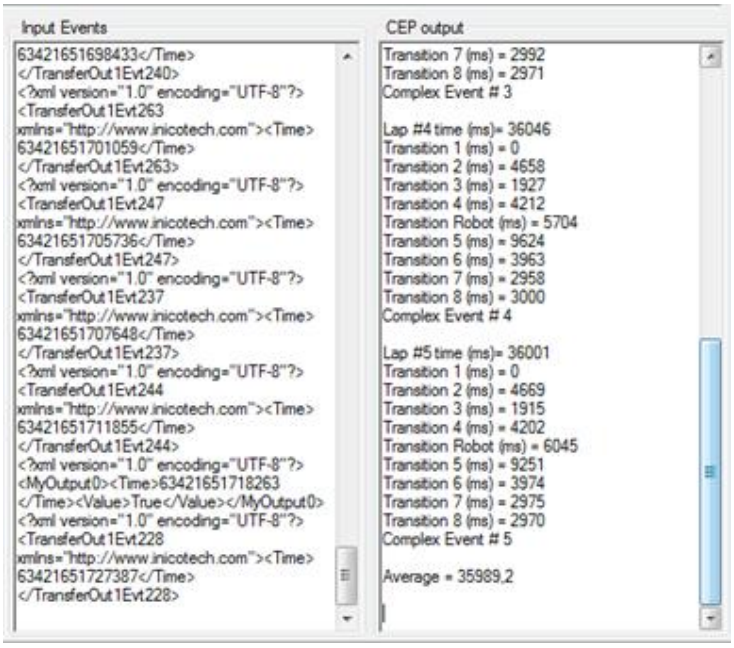


Figure 47: CEP output Visual dashboard

5.1.4 Flaw detection test results

Finally this test aims to validate the results of the first test. The *lapTime* rule successfully filtered the information needed from the *ItemTransferOut* events of each conveyor segment. The action triggered calculated the transition times of each conveyor interaction showing some considerable delay in the 5th transition as shown in Figure 48. Similar conveyor transactions returned an average of four seconds for transaction. The threshold of 8 seconds was broken on the fifth transaction, triggering a notification. This output showed that results of the first test were not optimal. Using the information from the complex event, two mistakes were found after a brief inspection of the conveyor system. The first being a misalignment of the conveyor, making the pallet to slip for some seconds before being successfully transfer. And other located within the controller that contained a WAIT statement placed by mistake. Corrections were made and a second run was performed to compare results. As expected the system showed a significant change on the 5th transition, lowering the transaction time around three seconds as shown in Figure 49.

As seen from the results, the monitor cannot specifically tell the right location of the flaw. The events generated by the SUO are just expressive enough to threat the conveyor as a black box where a flaw exists but no specific cause is found. Therefore the monitor's flaw detection capability is directly dependant on the expressiveness of the SUO.

Figure 48: Flaw detected on the 5th transition of the pallet

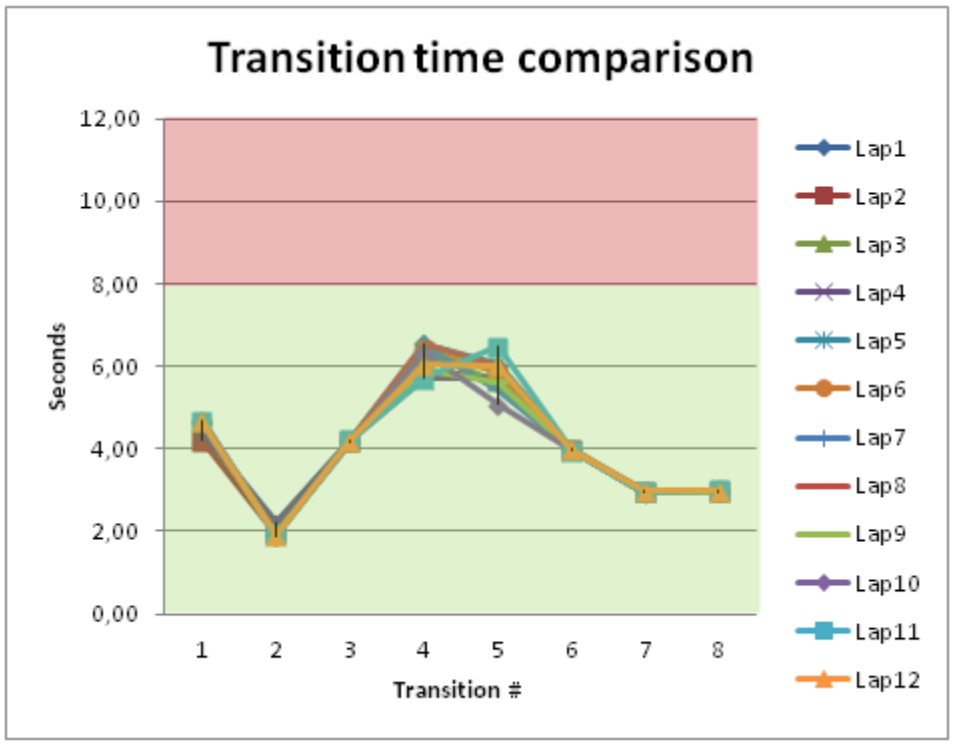


Figure 49: Transition times after system correction

5.2 Conceptual results

Disparity between problems stated and results are presented in order to corroborate the usefulness of the outcome of this work. This section presents the conceptual results obtained by answering each of the questions stated in chapter 1.

- *How to simplify integration of different heterogeneous information sources into a single event-processing system?*

As shown by this work, event-driven approaches are under the spotlight within the research community. Web-based protocols for system integration in manufactory also follow this tendency. Due to this, heterogeneous integration of event sources can be tackled in different levels. Having direct contact with the information sources makes the middleware protocols for system integration, such as OPC-UA and DPWS, to be located in the first level. Legacy systems and devices can be exposed in a standard format to others using these protocols. However, interoperability among protocols is not yet complete. Complex event processing architectures allow heterogeneous systems to connect via input adapters. These input adapters can be considered as a second level for heterogeneous system integration. Translation and homogenization operations convert event formats into a processable format for a CEP. These operations can be made within an input adapter, or via an ESB. Both approaches can be valid, however, study of this work shows that ESB, would be an overstated solution in a factory shop-floor level due to the integration capabilities of DPWS or OPC-UA based on middleware. Even so, the work presented had to combined concepts from both approaches. ESB solutions provide translation features that allow messages to be homogenized as well as a standard connectivity approach for high-level systems. Input adapters are customizable; but still do not provide a standard method for message transformation. Overall, the implementation presented achieved heterogeneous connectivity on a factory shop level. Scaling the automation pyramid for integration with the event-processor is no longer required due to the trends that communication technology is following. The levels of automation have no longer boundaries with one to another. This means that hierarchical view for integration is no longer necessary due to the omnipresence of information exposed by Message Oriented Middleware (MOM). Due to this, further expansion to systems located in higher-layers would only require of an input/translation implementation to integrate external systems to the event processor.

- *How to automate event aggregation to provide new higher level event generation?*

Automatic event aggregation refers to a skill of a system to perform calculation of incoming events by its own capabilities. The state-of-the-art presented shows that event aggregation is currently managed by rule engines. CEP provides reactivity

features to a system allowing it to be aware of situations and generate alarms and high-level events that notifies other systems

Higher level events are commonly constituted by low level events. The main difference between these two is that low level events contain data while high level events contain information. Data becomes information when set on a certain context. CEP provides context-sensitive rules from which situations can be detected. For instance, gathering information from simple timestamp value and aggregating it into production performance KPI's is an example of converting data into information within a temporal context for production. In this case, the architecture developed during this case allows higher event generation by defining rules containing operations and patterns that offer the user capabilities to add rules to provide context and transform data into information.

- *How to leverage the factory-shop floor information?*

Similarly as the previous question, factory-shop floor information can be leveraged by converting low-level events such as sensor data into high-level events. Rules apply aggregators and operations to transform data received into events and alarms which notify on system changes within a context. Complex event detection solves the "IT-blindness" caused by factory-shop floors by digesting the information to other interested systems. At the same time, this technique allows reduction of the event cloud volume into a more manageable and rich event cloud. In this work, results showed an event cloud reduction rate of 1/9. This means that from 9 low-level events containing data, one high-level event was produced. The significant reduction of this quantity allows other systems to obtain messages when relevant; this diminished the quantity of messages in the network adding reliability into the monitoring system.

- *What components could create a framework that can allow event management?*

Event managers perform the task of composing, route and, process events in a message oriented system. Four generic components were identified in this work to create a fully functional event manager. To begin with, the first component has to handle subscription and translation operations. Subscription as well as homogenization of events allows processing engines to perform operations and further distribution of events. These operations have the possibility to be implemented completely within input adapters, or act in combination with an ESB as backbone for event input and transformation. Independently of the selected approach, the input adapter is vital components for the framework. Furthermore, the second component identified consists of configuration interface. Commonly, event managers require of offline configuration that define the operations and flow routes that events must follow inside and outside the framework. Third component consists of the processing engine. The engine loads rules configured and applies them to each

event received. This components as seen in the results, takes charge of the runtime and even real-time aggregation and correlation. Without such component, the actual event generation and routing is not possible. The fourth and last component consists of output adapters. Feeding high-level events to consumers is the end goal of the event manager. Similarly as the input adapter this component must translate the events generated to an understandable format for external systems.

In summary, as shown in the architecture described within this work, these four generic components are the minimum requirements for the successful deployment of an event management platform. As results showed in the previous chapter, the tests performed successfully demonstrate data aggregation, event routing and event acquisition. Moreover, other concepts such as recursiveness and causality showed potential in future implementation and research for situational and context awareness.

6. Conclusions

This chapter evaluates the results and approaches taken during this work. In addition it presents future work and research opportunities that can extend this work.

6.1 Implementation conclusions

Due to years of experience and dominance in the manufacturing market, OPC is a *de facto* specification for interoperability of systems in factory-shop floor systems. OPC-UA is constantly expanding and evolving by joining forces with other specifications such as MTConnect and PLCopen. However, its architecture defies the concept of the internet of things where each device can be accessed directly by any system without the need of a gateway such as the one OPC-UA introduces. For that case, DPWS enters the scene by including potential to each component to express their operations using cross-platform Web Services. In other words, both specifications provide the two most important means of communication in the factory shop floor level. Furthermore, instead of being considered as competitors they should be seen as complements due to their strengths and drawbacks. The implementation presented successfully provides a mean for interoperability between these two major specifications in a monitoring perspective. The implementation breaks the boundaries between the specifications by marshalling their message outputs with a processing unit that leverages information. High-level systems currently rely on legacy services and connectors meaning that the transition towards a fully DPWS and OPC-UA compliance is complicated. A middleware implementation such as a Complex Event Processor has been proved to convey all the information into high-level events that can be seen by other systems without wasting their processing power deciphering and integrating thousands of messages generated by factory shop floors. This justifies the means of this implementation proposal to aggregate and convert the message into an internal message, due to the fact that not all systems posses capabilities to understand each other. Moreover CEP provides tools for transformation, routing, processing and generation of messages which allows it to re-structure any message. Additionally, CEP allows messages from DPWS can be sent to OPC-UA and vice versa proving the concept of interoperability between specifications in an application level.

6.2 Result conclusions

The use case presented has demanded from the monitoring system to receive and process semantic data incoming from heterogeneous systems in a factory shop environment. The test results support the assumptions taken during the architectural design of this implementation achieving all of the goals imposed. The outcome shows the monitor reacting instantly whilst preventing information lag for notifying the responsible and handling the situation accordingly.

Data aggregation becomes a crucial factor as exposed by the results of this work. A substantial amount of messages are generated in the factory floor level. Not all of the messages generated are relevant for a monitor or other system. These messages as seen in the tests can be aggregated and leveraged for other systems whilst reducing message transactions that cause networks to overload.

Additionally, it can be concluded that production losses can be decreased by increasing system's reactivity. Detecting transfer delays between pallets can save several seconds during production cycles. KPI values that evaluate the status of the manufacturing line trigger alarms that allow corrections. The corrections performed translate into reduction of production cycles can be at the same time translated into several thousand Euros saved per batch in terms of energy and efficiency.

Finally, the CEP-monitor implementation has been successfully deployed and tested within a manufacturing environment. The added value of this work can be summarized in increased reactivity, awareness, and, improved user experience for configurability of business rules.

6.3 Future work and final thoughts

This work boundary consists in the interaction of OPC-UA and DPWS in a shop-floor level monitoring. Further incorporation of Business Integration Solutions (BIS) such as ESB can leverage all of the monitoring messages into ERP, MES, or other production systems allowing supervisory control to be more reactive, whilst at the same time allowing the system to react to more complex situations due to the visibility reach of the monitor.

Rules and actions can be validated for syntactical errors; however semantic errors cannot be detected. This restricts the configuration user to input rules that are semantically correct. Rule validation in this monitor can be only done via trial and error. Further formalization and validation methods are needed for testing rule behaviors on the monitor.

Finally, Event-driven self-corrective systems could also be derived from the capabilities of CEP. Its real-time processing capabilities open the possibility to use it as an event-driven compensating control component for system corrections. This could be further evaluated on a real-time demanding situation for self-corrective actions that could increase process reliability.

REFERENCES

[Agarwala & Schwan 06] S. Agarwala, K. Schwan, "SysProf: Online Distributed Behavior Diagnosis through Fine-grain System Monitoring", Distributed Computing Systems, ICDCS 2006. *26th IEEE International Conference on Distributed Computing Systems*, pp. 8, 2006

[Balasubramanian et al. 09] K. Balasubramanian et al., "Remote control of digital factory through web", SSST 2009. *41st Southeastern Symposium on System Theory*, pp.368-372, 2009

[Barbon et al. 06] F. Barbon et al., "Run-Time Monitoring of Instances and Classes of Web Service Compositions," in *IEEE International Conference on Web Services* (ICWS'06), 2006.

[Barringer et al. 04] H. Barringer, A. Goldberg, K. Havelund, and K. Sen, "Program Monitoring with LTL in EAGLE", in *Proceedings of 18th International Conference on Parallel and Distributed Processing Symposium*, 2004.

[Bayer 09] Bayer, Gerhard, "The Synergy of SOA, Event-Driven Architecture (EDA), and Complex Event Processing (CEP)", presented at the International SOA conference 2009.

[Bernhard 02] S. Bernhard, "On-line and indirect tool wear monitoring in turning with artificial neural networks: A review of more than a decade of research", *Mechanical Systems and Signal Processing*, Volume 16, Issue 4, July 2002, Pages 487-546

[Bodden 05] E. Bodden, "J-LO A Tool for Runtime-Checking Temporal Assertions," Master's thesis, RWTH Aachen University, 2005.

[Bohn et al. 06] H. Bohn et al., "SIRENA – Service Infrastructure for Real-time Embedded Networked Devices: A service oriented framework for different domains", ICONS, pp. 43, 2006

[Brenna et al. 07] L. Brenna et al., "Cayuga: a high performance event processing engine", *International conference on Management of data* (SIGMOD '07).

[Burke 06] T. J. Burke, "OPC Foundation – OPC DevCon", OPC DevCon, 10-12 October 2006, Munich, 2006.

[Cachapa et al. 10] D. Cachapa et al., "Monitoring functions as service composition in a SoA-based industrial environment", IECON, *36th Annual Conference on IEEE Industrial Electronics Society*, pp.1353-1358, 2010

[Cannata et al 08] A. Cannata et al., "SOCRADES: A framework for developing intelligent systems in manufacturing", IEEM, *IEEE International Conference on Industrial Engineering and Engineering Management*, pp.1904-1908, 2008

[Carvalho & Simoes 11] N. Carvalho, A. Simoes, "OML: A scripting approach for manipulating ontologies", CISTI, *6th Iberian Conference on Information Systems and Technologies*, pp.1-6, 2011

[Cassandras & Lafortune 99] C.G. Cassandras, S. Lafortune, “Introduction to Discrete Event Systems”, *The Kluwer International Series on Discrete Event Dynamic Systems*, Kluwer Academic Publishers, September 1999.

[Collado et al. 08] Collado, E. M., Cavia Soto, M. A., Delamer, I. M.,Lastra, J.L.M., "Embedded XML DOM Parser: An Approach for XML Data Processing on Networked Embedded Systems with Real-Time Requirements", Journal on Embedded Systems., 2008

[Colombo et al. 10] G. Candido et al., “SOA at device level in the industrial domain: Assessment of OPC UA and DPWS specifications”, INDIN, *8th IEEE International Conference on Industrial Informatics*, pp.598-603, 2010

[Cormen et. al 01] T. H. Cormen et al., “Introduction to Algorithms”, *Second Edition*. MIT Press and McGraw-Hill, Section 22.3: Depth-first search, pp. 540–549, 2001

[DAWAC 05] IAEA, “Data processing technologies and diagnostics for water chemistry and corrosion control in nuclear power plants (DAWAC)” IAEA-TECDOC-1505, Vienna, 2006

[Delamer & Lastra 06] Delamer, I.M., Lastra, J.M.L., "Self-orchestration and choreography: towards architecture-agnostic manufacturing systems", Advanced Information Networking and Applications, AINA’2006

[Delamer & Lastra 07-1] Delamer, I.M, & Lastra, J.M.L. "Evolutionary multi-objective optimization of QoS-Aware" Engineering Applications of Artificial Intelligence ,pp. 593-607, 2007

[Delamer & Lastra 07-2] Delamer, I.M., Lastra, J.M.L., "Loosely-coupled Automation Systems using Device-level SOA", IEEE International Conference on Industrial Informatics ,Vol.2, pp. 743-748, 2007

[de Souza et al. 08] L. de Souza et al., “SOCRADES: A Web Service Based Shop Floor Integration Infrastructure”, *The Internet of Things*, Springer Berlin / Heidelberg, Vol., no., 4952, pp.60-67, 2008

[Eckerson 11] W. Eckerson, “Performance Dashboards: Measuring, Monitoring, and Managing your business”, Book, John Wiley & Sons, 2nd ed., 2011

[Eftimov 06] E. Eftimov, “Complex Event Processing – An Emerging Paradigm in Business Intelligence, Security and Monitoring and Control”, iSec Consulting, 20

[Fei et al. 06] L. Fei, K. Lee, F. Li, and S. P. Midkiff, “Argus: Online Statistical Bug Detection,” in *Proceedings of Fundamental Approaches to Software Engineering 2006* (FASE’06), pp. 308–323, Springer-Verlag, 2006.

[Goodloe & Pike 10] A. Goodloe, L. Pike, “Monitoring Distributed Real-Time Systems: A survey and future directions”, Available at: NASA Center for AeroSpace Information, 2010

[Grubic et al. 08] S. Grubic et al., “A Survey on Testing and Monitoring Methods for Stator Insulation Systems of Low-Voltage Induction Machines Focusing on Turn Insulation Problems”, *IEEE Transactions on Industrial Electronics*, vol.55, no.12, pp.4127-4136, Dec. 2008

[Gyllstrom et al. 08] D. Gyllstrom et al., “On Supporting Kleene Closure over Event Streams”, ICDE, *IEEE 24th International Conference on Data Engineering*, pp.1391-1393, 2008

[Han 03] Y. Han, Y.H. Song, “Condition monitoring techniques for electrical equipment-a literature survey”, *IEEE Transactions on Power Delivery*, vol.18, no.1, pp. 4- 13, Jan 2003

[Hannelius et. al 08] T. Hannelius et al., “Roadmap to adopting OPC UA”, 2008. INDIN 2008. *6th IEEE International Conference on Industrial Informatics*, pp.756-761, 2008

[Hardy 08] S. F. Hardy, “A New Approach for The Millennium: Holistic Manufacturing”, 2008

[Havelund & Roşu 04] K. Havelund, G. Roşu, “Efficient monitoring of safety properties” , Springer Berlin / Heidelberg , *International Journal on Software Tools for Technology Transfer*, vol. 6, pp. 158-173, 2004

[Hongliang et al. 09] W. Hongliang et al., “The CAN bus monitor system for the three phase inverters,” Electrical Machines and Systems, 2009. ICEMS 2009, pp.1-4, 2009

[Hou et al. 03] T-H. Hou et al., “Intelligent remote monitoring and diagnosis of manufacturing processes using an integrated approach of neural networks and rough sets” , Springer, *Journal of Intelligent Manufacturing* , vol.14, no.2, pp.239-253, 2003

[isa 95] The Instrumentation, Systems and Automation Society, “ANSI/ISA-95.00.01-2000: Enterprise Control System Integration”, 2000.

[Izaguirre 11] M.J.A.G. Izaguirre, et al., "OPC-UA and DPWS interoperability for factory floor monitoring using complex event processing", INDIN, *9th IEEE International Conference on Industrial Informatics*, pp.205-211, 2011

[Jammes & Smit 05] F. Jammes, H. Smit, “Service-oriented paradigms in industrial automation”, *IEEE Transactions on Industrial Informatics*, vol.1, no.1, pp. 62- 70, 2005

[Jammes et al. 05] F. Jammes et al., “Orchestration of service-oriented manufacturing processes”, ETFA, *10th IEEE Conference on Emerging Technologies and Factory Automation*, vol.1, pp.8 pp.-624, 2005

[Jammes et al. 07] F. Jammes et al., “Service-Oriented Device Communications Using the Devices Profile for Web services”, AINAW, vol.1, pp.947-955, 2007

[Jardine et al. 05] A.K.S, Jardine, et al., "A review on machinery diagnostics and prognostics implementing condition-based maintenance", *Mechanical Systems and Signal Processing*, Volume 20, Issue 7, October 2006, Pages 1483-1510

[JbossCom 11] Jboss Community, "Drools expert user guide", Tool documentation, 2011

[Jovane et al. 09] F. Jovane et al., "The manuFuture road: towards competitive and sustainable high-adding-value manufacturing" Edition illustrated, Springer, pp. 12-15, 2009

[Karnoskous et al 07] S. Karnouskos et al., "Integration of SOA-ready networked embedded devices in enterprise systems via a cross-layered web service infrastructure", ETFA, *IEEE Conference on Emerging Technologies and Factory Automation*, pp.293-300, 2007

[Karnouskos et al. 09] S. Karnouskos et al., "Towards the Real-Time Enterprise: Service-based Integration of Heterogeneous SOA-ready Industrial Devices with Enterprise Applications" , *13th IFAC Symposium on Information Control*, 2009

[Kellner & Fiege 09] F. Kellner, L. Fiege, "Viewpoints in Complex event processing", DEBS,2009

[Khoshkbarforoushha el al. 10] A. Khoshkbarforoushha et al., "Towards a Metrics Suite for Measuring Composite Service Granularity Level Appropriateness", SERVICES-1, *6th World Congress on Services*, pp.245-252, 2010

[Krauze et al. 10] J. Krause et al., "Factory-wide predictive maintenance in heterogeneous environments", WFCS, vol., no., pp.153-156, 2010

[Kusunoki et al. 98] K. Kusunoki et al., "A CORBA-based remote monitoring system for factory automation", ISORC, *Proceedings. 1998 First International Symposium on Object-Oriented Real-time Distributed Computing*, pp.396-402, 1998

[Lastra & Delamer 06] Lastra, J.M.L., Delamer, I.M., "Semantic web services in factory automation: fundamental insights and research roadmap", IEEE International Conference on Industrial Informatics ,Vol.2, pp. 1-11, 2006

[Leitao 09] P. Leitao, "Agent-based distributed manufacturing control: A state-of-the-art survey, Engineering Applications of Artificial Intelligence", Volume 22, Issue 7, *Distributed Control of Production Systems*, October 2009, Pages 979-991

[Li et al. 09] B. H. Li et al., "Cloud manufacturing: A service-oriented new networked manufacturing model", *Computer integrated manufacturing systems*, vol. 16, No. 1, pp. l -9, 2010.

[Liotta 02] A. Liotta et al., "Exploiting agent mobility for large-scale network monitoring," *Network*, IEEE , vol.16, no.3, pp.7-15, May/Jun 2002

[Luckham & Vera 95] D.C. Luckham, J. Vera, "An event-based architecture definition language," Software Engineering, IEEE Transactions on , vol.21, no.9, pp.717-734

[Luckham & Frasca 98] D. C. Luckham, B. Frasca. "Complex Event Processing in Distributed System". , 1998. Print.

[Luckham 02] D.C. Luckham, The power of events, Addison Wesley , Boston, San Francisco, New York et al., 2002.

[Luckham 04] D.C. Luckham, "The Beginnings of IT Insight: Business Activity Monitoring" [Available Online] http://www.ebizq.net/topics/bam/features/4689.html, 2004

[Luckham 05] D.C. Luckham, "Why we need a new technology to manage Event Driven Systems", [Available Online] http://www.complexevents.com. 2005.

[Luckham 06] D.C. Luckham, "What's the Difference Between ESP and CEP?" Complex Event processing article , 2006

[Magid et al. 10] Y. Magid et al., " Industry Experience with the IBM Active Middleware Technology (AmiT) Complex Event Processing Engine", DEBS, 2010

[Marechaux 06] J-L. Marechaux, "Combining Service-Oriented Architecture and Event-driven architecture using an enterprise service bus", *IBM developer work*, 2006

[Microsoft 11] Microsoft SQL server 2008 documentation, [Available online] http://msdn.microsoft.com/enus/library/ee391536.aspx

[Ming et al. 09] L. Ming et al., "Event-driven service oriented framework for integrative serviceability management of networked manufacturing systems," AIM, IEEE/ASME, pp.392-397, 2009

[monALISA 04] C. Legrand, "MonALISA: An Agent based, Dynamic Service System to Monitor, Control and Optimize Grid based Applications", *Proc. Of the Intl. Conf. on Computing in High Energy and Nuclear Physics*, pp. 907-910, 2004

[Moritz et al. 09] G. Moritz et al., "Devices Profile for Web Services in Wireless Sensor Networks: Adaptations and enhancements", ETFA, pp.1-8, 2009

[Moritz et al. 10] G. Moritz et al., "encDPWS – message encoding of SOAP Web Services", PERCOM , pp.784-787, 2010

[Moritz et al. 10-2] G. Moritz et al., "Encoding and Compression for the Devices Profile for Web Services", WAINA , pp.514-519, 20-23 April 2010

[OASIS 06] OASIS, "Reference Model for Service Oriented Architercture 1.0", OASIS standard, 2006

[OPC-UA 6] OPC Foundation, "OPC UA Part 6 – Mappings Specification " , http://www.opcfoundation.org, 2006

[OPC-UA 3] OPC Foundation, "OPC UA Part 3 – Address Space Model 1.00 Specification", http://www.opcfoundation.org, 2006

[OPC-UA 4] OPC Foundation, "OPC UA Part 4 – Services 1.0 Specification", http://www.opcfoundation.org, 2007

[OPC-UA 5] OPC Foundation, "OPC UA Part 5 – Information Model 1.00 Specification", http://www.opcfoundation.org, 2006

[Panetto & Molina 08] H. Panetto, A. Molina, "Enterprise integration and interoperability in manufacturing systems: Trends and issues", *Computers in Industry*, Volume 59, Issue 7, Enterprise Integration and Interoperability in Manufacturing Systems, September 2008, Pages 641-646

[Park 10] J-I. Park, "A smart factory operation method for a smart grid", CIE, pp.1-5, 2010

[Pellizzoni et al. 08] R. Pellizzoni et al., "Hardware runtime monitoring for dependable cots-based real-time embedded systems", RTSS'08: Proceedings of the 2*9th IEEE Real-Time System Symposium*, pages 481–491, 2008.

[Perrochon et al. 99] L. Perrochon et al., "Event Mining with Event Processing Networks", PAKDD'99, LNAI 1574, pp.474-478, 1999

[Philippe et al. 00] J. Philippe, et al., "Two Taxonomies of Distributed Network and System Management Paradigms," Emerging Trends and Challenges in Network Management, S. Erfani and P. Ray, Eds., Plenum, 2000.

[Qichao et. al] L. Qichao et al., "Metamodel Recovery from Multi-tiered Domains Using Extended MARS," COMPSAC, *IEEE 34th Annual on Computer Software and Applications Conference*, pp.279-288, 2010

[Rosales et al. 10] P. Rosales et al., "Leveraging business process management through complex event processing for RFID and sensor networks," CIE, *40th International Conference on Computers and Industrial Engineering*, pp.1-6, 2010

[Schleipen 08] M. Schleipen, "OPC UA supporting the automated engineering of production monitoring and control systems," ETFA, IEEE International Conference on Emerging Technologies and Factory Automation, pp.640-647, 2008

[Schroeder et al. 08] K. Schroeder et al., "A Factory Health Monitor: System identification, process monitoring, and control", CASE , *IEEE International Conference on Automation Science and Engineering*, pp.16-22, 2008

[Seilonen et al. 11] J. Seilonen, "Service-Oriented Application Integration for condition-based maintenance with OPC Unified Architecture", INDIN, *9th IEEE International Conference on Industrial Informatics*, pp.45-50, 2011

[Sleman & Moeller 08] A. Sleman, R. Moeller, "Integration of Wireless Sensor Network Services into other Home and Industrial networks; using Device Profile for Web Services (DPWS)", ICTTA, *3rd International Conference on Information and Communication*

Technologies: From Theory to Applications, pp.1-5, 2008

[Tanenbaum & van Steen 06] A.S. Tanenbaum, M. van Steen, "Distributed Systems: Principles and paradigms", Prentice Hall 2nd edition, 2006

[Tang et al. 07] C.S. Tang et al., "A Generic System Monitoring Technique by Using Similarity Recognition on the Flowing Entity Pattern," *Natural Computation*, ICNC, vol.5, pp.389-393, 2007

[Tibco 07] D. Adams, "Predictive battlespace", *Tibco* whitepaper, 2007

[Trinitis 00] J. Trinitis, "Interoperability Support in Distributed On-Line Monitoring Systems", *High Performance Computing and Networking* , Springer Berlin / Heidelberg , vol.1823, pp. 261-269, 2000

[Valipour et al. 09] M.H. Valipour, et al., "A brief survey of software architecture concepts and service oriented architecture", ICCSIT, 2nd IEEE International Conference, pp.34-38, 2009

[Van Hoof 07] J. Van Hoof, "SOA and EDA: Using Events to Bridge Decoupled Service Boundaries", *SOA Magazine,*: 2007

[Van Tan et al. 09] V. Van Tan et al., "A SOA-Based Framework for Building Monitoring and Control Software Systems", *Emerging Intelligent Computing Technology and Applications*, Springer Berlin / Heidelberg, Vol. 5722, pp. 1013-1027, 2009

[Vidackovic et al. 10] K. Vidackovic et al., "Business-Oriented Development Methodology for Complex Event Processing", DEBS, 2010

[Vieira et al. 03] G. E. Vieira, et al., "Rescheduling Manufacturing Systems: A Framework of Strategies, Policies, and Methods" Springer, *Journal of Scheduling*, no 6, pp. 39-62, 2003

[Virta et al. 2010] J. Virta et al., "SOA-Based integration for batch process management with OPC UA and ISA-88/95", ETFA, *IEEE Conference on Emerging Technologies and Factory Automation*, pp.1-8, 13-16, 2010

[Walzer et al. 08] K. Walzer et al., "Event-driven manufacturing: Unified management of primitive and complex events for manufacturing monitoring and control", WFCS, *IEEE International Workshop on Factory Communication Systems*, pp.383-391, 2008

[Weidong & Warren 96] C. Weidong, D.S. Warren, "Computation of stable models and its integration with logical query processing", *IEEE Transactions on Knowledge and Data Engineering*, vol.8, no.5, pp.742-757, Oct 1996

[W3C 04] W3C, "Web Services Architecture", Technical report, 2004

[W3C 01] W3C ," Web Services Description Language (WSDL) 1.1" , http://www.w3.org/TR/wsdl, 2001

[W3C 06] W3C ,"Web Services Eventing- Specification" , http://www.w3.org/Submission/WS-Eventing/, 2006

[Xiuqin et. al 09] J. Xiuqin et al., "Common and distinct neural substrates of

forward-chaining and backward-chaining syllogistic reasoning", CME, *ICME International Conference on Complex Medical Engineering*, pp.1-6, 2009

[Zackman 87] J.A. Zackman, "A framework for Information Systems Architecture" , *IBM Systems Journal*, vol.26, pp.276-292, 1987

[Zeeb et al. 09] E. Zeeb et al., "Service-Oriented Architectures for Embedded Systems Using Devices Profile for Web Services", AINAW, vol.1, pp.956-963, 2007

[Zhaohua et al. 07] R. Zhaohua et al., "AgeMoS: An Agent-Based Generic Monitoring Approach for Self-Management Systems", CSCWD, pp.452-457,April 2007

APPENDIX A – CEP platforms

PROGRESS APAMA

Apama event driven architecture is an event processing platform that is capable in a bi-directional way to handle and process messages between the platform's correlator engine and the event source. It ensures real-time operational responsiveness to fast moving events of any kind, leveraging a platform that combines flexibility, performance and interoperability. Apama consists of four main components for the design of event processing modules:

Apama's corretator: provides real-time execution of event processing scenarios, monitoring the events for patterns identified within Apama scenarios.

Apama's Studio: is an Eclipse-based Integrated Development Environment (IDE) for development, debugging, testing, profiling, back testing and deployment of Apama applications.

Apama Data player: Supports the replay of event data that has been captured in the event database. Used for complex historical analysis of the events.

Progress Apama
Features
• **Graphical development tools accessible to business users** • **Event processing language (Apama's EPL). Available natively and in Java, it delivers the deepest range of Complex Event Processing (CEP) available to the market.** • **Sophisticated analytics with native support for temporal arguments.** • **Sub-millisecond response to detected event patterns.** • **Highly scalable, patented event-driven architecture, supporting tens of thousands of scenarios.** • **Auto-generation of visually appealing user dashboards.** • **Flexible event replay for testing new event scenarios and analyzing existing ones.**
Benefits
• **Apama's Integration Adapter Framework (IAF) provides bi-directional connectivity to different event sources, messaging infrastructures and databases. IAF is complemented by a functionally-rich set of APIs for integration with customer-specific event data sources and application environments.** • **An integration environment that can synchronize RFID data streams with business events captured from different middleware systems, applications and other endpoints - without disrupting the current IT infrastructure.**
Infrastructure connectivity
• **Messaging transports: TCP/IP, UDP, CORBA, Java RMI, JMS (SonicMQ, IBM MQ Series, WebMethods).** • **Databases: ODBC, JDBC (for SQL Server, Oracle, DB2, MySQL, etc...) , KDB+ (KX Systems).**

Coral 8

Coral 8 CEP engine is designed for high-volume, low-latency applications where data analysis must occur in a question of milliseconds. It also provides important information to key in a timely fashion or drive instantaneous actions. Coral 8 architecture consists in three main components:

Coral8 Server: The Coral 8 Server is the high-throughput, low-latency runtime server for Coral8 applications. It offers features to deploy, integrate, and manage Coral8 applications. It comes with a number of packaged adapters for common high-speed data sources such as market data, messaging software and databases.

Coral8 Studio: The Coral8 Studio is a graphical environment for developing, testing, and deploying Coral 8 components and modules. The Studio also acts as a central management console for distributed network of Coral 8 Servers.

Coral8 Portal: The Coral8 Portal is a dashboard and visualization server that allows users to dynamically query and work with real-time CEP output. It offers a self-service environment that puts real-time information from CEP applications in the hands of business users.

Coral 8
Features
• **Transform large volumes of quickly changing and historical data into immediate insight, with details that drive in-the-moment decisions, recommendations, and actions.** • **Configurable enterprise-class clustering and high availability. State persistence and guaranteed messaging options can be configured. This enables fast, flexible clustering for mission-critical environments without any extra application programming.** • **Uses Continuous Computation Language (CCL) as event processing language. CCL has a SQL-like syntax.**
Benefits
• **Output is sent at the speed appropriate to the use. Continuous queries can send results or alerts to fast-moving charts.** • **Includes a large number of built-in adapters that connect to live data sources, such as market and trade data, Internet/e-commerce interactions, RFID data, transactions, sensor data, and others.**
Programming Interfaces
• Publish-Subscribe API – **used to stream event data into the CEP server, and to subscribe to output streams from the CEP server. Available for C++, Java and .NET. The Coral8 platform also includes SDKs for Perl and Python.** • User Defined Function Interface – **used to link external function libraries that can be called from within expressions in an event model running on the CEP server.** • On-Demand Query Interface – **allows you to run SQL queries against retained data sets (windows) in the CEP server. Delivers a snapshot as a response (SQL queries for an image followed by updates can be issued using the pub/sub API).**
Price

- **License cost is around 20k per core.**

Esper/Nesper

NEsper is a CLR-based component for building CEP and ESP engines. NEsper is based upon the Esper baseline, but includes customizations that are specific to the .NET CLR. NEsper was created to make it easier to build CEP and ESP applications. NEsper is open-source software available under the GNU General Public License (GPL) license. NEsper and Esper share the same grammar meaning that the two environments to be compatible

Esper supports a wide variety of event representations, such as Java beans, XML document, legacy classes, or simple name value pairs. It can be easily embedded in an existing Java application or middleware to add real-time event-driven capabilities to existing platforms without paying high serialization cost or network latency for every message received and action triggered.

Once event queries and pattern statements are registered in the Esper core container, events flow in at real-time speed and trigger arbitrary logic bound to the engine in the form of Plain Old Java Objects. This enables leveraging any existing Java technology and ensures easily connection to existing SOA building blocks.

Esper and Nesper

Features

- **Esper exceeds over 500 000 event/s on a dual CPU 2GHz Intel based hardware, with engine latency below 3 microseconds average (below 10us with more than 99% predictability) on a VWAP benchmark with 1000 statements registered in the system - this tops at 70 Mbit/s at 85% CPU usage. Esper also demonstrates linear scalability from 100 000 to 500 000 event/s on this hardware, with consistent results accross different statements.**
- **Esper is an Event Stream Processing (ESP) and event correlation engine (CEP, Complex Event Processing). Targeted to real-time Event Driven Architectures (EDA), Esper is capable of triggering custom actions written as Plain Old Java Objects (POJO) when event conditions occur among event streams**
- **Business process management and automation (process monitoring, BAM, reporting exceptions, operational intelligence)**
- **Finance (algorithmic trading, fraud detection, risk management)**
- **Network and application monitoring (intrusion detection, SLA monitoring)**
- **Sensor network applications (RFID reading, scheduling and control of fabrication lines, air traffic)**

Benefits

- **OPEN SOURCE**

Programming Interfaces

- **Esper for Java and NEsper for .NET**

Price

- **Free**

Tibco Business Events

BusinessEvents is another complex event processing (CEP) software that enables organizations to identify patterns among the event cloud that surrounds their business. It allows constructing a UML-based model to describe the applications, servers and services. The models define the relationship between assets.

The CEP rule engine is based on industry-standard RETE protocol, and it can support simultaneous application of thousands of rules to millions of events. The modeling of states of events, describes how the application and services interact as part of activities and processes. A state machine captures and stores in an in-memory database the status of events relative to causes, roles and expected behavior for instant correlation against other events. Data can persist for any length of time depending on how long an event is relevant. Figure 10 shows the Tibco's framework for event processing.

Tibco Business Events
Features
• **UML-Based Modeling: A UML-based state model describes how applications and services interact as part of activities and processes.** • **RETE-Based Rules Engine Based on industry-standard RETE protocol for familiarity and stability, the Business Events rules engine has been recompiled and tuned to support simultaneous application of thousands of rules to millions of events.** • **Events Capture Business Events can capture and process events being routed across TIBCO's integration and messaging infrastructure as well as other vendors' implementations of JMS and other integration platforms including IBM's MQSeries messaging software.**
Benefits
• **Accelerates response to threats and opportunities by automatically identifying obscure but important relationships between seemingly unrelated events before they result in situations that impact customer experience or the bottom line.** • **Improves resource allocation and problem resolution by helping organizations prioritize situations that require the most urgent attention based on a sophisticated analysis of likely outcome and secondary or indirect impacts.** • **Applications include service assurance, fraud detection, logistics, compliance and security.**

APPENDIX B – NEsper EPL

NEsper EPL statement expressiveness for rule composition

Manufacturing monitors are changing from single sensor/single indicator towards more descriptive dashboard solutions. Current HMI solutions display KPI values and process variables that are calculated by aggregating information from distributed sources. However this aggregation is not done real-time and due to this powerful aggregators and historians may be used for calculation. In the case of CEP, the aggregation is done on data arrival. Because of this, EPL statements should be expressive enough to match these aggregators functionality while allowing real-time processing. NEsper provides a rich EPL for rule definition that proves to be adequate for different conditions. The EPL rules are defined using a DSL. Due to this, NEsper as many other processors could be confused as a Deterministic Engine; However Most of CEP solutions follows the forward chaining inference method. In addition, currently there are no standard languages for event processing; this leads developers to define DSL with different focuses and levels of expressivity.

NEsper EPL follows the native structure of SQL providing the following known clauses while extending others for pattern recognition. The main clauses found in the EPL vocabulary are the explained in Table 13.

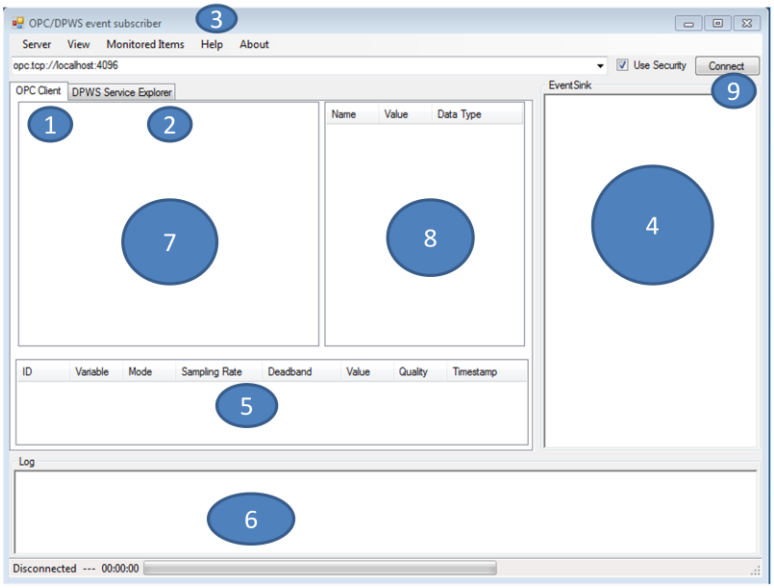
NEsper do not limit its expressiveness with lone EPL clauses, it provides time windows views, operators and expressions that extend this clauses for an improved expressiveness. For manufacturing purposes, the expressiveness of EPL statements should be adequate to calculate KPI's, detect flaws in the system, and generate alarms triggered by thresholds. To justify the expressiveness of NEper EPL in this domain, it is necessary to assume these simple cases and categorize the clauses and operators required to perform these operations.

The WHERE and HAVING clauses are used to filter events. By setting low and high parameters, this clause gives the possibility to act as KPI thresholds. The SELECT allows aggregation operators to expand its expressiveness, due to this it can be used aggregate data and calculate KPI values. Furthermore, data aggregation can be further extended by taking advantage of the Event-Condition-Action paradigm of CEP solutions. A rule triggers an action which has the aggregation business logic. This approach could be used for KPI formulas or operations where the EPL do not satisfy. Flaws from equipment and bottleneck can be detected based on status notification of equipment. The PATTERN clause offers to the business logic of the CEP the capability to detect causal logical and temporal sequences of events. Using this clause it's possible to detect and even infer flaws in the system by looking for sequences inside cloud of events.

Table 13: NEsper EPL clauses

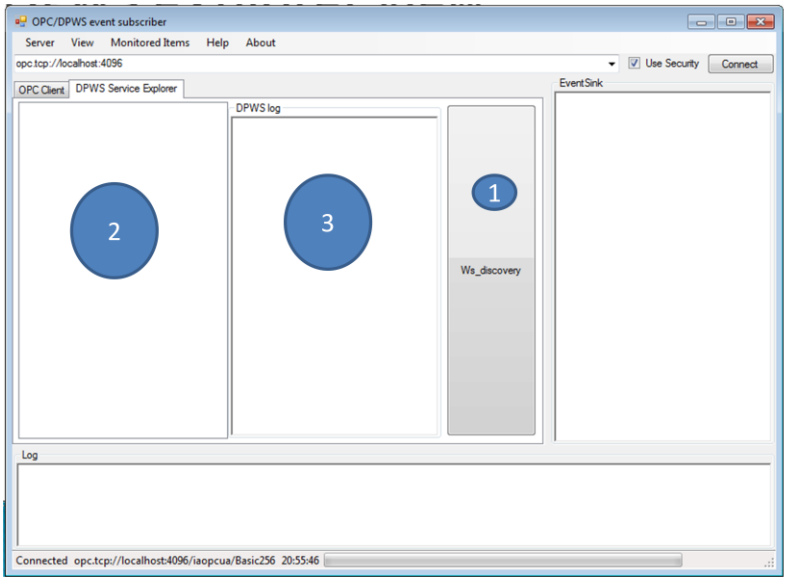
Clause	Examples	Description
SELECT	`SELECT a.custId, sum(a.price + b.price)`	Select clause grabs specific elements of an event. Aggregation is possible by using expressions common pre-define aggregators (avg(), stddev() , sum())
FROM	`FROM pattern [every a=ServiceOrder-> b=ProductOrder(custId= a.custId)`	Defines the event of interest for the rule. It is extended by the pattern cause in case of multiple events of interest.
WHERE/ HAVING	`where a.name in ('Repair', b.name)`	Filters events
INSERT INTO	`insert into DoubleWithdrawalStream select a.id, b.id, a.account as account, 0 as minimum from pattern [a=Withdrawal -> b=Withdrawal(id = a.id)]`	Input events into other streams to increase abstraction.
GROUP BY	`select symbol, sum(price) from TickEvent group by symbol having sum(price) > var_threshold`	Divides the output of an EPL statement into groups
PATTERN	`pattern [every a=ServiceOrder-> b=ProductOrder(custId= a.custId)`	Extendend by logical and/or causal or lifecycle operators for pattern matching (->, until, and, or, every)
OUTPUT	`select sum(price) from OrderEvent.win:time(30 min) output snapshot every 60 seconds`	Control or stabilizes the rate at which events are output and to suppress output events
SQL tag	`select custId, cust_name from CustomerCallEvent, sql:MyCustomerDB ['select cust_name from Customer where cust_id = ${custId} ']`	Allows combination of database results with event streams and clouds

APPENDIX C – Monitor configuration and initialization



1.- OPC Client tab
2.- DWPS explorer tab
3.- Application menu
4.- Eventsink viewer
5.- Monitored items list
6.- Log of actions performed
7.- OPC/UA address space navigator
8.- Attribute viewer.
9.- OPC Connect button

Figure 50: Event manager UI description (OPC-UA tab)



1.- Ws_Discovery
2.- Tree view of the parsed WSDL's
3.- DPWS log

Figure 51: Event manager UI description (DPWS tab)

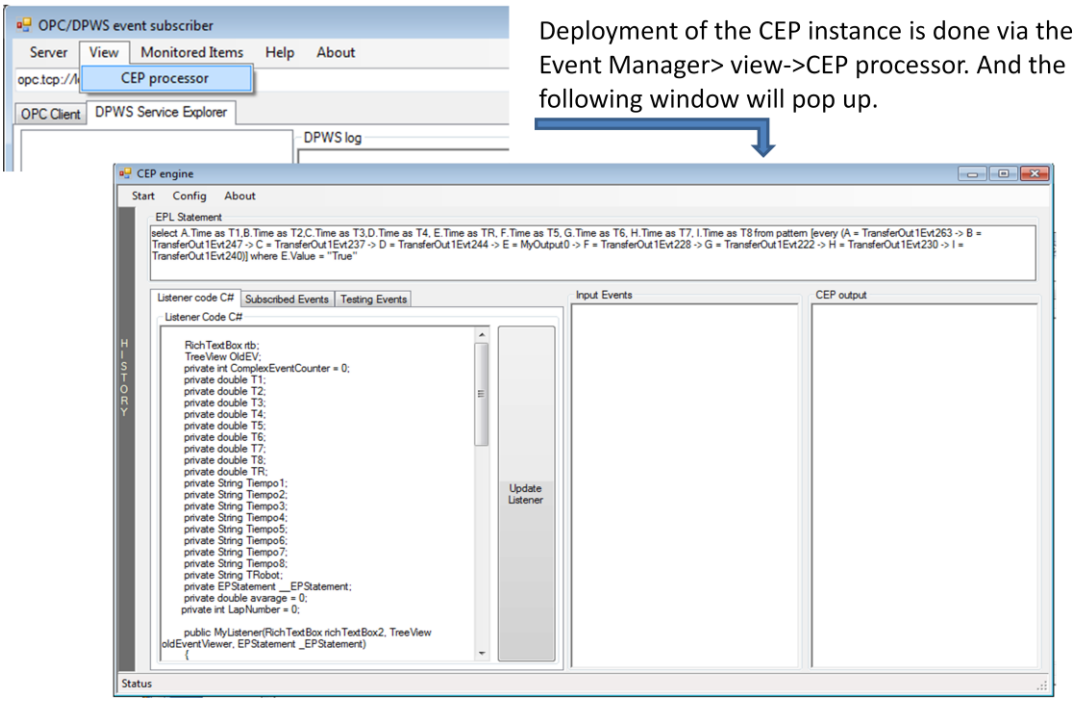


Figure 52: CEP deployment

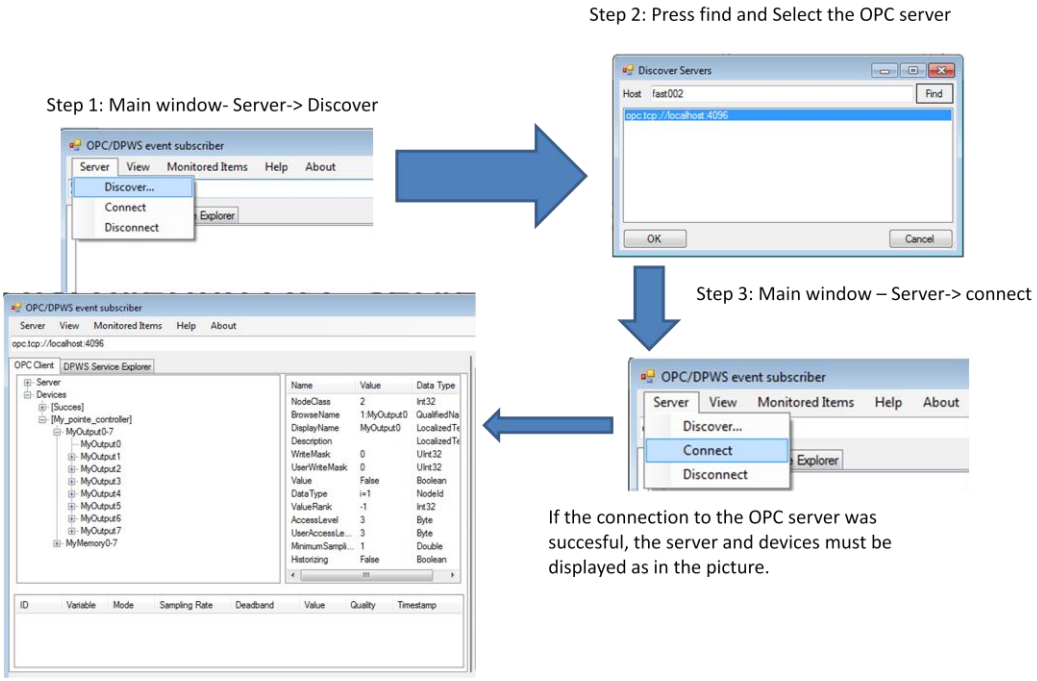
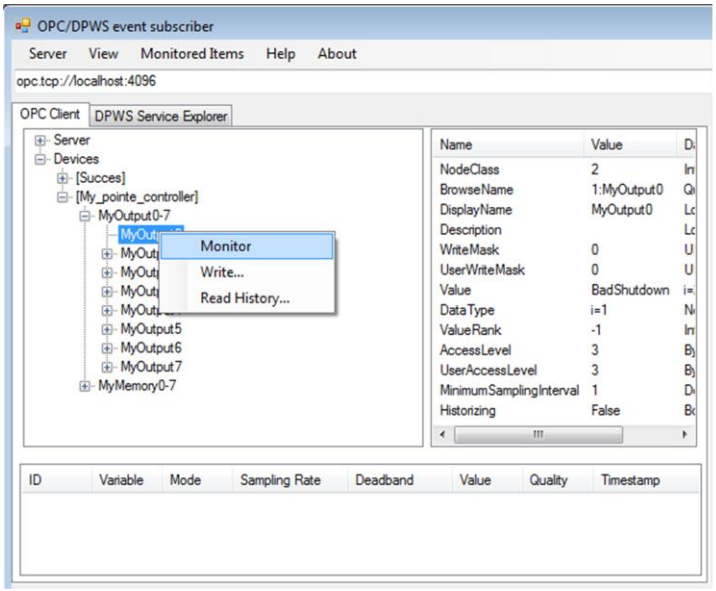


Figure 53: OPC-UA server discovery and connection



While navigating through the OPC-UA address space, it is possible to monitor by right clicking the desired event and select Monitor.

Figure 54: Subscription for OPC-UA notifications

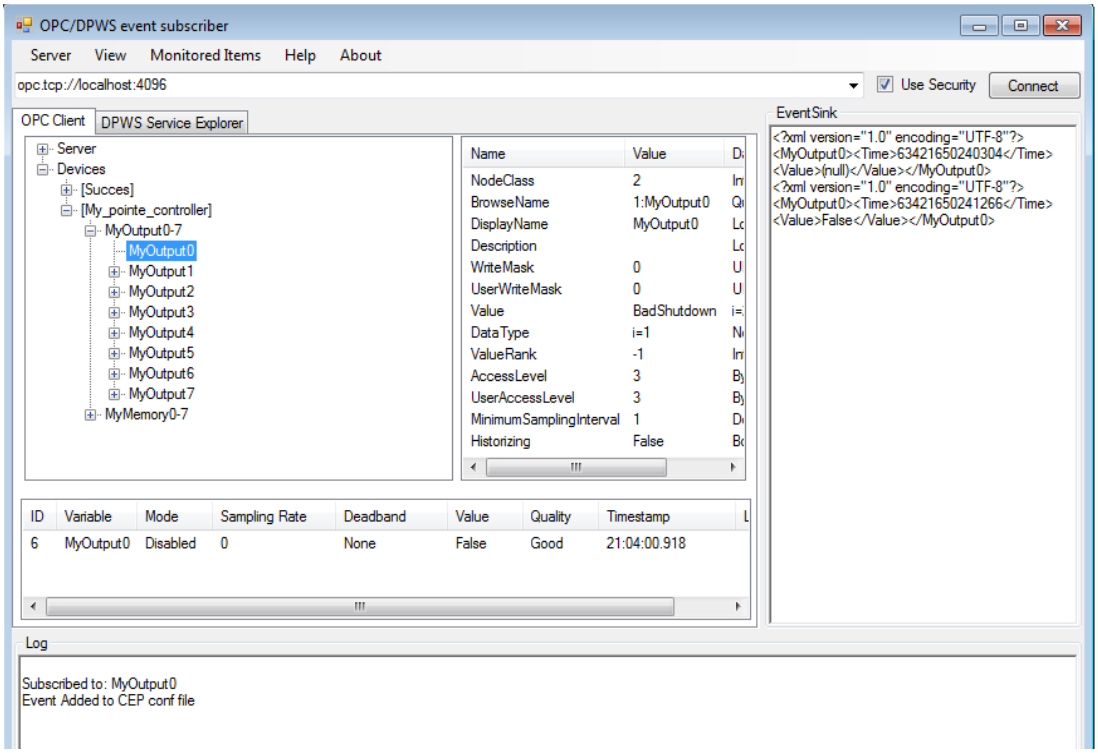


Figure 55: OPC-UA notification XML conversion

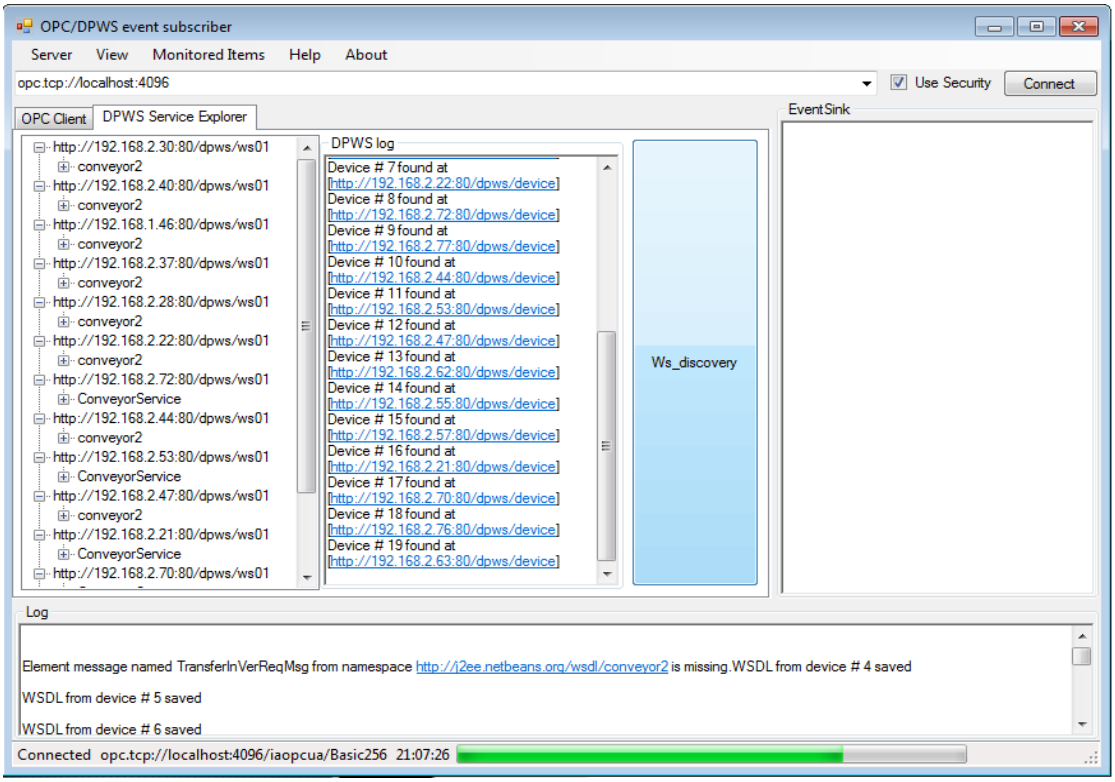
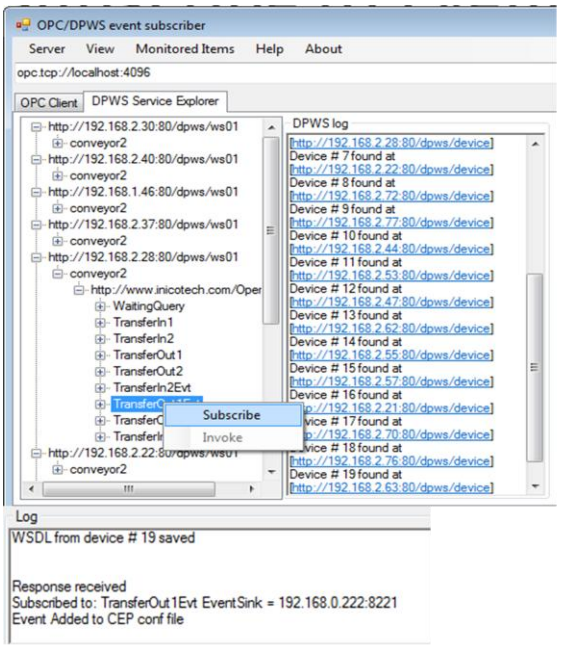


Figure 56:DPWS device discovery



In the tree view, it is possible to navigate through the devices events and services. To subscribe to the events just right click the desired event and select Subscribe. Note that the events are being registered the Subscribed Events tab inside the CEP engine window

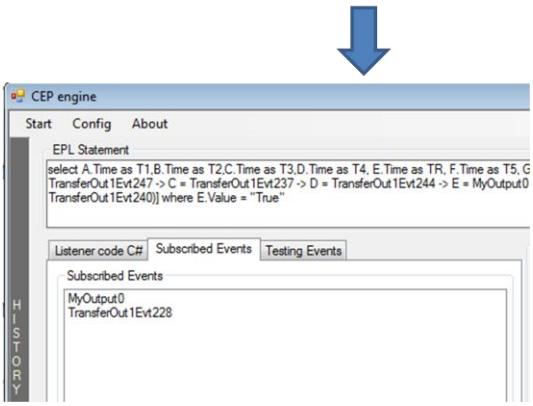


Figure 57: DPWS event subscription

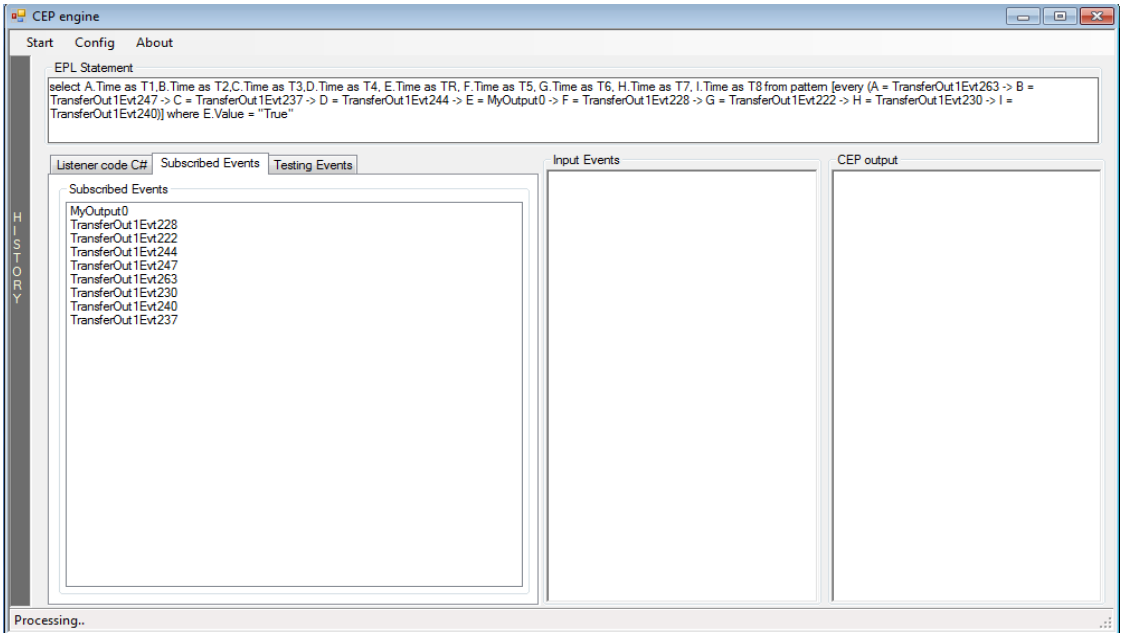
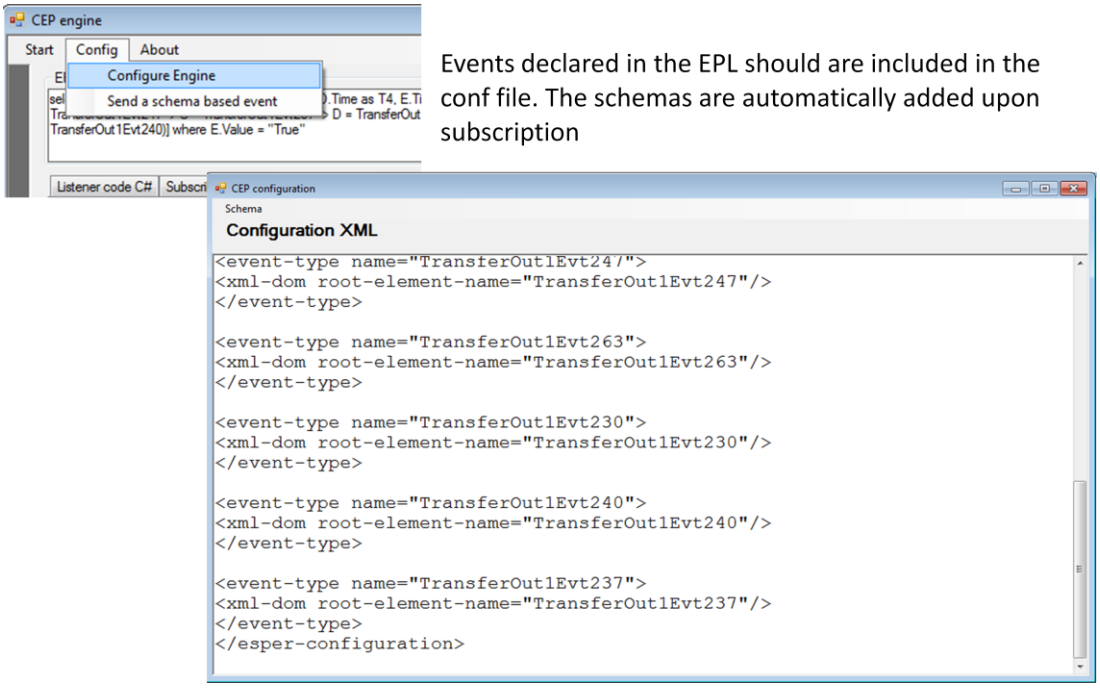


Figure 58: CEP engine UI (Event schemas loaded for EPL definition)



Events declared in the EPL should are included in the conf file. The schemas are automatically added upon subscription

Figure 59: Event type registration for CEP engine

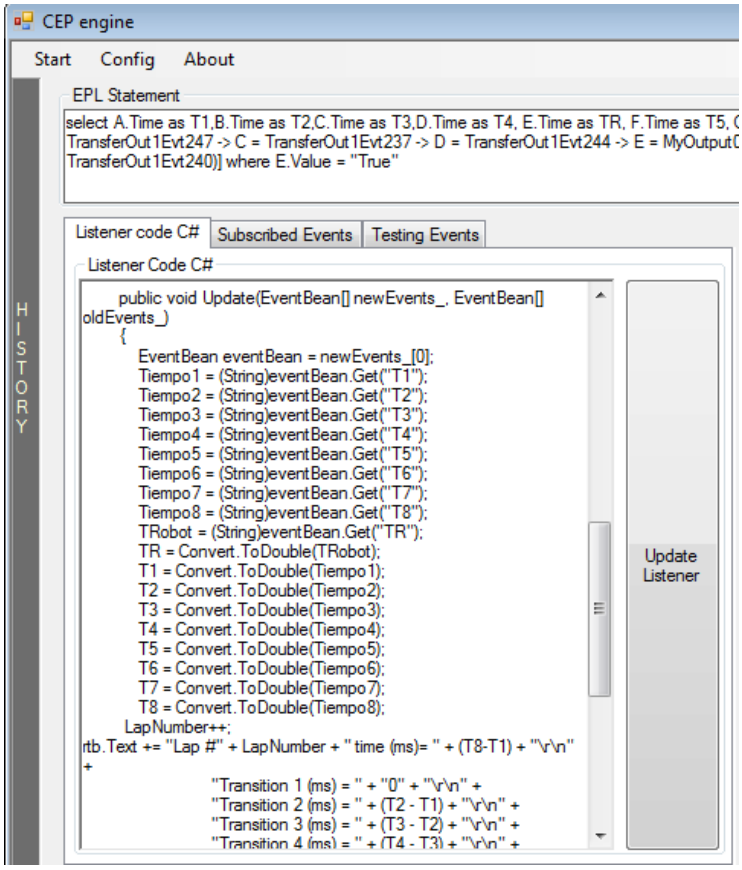
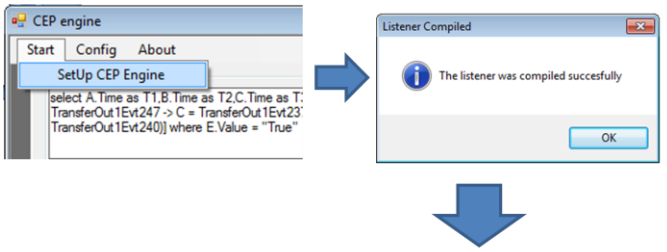


Figure 60: Rule ActionListener script UI



If the conf file, the EPL statement and the Listener code is correct, the monitoring tool gives a success message.

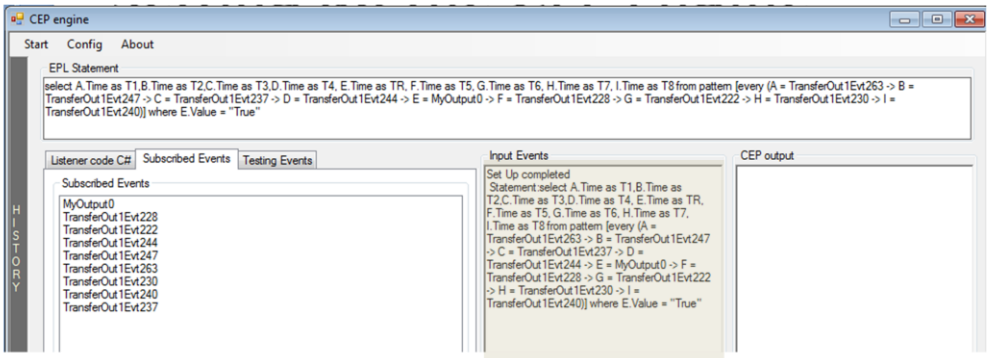


Figure 61: CEP initialization

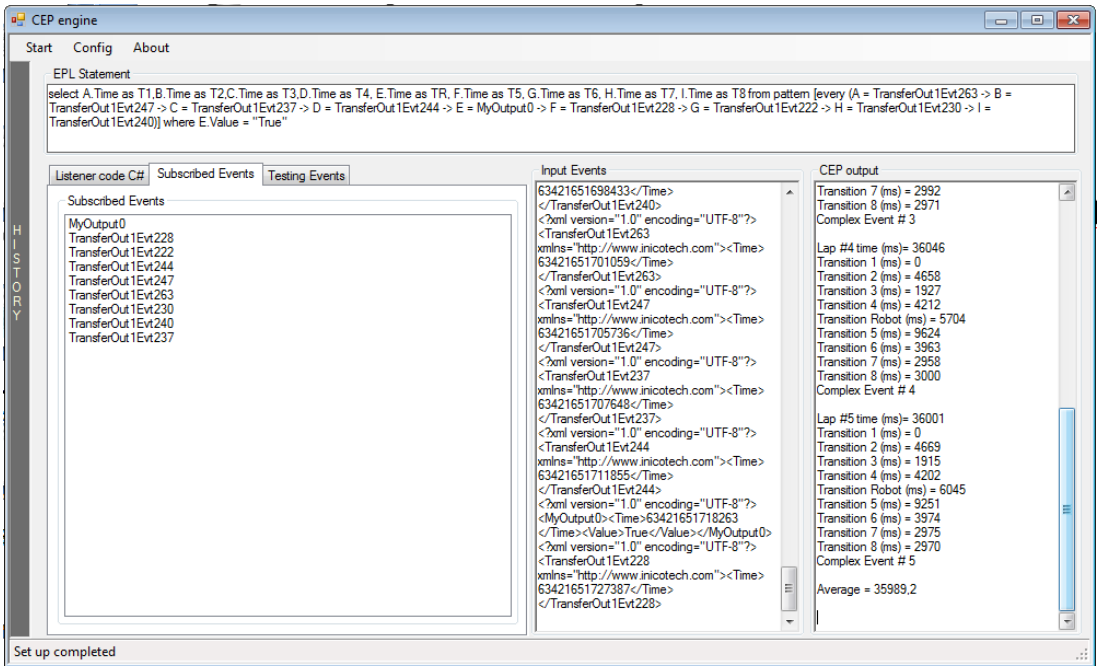


Figure 62: Complex event generation