Messaging Methods in a Service-Oriented Architecture for Industrial Automation Systems

Vlad M. Trifa*†, Dominique Guinard*‡, and Moritz Koehler*†
*SAP Research Switzerland, Kreuzplatz 20, Zürich CH-8008, Switzerland
†ETH Zürich, Institute for Pervasive Computing, Haldeneggsteig 4, Zürich CH-8092
‡ETH Zürich, Auto-ID Labs, Sonneggstrasse 63, CH-8092 Zürich
Email: `vlad.trifa@ieee.org,dguinard@guinard.org,mkoehler@ethz.ch`

Abstract— **In today's business landscape, manufacturing companies are facing a huge market pressure for reduced production time and costs. Available industrial automation systems are often based on proprietary communication protocols; therefore integration with the company's Enterprise Information Systems (EIS) is a tedious and costly procedure. As real-time visibility of devices in the organization is essential to optimize production processes, we compare and discuss two messaging methods in a Service-Oriented Architecture (SOA) for embedded devices, and contrast the advantages of the flexible event-based messaging mechanism offered by Web Services (WS-Eventing) with the more reliable and scalable, but less interoperable Java Messaging System (JMS). We finally propose several key factors that should be carefully evaluated when implementing a robust industrial messaging system.**

I. Introduction

Globalization has deeply transformed the manufacturing domain, and the perpetual race for cheaper products with decreased time-to-market is one of its main consequences. In order to remain competitive in today's market, companies need to upgrade their computing infrastructure so that is allows an enhanced end-to-end supply chain visibility which could provide timely detection and response to events on shop floors across the whole supply chain. For this purpose, system integration and seamless data exchange between partners is essential, and the service-oriented architecture (SOA) paradigm has quickly gained popularity as the most reasonable method to build robust and flexible infrastructures for connecting enterprise business applications. Nowadays, countless companies are aware of the competitive advantages brought by an efficient and reliable SOA, with Amazon, Google, or Flickr being only a few of them.

The manufacturing industry is a huge market, and also faces constantly changing demands and strong time-to-market pressure. Unfortunately, currently available systems for industrial automation suffer from the usage of proprietary communication protocols, making interoperability between manufacturing systems from different manufacturers a difficult task. As a result, the installation, customization, and maintenance of such obsolete systems often requires colossal investments, both in time and money.

Additionally, shop floor systems based on distributed embedded devices concentrate the programming logic on a few centralized mainframe computers accompanied by a large quantity of "dumb" devices. The intelligence and behaviour are tailored and individually programmed for each application, with almost no possibility for run-time reconfiguration of the different processes. One can overcome the rigid, monolithic organization of traditional enterprise applications by using a SOA, which allows applications and business processes to be modeled at the top of a cross-institutional service landscape. Data exchange between different departments and strategic units has never been as easy as it is within a SOA, however, when it comes to real-time visibility of the different assets of a company (equipment, inventory, machines, etc.), there is still a huge gap between the physical world and the high-level monitoring systems. Coupling these higher-level applications with shop floor machinery would increase the efficiency of the business processes, and the integration of device-level services with enterprise systems is essential for real-time visibility with fine granularity in the whole supply-chain. This in turn allows critical information to be delivered to the interested parties in a timely manner, so that appropriate measures can be taken with minimal impact on the production process. In particular, a judicious processing and filtering of large amounts of real-time information from the shop floor is required for applications such as business activity monitoring, where the collected information can serve for the optimization and maintenance of the equipment.

The results presented here are part of the EU-project SOCRADES [7], whose goal is to provide a set of protocols, technologies, and middleware with the required flexibility, reliability, and safety to integrate embedded systems into a company so that it maximizes the benefits it can bring to industrial processes. This article depicts our initial efforts towards a system architecture that supports seamless integration of device-level services with higher-level Web Services and business processes situated at the level of business applications such as Enterprise Resource Planning (ERP) systems. Our work differs significantly from general purpose infrastructures for distributed sensing application such as SenseWeb [6], because the requirements for industrial automation are different, and the need for automated discovery of devices, dynamic reconfiguration, and reliable messaging with hard real-time constraints are only a few of important issues for an industrial solution. For example, we propose a set of requirements for an efficient Real-Time Enterprise infrastructure and suggest how existing technologies could be leveraged to fulfill such requirements. In particular we compare two messaging methods for data exchange between low-power devices and back-end infrastructures.

Many approaches for building distributed computing applications have been proposed. Early component and object-based communication middleware, such as CORBA and DCOM were followed by more flexible architectures such as Jini or UPnP, which are more suited to highly dynamic environments such as the Internet and do not rely on specific communication protocols. To further reduce platform and programming language dependability, Web Services (WS) were proposed for implementing an SOA, and have been used to build industrial solutions for collaborative manufacturing [9], robotic cells control and configuration [3], multi-robot remote monitoring [10], or e-diagnostic [4]. Devices Profile for Web Services (DPWS) [5] is a set of Web Services for implementation constraints particularly suited for physical devices and has been used to build highly integrated systems for the SIRENA project [1]. ISA-95 is a standard for exchanging data between business and manufacturing systems, and an XML-based implementation (B2MML) compatible with web services has been proposed [2]. OPC is also the most widely adopted communication protocol in the manufacturing domain, however, few ERP systems support full connectivity with shop floor devices.

The main features for future automation systems are dynamic and automated reconfiguration of production systems, cross-enterprise cooperation, scalability, and fault-tolerance. Several requirements for the implementation of such a flexible, scalable, robust, and platform independent SOA infrastructure are identified here:

- *Logical view:* The service is an abstract representation of the functionality, and should be independent of the implementation,
- *Message orientation:* The service is defined by messages exchanged between agents rather than the properties of the agents themselves (ideally stateless),
- *Description orientation:* A service is described by machine-processable metadata which describes the interface,
- *Platform neutral:* Messages are sent in a standard platform- and language-neutral format to the services defined in the interface.

II. Tools and methods

This section describes the different units that are useful to build a robust and flexible enterprise infrastructure, and are described only at a conceptual level. Devices refer to low-power embedded systems that have a TCP/IP connection and different type of sensors attached.

In a business process, a large amount of data and events are generated on the shop floor during normal operation. Some events provide a general overview of the system status, while others can indicate unexpected issues and failures. Not all of these events provide meaningful information for business applications; therefore the processing of events (e.g. filtering or evaluation) can be done at several layers between the back-end systems and the devices, where it makes the most sense, while others are critical and need to be propagated quickly to the back-end system.We are interested in how data can be exchanged between such devices and the back-end Enterprise Information Systems (EIS).

A. Device profile for Web Services (DPWS)

A Web Service is defined as a loosely-coupled, modular, self-contained, and reusable software component that can be used to develop distributed applications over standard Internet protocols. As opposed to component technologies such as DCOM or RMI, Web Services are not accessed via object-model-specific protocols, but instead using standard Web protocols (HTTP) and interoperable data formats (XML). The W3C Web Service definition encompasses many different systems, but in common usage the term refers to clients and servers that communicate using XML messages that follow the Simple Object Access Protocol (SOAP) format to exchange data usually using HTTP/HTTPS. Web Services Description Language (WSDL) is another XML-based language used to describe Web Service interfaces, and is useful for dynamic access to a devices available services and to their metadata.

Devices Profile for Web Services (DPWS[1]) defines a set of constraints to implement secure Web Service messaging, discovery, and event notification. Unlike UPnP, DPWS is fully aligned with Web Services technology and benefits from several specifications that allow seamless integration of device-provided services in enterprise-wide application scenarios. In addition, Microsoft's Windows Vista and CE 6 platforms natively integrate DPWS support, which is a strong incentive for market adoption.

In previous work, we have implemented WS-Eventing with a temperature sensor that sends periodically its readings to an EIS system [8]. This worked well as a single sensor was used, but this would have been really problematic with multiple sensors that send high-frequency data to a unique system.

B. JMS

The Java Message Service (JMS) defines a standard for reliable Enterprise Messaging, which is recognized as an essential tool for building enterprise applications. Enterprise messaging provides a reliable, flexible service for the asynchronous exchange of critical business data and events throughout an enterprise. The JMS API adds to this a common API and provider framework that enables the development of portable, message based applications in the Java programming language.

Java Platform Enterprise Edition (Java EE) is a widely used platform for server programming using the Java programming

[1]The latest DPWS specification can be found here: `http://schemas.xmlsoap.org/ws/2006/02/devprof/`

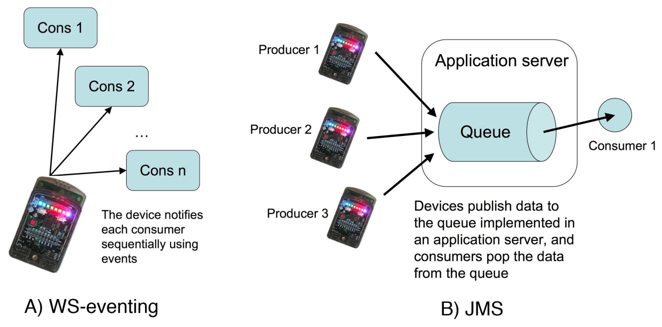
Cons 1
Cons 2
...
Cons n
The device notifies each consumer sequentially using events
A) WS-eventing
Producer 1
Producer 2
Producer 3
Application server
Queue
Consumer 1
Devices publish data to the queue implemented in an application server, and consumers pop the data from the queue
B) JMS

Fig. 1. A) WS-Eventing: each device periodically notifies each consumer with new events. B) in the point-to-point mechanism, several producers only send a single message to a queue located within an application server. The consumers then access the information from the queue.

language. The Java EE Platform proposes libraries and functionality to deploy fault-tolerant, distributed, multi-tier software, based on modular components running on an application server (AS). The addition of the JMS API enhances the Java EE platform by simplifying enterprise development, allowing loosely coupled, reliable, asynchronous interactions among Java EE components and legacy systems capable of messaging. The Java EE platform's Enterprise JavaBeans (EJB) container architecture enhances the JMS API by allowing for the concurrent consumption of messages, and by providing support for distributed transactions, so that connections to EIS systems using JMS implementations from different vendors can all participate in the same transaction context.

C. *Messaging*

Two types of inter-object messaging paradigms are considered in this article and illustrated in Fig. 1. On the one hand, the WS-Eventing specification is a publish-subscribe event handling protocol (pub/sub) that allows consumers to subscribe to a topic and be asynchronously notified by events generated by a Web Service. This is simple to implement in an SOAs, however the resulting architecture is quite complex. On the other hand, the point-to-point mechanism where producers insert their message in a queue, where it is read by only one consumer. A queue may contain more than one consumer, but the message is delivered to only one consumer. This has several advantages over pub/sub as it is more efficient, reliable, and scalable as load balancing between consumers is inherent. It is especially suited to transmit large amounts of high-frequency data, thus is very appropriate for sending sensor readings. JMS supports both pub/sub mechanism and point-to-point, but DPWS do not use a queuing mechanism and only WS-Eventing is available.

III. Discussions

The two messaging methods described earlier are not mutually exclusive, and in some cases an ingenious mix of both might exhibit superior performance and flexibility. Therefore, a list of requirements based on several factors has to be carefully crafted. One major issue would be the performance evaluation of the system, which should include variables such as the throughput (messages per second), latency under different network loads, amount of system and communication failures, and so on. Other essential criteria that need to be considered for the implementation of a messaging infrastructure are presented in this chapter.

WS-Eventing is useful for sending data across very different hardware and software platforms thanks to the versatility of SOAP messages. However, pub/sub is a very inefficient method when implemented on a low-power device. An alternative would be to use a intermediate machine that does the distribution of events to subscribers. In a JMS queue, the messages are consumed on a first come first served basis, preventing deadlocks due to blocking calls. This is especially interesting as business applications evolve towards an Internet of Things, where thousands of sensors and devices send asynchronous events to enterprise back-ends. Processing such an enormous amount of data sent using blocking RPC calls or classic SOAP webservice invocations is unlikely to be reliable enough for industrial applications.

Scalability alone is not always sufficient, and for certain business applications reliability, flexibility, and transaction control are also needed. To achieve this, concrete JMS implementations are often deployed within a Java Enterprise compliant Application Server (AS). This combination enables JMS queues to be reliable and transactional. Indeed, the use of Message Driven Beans as entry points for JMS messages enables these to be recovered after an application crash. Finally, this combination allows for distributed JMS implementations, automatically balancing the load amongst the JMS brokers.

a) Automatic discovery: Dealing with an ecosystem where devices continuously connect and disconnect from a network is not a new problem, and several solutions for automated discovery have already been proposed and commercialized, such as Apple's Bonjour, Zeroconf, UPnP, or DPWS. Automated discovery is an essential feature in any dynamic environment, therefore very convenient for simplified configuration of complex end-to-end industrial systems. An automated discovery mechanism should be simple enough to be implemented on low-power embedded devices, while offering a metadata exchange mechanism that provides information about the capabilities and offered services of the different devices.

b) Interoperability: JMS is a set of standards created by multiple vendor implementations; therefore, one avoids vendor lock-in problems, but the implementation is more complex than the WS-Eventing mechanism. JMS allows for abstraction between clients and servers; one can change the implementation of different components without changing the application layer. However, the JMS API is not always interpreted in the same way by different vendors and so subtle differences between different JMS solutions might exist. The flexibility offered by the SOAP messaging is much wider, and the interoperability is enhanced as data is exchanged using a standard format defined by the WWW Consortium.

c) Reliability: A messaging system must offer different degrees of reliability for different type of information. Industrial systems require reliable messaging when critical data has to be transmitted (e.g., alarms), because the emitter must be confident that the alarm was received and processed. The WS-ReliableMessaging specification is being implemented in most enterprise development frameworks, including those provided by Microsoft, IBM and Sun, and also should be supported at the device level. Unfortunately, reliable messaging at the application layer is not part of the DPWS specification, but is available with JMS.

d) Scalability: The number of DPWS messages that have to be sent using WS-Eventing increases linearly with the number of consumers per topic, as each consumer is notified sequentially. As a certain amount of resources is associated to send each message, this mechanism is a clear handicap as the network becomes larger, and especially when this is implemented on devices with limited resources. The JMS mechanism can queue transactions and provides an optimal load balancing, thus is much more scalable, especially when multiple messaging servers are used.

e) Security: Security is not part of the JMS specification, therefore if the application server is accessible from outside the company, security flaws in the different vendors' JMS implementations can threaten the whole system. While existing firewalls and network security can be leveraged to protect one's back-end application and database servers from security breaches, there is a significant security risk when JMS servers are directly connected to the Internet. In contrast, the WS-Security of DPWS contains specifications on how integrity and confidentiality can be enforced on Web Services messaging, for example by attaching signature and encryption headers to SOAP messages.

IV. Conclusion

In this short paper, we have briefly identified and discussed some practical requirements for the creation of simple, yet robust, software infrastructure that enable real-time visibility of shop floor devices in a manufacturing plant, which is an essential feature for enabling Real-Time Enterprise and future distributed manufacturing systems. Implementation of DPWS on every device in the system is not feasible using current state-of-the-art technologies, therefore efficient data transmission between devices and higher levels needs to be optimized. However, a detailed quantitative performance comparison of the different approaches would be necessary for concrete implementations of messaging in enterprise systems. As each system will have a particular set of requirements, only a general guideline is proposed here.

When real-time constraints are present, the usage of shared communication channels is not recommended, especially if a lot of data has to be conveyed to a central location for processing, and proprietary communication protocols are still the best option - as for example synchronization of devices in a production line. However, using intermediate gateways that can process some data locally, and then forward to EIS only the relevant information in a reliable way using open messaging protocols is a reasonable approach.

The automated discovery mechanism combined with the interoperability of DPWS could be significantly improved by using JMS to transmit large amounts of data from devices to EIS. However, JMS is complicated and is an additional layer with its set of servers. Also, more specific tools are needed to manage JMS (dedicated servers, customized monitoring and support software). Server monitoring and security issues are just a few of the problems associated with a JMS, and should be adopted only for internal messaging. Nonetheless, reliable messaging at the application level can be ensured when the queue is implemented within an application server, and JMS should be preferred when reliability is a key requirement.

A meticulous system analysis will be required for each application at hand, and JMS should be considered only if the analysis suggests that performance and scalability of the system can highly benefit from multiple JMS servers. It should also be envisaged for transmitting business critical information such as banking transactions, error alerting, etc. Many simpler alternatives can provide the requisite layer of abstraction between client and server while taking advantage of standard HTTP and XML, and offer secure and asynchronous communication, so JMS should be avoided unless its benefits clearly compensate its drawbacks.

References

[1] Hendrik Bohn, Andreas Bobek, and Frank Golatowski. Sirena - service infrastructure for real-time embedded networked devices: a service-oriented framework for different domains. In *Proc. of the International Conference on networking, Systems, mobile communications and learning technologies (ICNICONSMCL'06)*, 2006.

[2] Dave Emerson, Kawamura Haruhisa, and Matthews Wayne. Plant-to-business interoperability using the isa-95 standard. Technical Report English Edition 43, Yokogawa, 2007.

[3] Veiga G., Pires J.N., and Nilsson K. On the use of service oriented software platforms for industrial robotic cells. In *IFAC International Workshop Intelligent Manufacturing Systems (IMS'07)*, 2007.

[4] Min-Hsiung Hung, Fan-Tien Cheng, and Sze-Chien Yeh. Development of a web-services-based e-diagnostics framework for semiconductor manufacturing industry. *Semiconductor Manufacturing, IEEE Transactions on*, 18(1):122 – 135, 2005.

[5] Francois Jammes, Antoine Mensch, and Harm Smit. Service-oriented device comunications using the devices profile for web services. In *Proc. of 3rd International Workshop on Middleware for Pervasive and Ad-Hoc Computing (MPAC05) at the 6th International Middleware Conference*, 2005.

[6] Aman Kansal, Suman Nath, Jie Liu, and Feng Zhao. Senseweb: an infrastructure for shared sensing. *IEEE Multimedia*, 14(4):8–13, 2007.

[7] S. Karnouskos, O. Baeker, L. M. S. de Souza, and P. Spiess. Integration of soaready networked embedded devices in enterprise systems via a cross-layered web service infrastructure. In *12th IEEE Conference on Emerging Technologies and Factory Automation*, 2007.

[8] Moreira L., Spiess P., Köhler M., Guinard D., Karnouskos S., and Savio D. Socrades: A web service-based shop floor integration infrastructure. In *Internet of Things 2008, First International Conference for Academia and Industry*, 2008.

[9] Weiming Shen, Yinsheng Li, Qi Hao, Shuying Wang, and Hamada Ghenniwa. Implementing collaborative manufacturing with intelligent web services. In *Proc. of the 5th International Conference on Computer and Information Technology*, 2005.

[10] Bin Wu, Bing-Hai Zhou, and Li-Feng Xi. Remote multi-robot monitoring and control system based on mms and web services. *Industrial robot: an international journal*, 34(3):225–239, 2007.